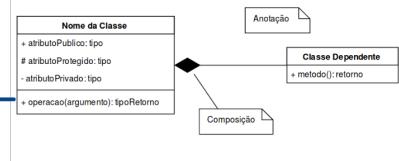


Resumo: Modelagem do Contexto do Problema - Modelo do Domínio/Negócio



1. Caracterização dos conceitos do domínio de aplicação:

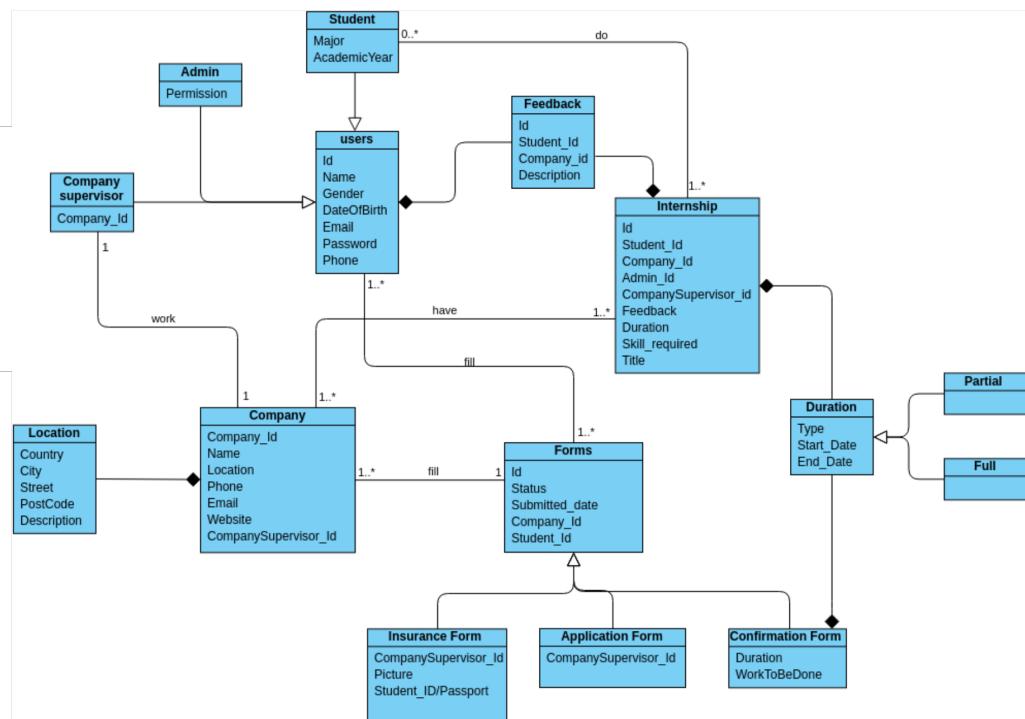
- Criação de um **diagrama de classes simples** para representar os conceitos do domínio do problema.
 - Utilização de **duas estratégias sistemáticas** para identificar conceitos candidatos ao modelo de domínio:

~~identificar conceitos~~

- a) Análise de documentos e entrevistas com stakeholders.

- b) Inspeção de casos de uso ou requisitos do sistema.

- Identificação e remoção de construções específicas de implementação que podem prejudicar a clareza do modelo (fase de análise), como atributos ou métodos técnicos não essenciais para o entendimento do domínio.



2. Caracterização dos processos do negócio/organizacionais:

- Leitura e criação de **diagramas de atividades** que descrevem os fluxos de trabalho organizacionais e os processos internos.
 - ↳ *Bom para representar*

