Diagrama de Classes

Representa **a estrutura estática do sistema**, mostrando **classes, atributos, métodos e relações** entre elas.

1️⃣ O que é?

📌 **É um modelo visual** que mostra **como as classes do sistema se relacionam**.

📌 **Define a estrutura do software**, permitindo compreender **como os objetos interagem**.

📌 