— Bom para representar processos passo-a-passo

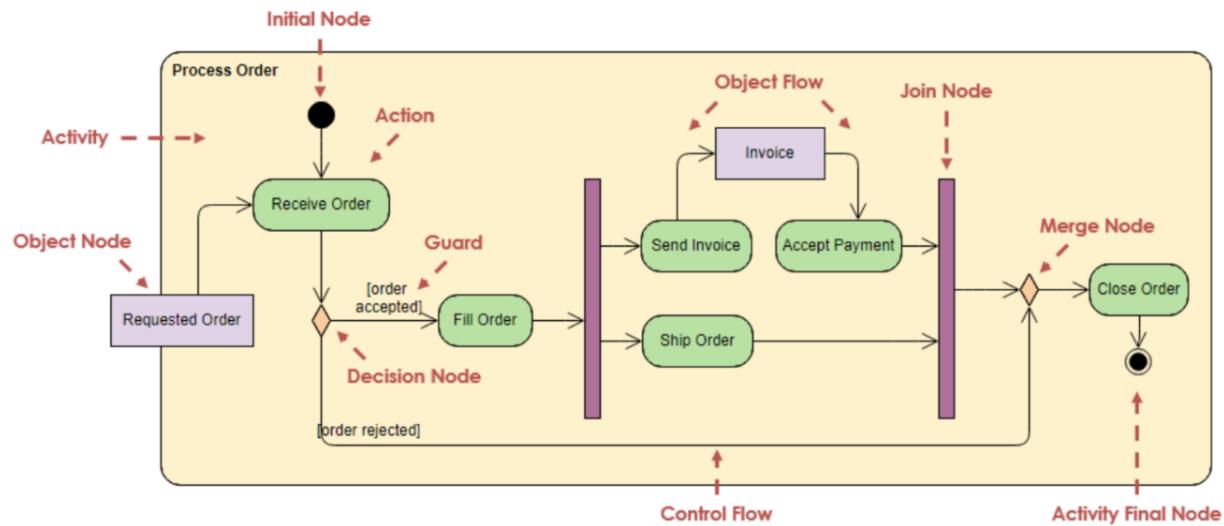
- Uso adequado de elementos de modelagem: **ações** (atividades realizadas), **fluxo de controle** (sequência das ações), **fluxo de objetos** (dados manipulados), **eventos** (gatilhos) e **partições**

Ações = tarefas executadas

Fluxo de controle = sequência lógica

**Fluxo de objetos = entrada/saída
de dados**

Partições = responsáveis (ex: departamentos)



1. Initial Node (Início do Processo)

● Círculo preto - Indica o ponto de partida do fluxo.

2. Action (Ação)

◻ Retângulo com cantos arredondados - Representa uma atividade ou tarefa realizada durante o processo.

3. Control Flow (Fluxo de controlo)

→ Seta - Indica a sequência de execução entre as atividades.

4. Decision Node (Nó de decisão)

◇ Losango - Representa um ponto de decisão com diferentes caminhos (exemplo: "Sim" e "Não" ou "Aceitar" e "Rejeitar").

5. Merge e Fork

▀ Linha horizontal ou vertical -

- Fork (Divisão Paralela): Uma linha em que depois se divide em várias linhas e as ações são executadas em paralelo.
- Join (Sincronização): Várias setas convergem para uma única linha, indicando que todas as atividades devem ser concluídas antes de continuar.

6. Activity Final Node (Fim do processo)

● Círculo preto dentro de outro círculo — Indica o fim do fluxo do processo.

- Relação entre os "conceitos da área de negócio" (classes do modelo de domínio) e os fluxos de objetos nos modelos de atividade, permitindo alinhamento entre a estrutura estática (modelo de domínio) e a dinâmica (modelo de processo).

3. Benefícios da modelagem de domínio e processos:



- Facilita o entendimento comum entre equipe técnica e stakeholders.
- Apoia a validação de requisitos e detecção de inconsistências.
- Serve como base para derivação de modelos mais técnicos (como diagramas de sequência ou estrutura de banco de dados).
- Contribui para a reusabilidade e manutenção do sistema ao longo do tempo.

4. Boas práticas:

- Evitar misturar aspectos técnicos de implementação na fase de análise conceitual.
- Validar constantemente os modelos com os especialistas do domínio.
- Utilizar ferramentas adequadas (como UML) para garantir padronização e clareza.

Resumo: Orientação a Objetos e Modelação Estrutural

Orientação aos Objetos no SDLC

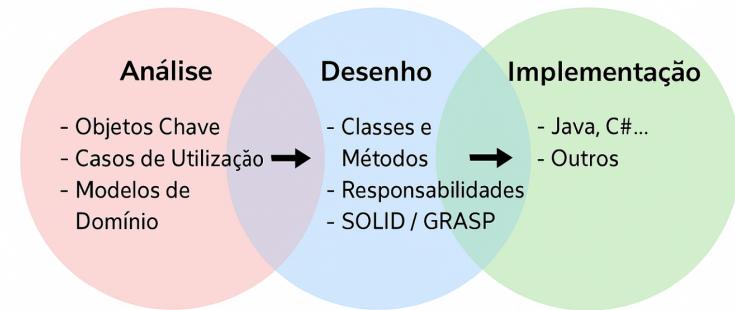
A orientação a objetos (OO) no ciclo de vida do desenvolvimento de sistemas (SDLC) abrange três fases principais:

- Análise orientada a objetos (OOA): identifica os conceitos centrais do domínio do problema. Define atores, entidades e seus relacionamentos com base nos requisitos. Usa frequentemente casos de utilização e modelos de domínio.
- Desenho orientado a objetos (OOD): transforma os conceitos identificados na análise em classes técnicas, métodos, estruturas e padrões. Define responsabilidades e colaborações com base em princípios como GRASP e SOLID.
- Programação orientada a objetos (OOP): a fase de implementação prática dos modelos em linguagens como Java, C#, etc.

Distribuição de responsabilidades e colaboração:

- Objetos colaboram por envio de mensagens (chamada de métodos).
- Cada objeto deve ter uma única responsabilidade bem definida (SRP).
- Padrões como Expert, Creator e Controller ajudam a distribuir responsabilidades corretamente.
- Objetos bem definidos facilitam reutilização, testes e manutenção do sistema.

Objectos no SDLC



Modelação Estrutural

Foca-se na representação estática do sistema, mostrando as classes, seus atributos, métodos e relações entre si.

- **Análise top-down**: abordagem funcional, típica de sistemas legados, que parte de funções principais e decompõe em subtarefas.
- **Orientação a objetos**: baseia-se nos conceitos do domínio para identificar entidades persistentes e suas responsabilidades.

Relação entre diagramas de classes e objetos:

- 1 - **Diagrama de classes**: mostra estrutura estática - nome da classe, atributos, métodos e relações como associações, herança, composição e agregação.
- 2 - **Diagrama de objetos**: mostra instâncias reais dessas classes, úteis para validar cenários e testar comportamento.



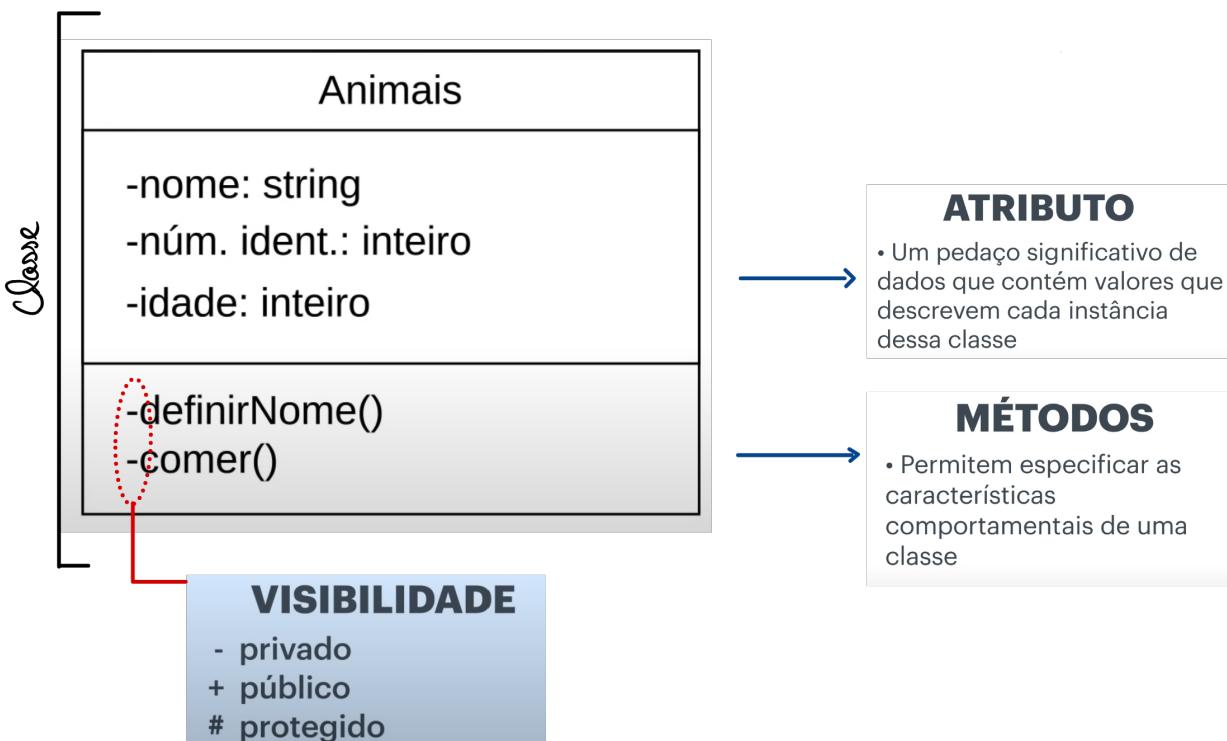
Resumo: Orientação a Objetos e Modelação Estrutural



Validação de modelos:

- **Verificar sintaxe:** conformidade com notação UML.
- **Verificar semântica:** representação fiel da realidade do domínio.
- As associações devem ter **multiplicidade, direcionalidade e nomes representativos claros.**

[Espaço reservado para exemplo de diagrama de classes com associação e composição]

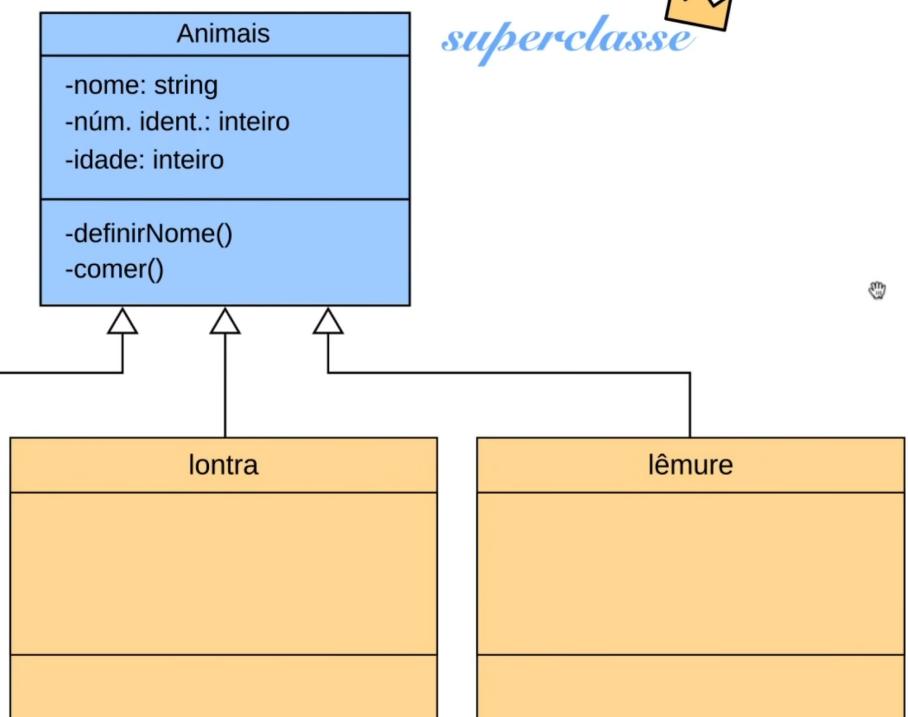


• Se quisermos criar novos animais
eles podem ter herança de uma classe anterior

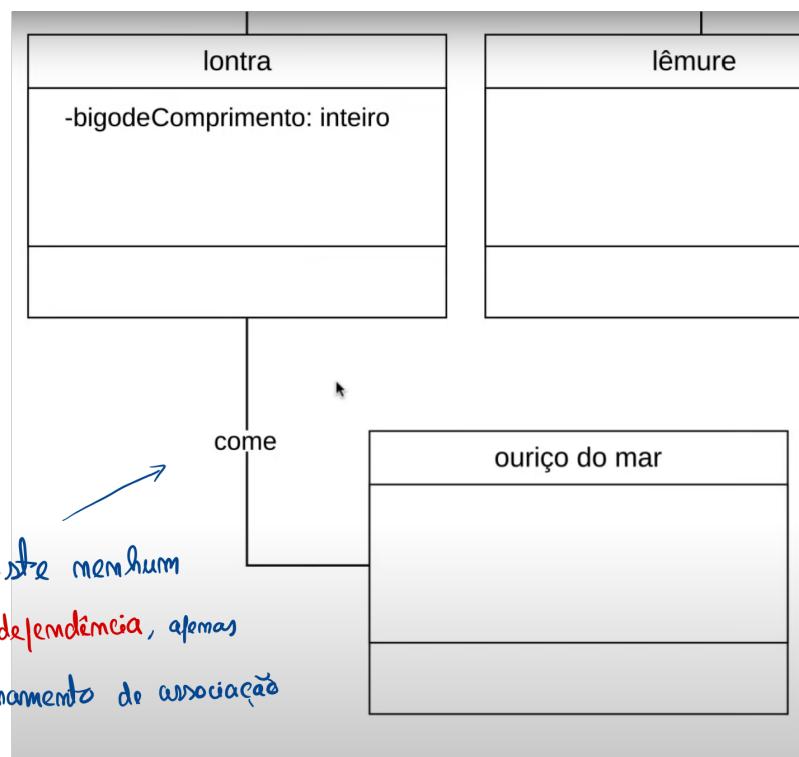
RELACIONAMENTOS

- herança →

subclasse



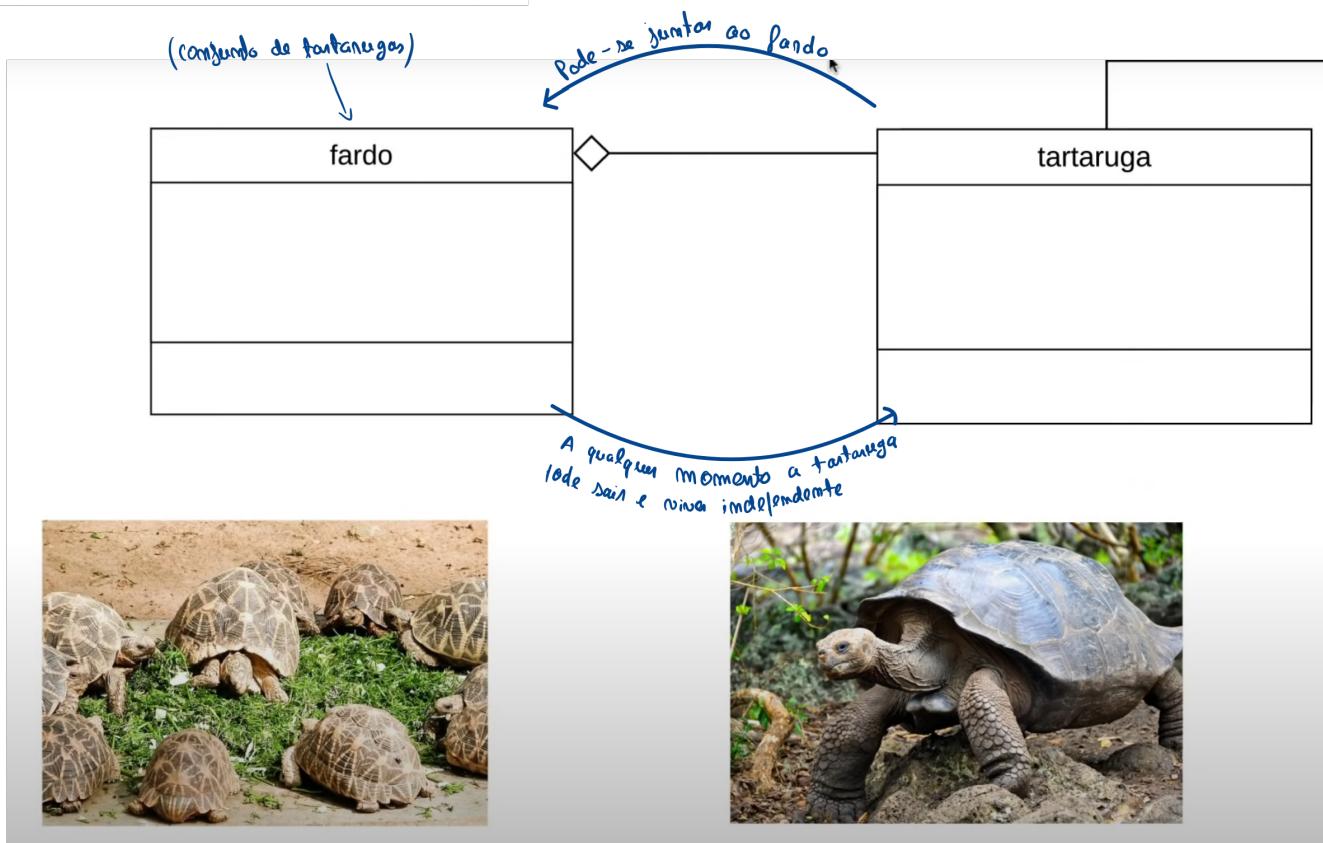
• associação —



• agregação —

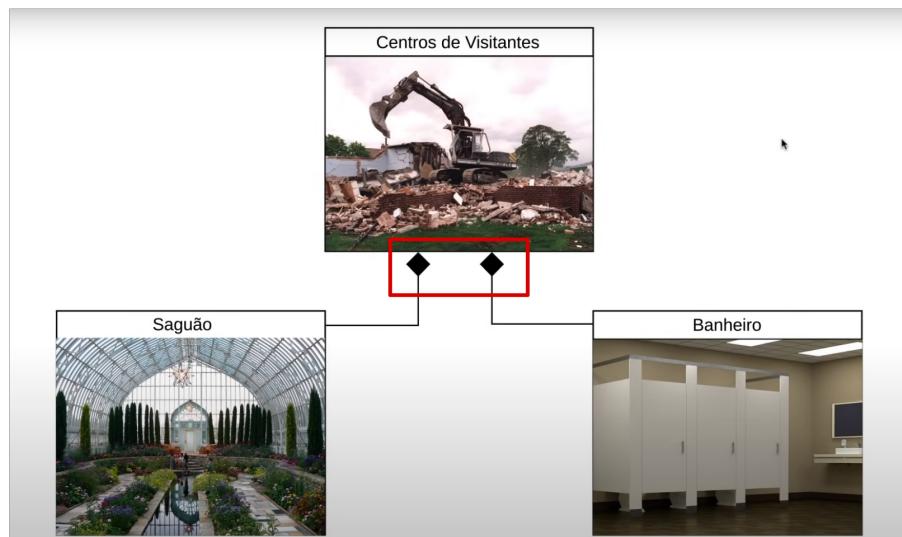
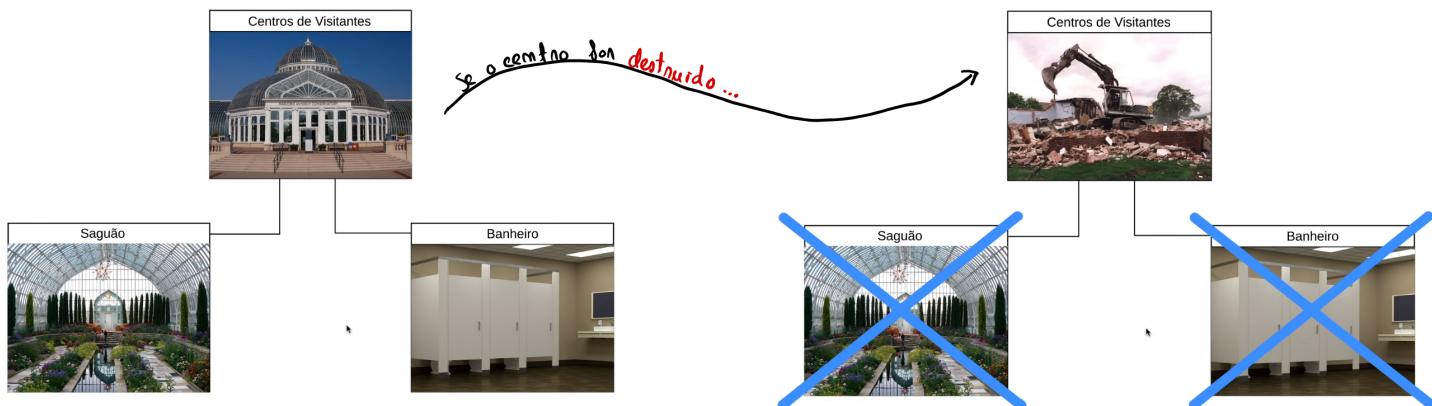


"uma parte pode existir fora de um todo "



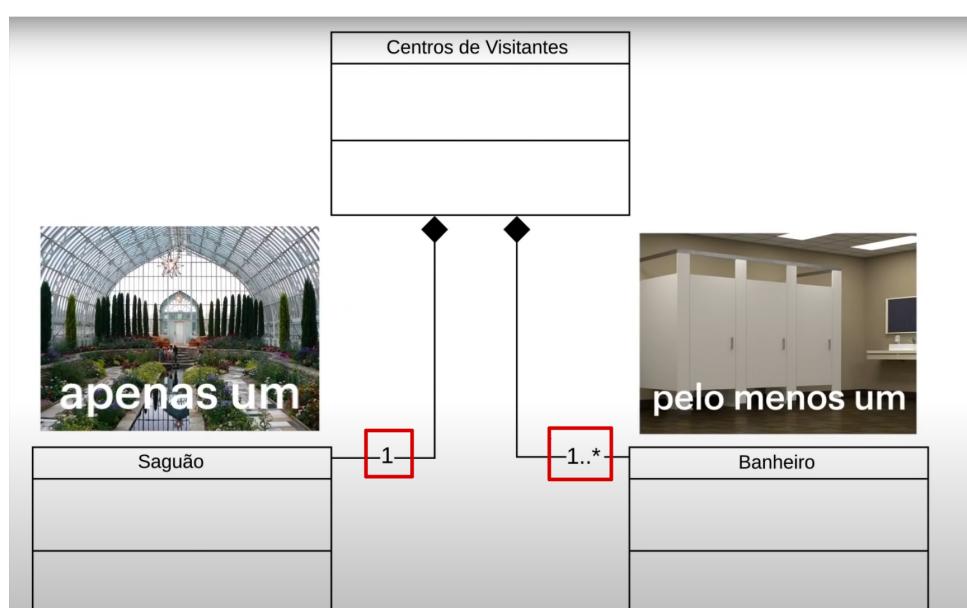
• composição

ocorre quando um objeto secundário não consegue existir sem o objeto primário

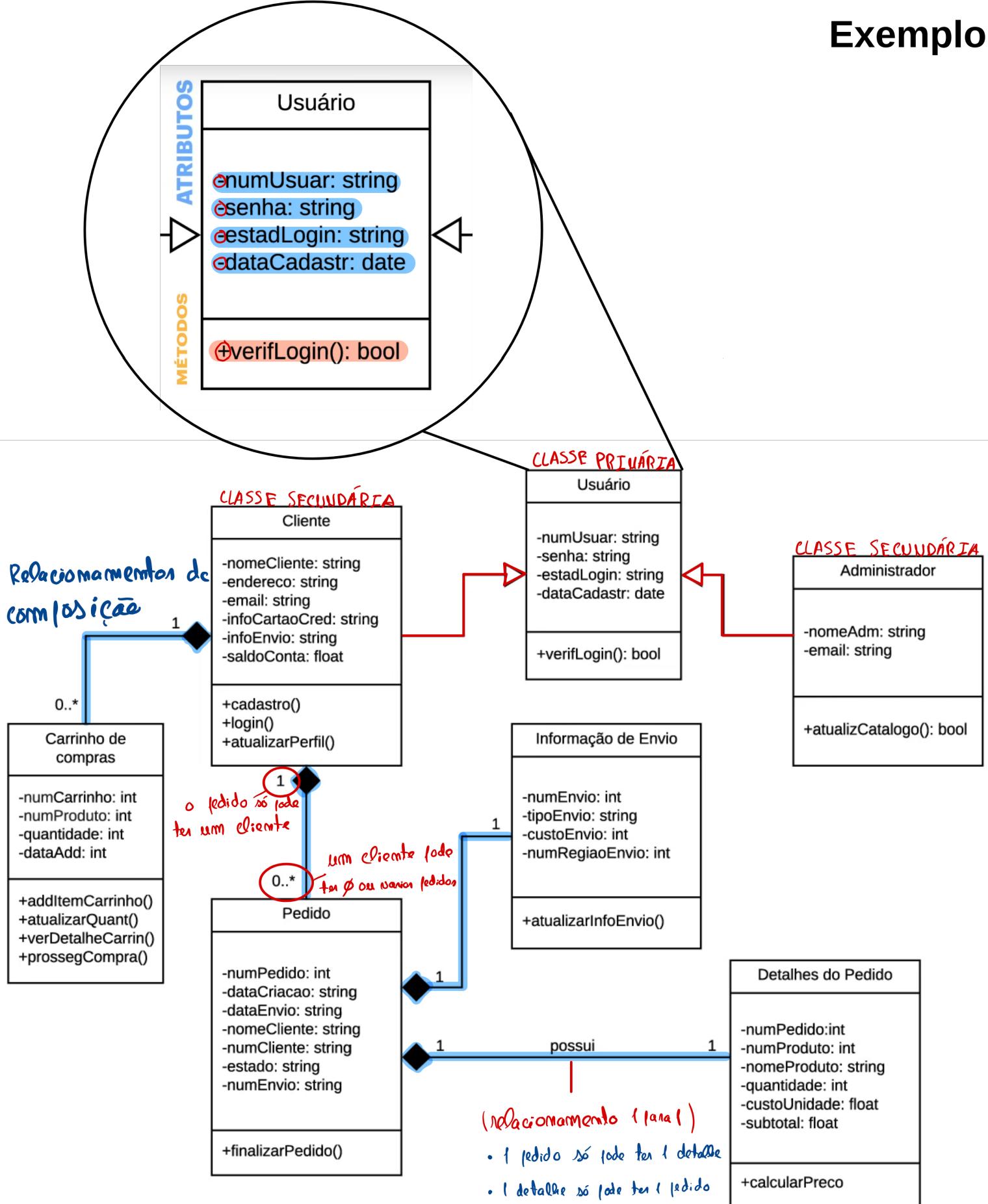


MULTIPLICIDADE

n (montante específico)
 $0..*$ zero a muitos
 $1..*$ um a vários
 $m..n$ intervalo específico



Exemplo



Vistas de arquitetura

- Explicar as atividades associadas ao desenvolvimento de arquitetura de software, no openUP.
- Explicar a relação entre requisitos e a arquitetura (como é que aqueles influenciam esta). Exemplificar requisitos com impacto na arquitetura.
- Explique a prática de “arquitetura evolutiva” proposta no OpenUp.
- Identifique as camadas e partições numa arquitetura de software por camadas.
- Usar diagramas de sequência para descrever a cooperação entre módulos/elementos de uma arquitetura.
- Identifique os três tipos de estruturas principais, constituintes de uma arquitetura (segundo L. Bass *et al*). Relacionar essas categorias com os diagramas de UML mais relevantes para as documentar.
- Discutir razões técnicas e não técnicas que justificam o desenvolvimento de uma (boa) arquitetura para o sistema a construir.

Resumo: Vistas de Arquitetura

Vistas de Arquitetura

As vistas de arquitetura organizam a estrutura do software em diferentes perspectivas, permitindo compreender a organização técnica e lógica do sistema. No OpenUP, a arquitetura é desenvolvida iterativamente, com foco na mitigação de riscos técnicos desde cedo.

- Atividades principais:

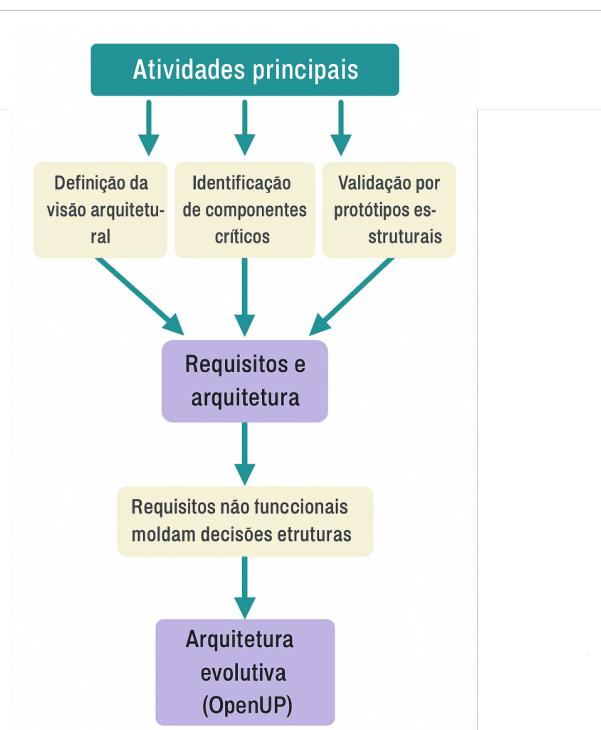
- Definição da visão arquitetural.
- Identificação de componentes críticos.
- Validação por protótipos e provas de conceito.

- Requisitos e arquitetura:

- Requisitos funcionais afetam os módulos e funcionalidades.
- Requisitos não funcionais (performance, segurança, escalabilidade) moldam decisões estruturais.
- Exemplos: requisito de alta disponibilidade -> clusterização; requisito de segurança -> autenticação forte.

- Arquitetura evolutiva (OpenUP):

- Arquitetura não é fixa, evolui à medida que o sistema cresce.
- Suportada por refatorações, testes automatizados e validação contínua.
- Favorece flexibilidade, adaptação a mudanças e alinhamento com negócio.



- Camadas e partições:

- Separação lógica em camadas: Apresentação, Lógica de Negócio, Persistência, etc.
- Partições dividem o sistema por responsabilidades ou domínios.
- Facilita o isolamento de responsabilidades e a manutenção.

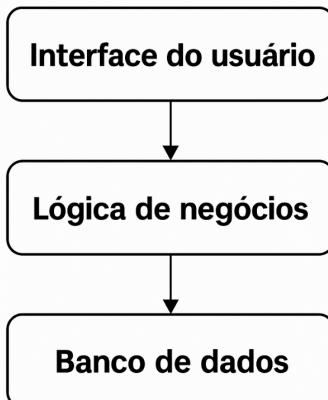
- Diagramas de sequência para arquitetura:

- Permitem representar a interação entre módulos/componentes.

Resumo: Vistas de Arquitetura

- Úteis para entender cooperação e dependências entre elementos da arquitetura.

Exemplo de evolução arquitetural ou camadas



- Três estruturas segundo Bass (2003):

- Estrutura de Módulos: organização do código-fonte. UML: pacotes.
- Estrutura de Componentes e Conectores: visão de tempo de execução. UML: componentes e deployment.
- Estrutura de Alocação: mapeia o software para hardware ou equipas. UML: nós e artefactos.

- Justificações da arquitetura:

- Técnicas: performance, reutilização, escalabilidade, manutenibilidade.
- Não técnicas: conformidade com padrões, decisões políticas, restrições de equipa ou legado.

Estrutura de Módulos

organização do código-fonte. UML: pacotes.



Estrutura de Componentes e Conectores

visão de tempo de execução. UML: componentes e deployment.



Estrutura de Alocação

mapeia o software para hardware ou equipas.
UML: nós e artefatos.

Desenho do software (perspetiva do programador)

- Explicar como os casos de utilização podem ser usados para orientar as atividades de desenho (na perspetiva do livro de Larman).
- Explicar os princípios do baixo acoplamento e alta coesão em OO. Discutir implicações do *coupling* e *cohesion*.
- Comparar, num dado desenho por objetos, a ocorrência de maior/menor *coupling* e *cohesion*.
- Relacionar código por objetos com a sua representação em diagramas de classes da UML.
- Modelar a interação entre unidades de software (objetos) como diagramas de sequência.
- Construir um diagrama de classes e um diagrama de sequência considerando um código Java.
- Explicar as implicações no código da naveabilidade modelada no diagrama de classes.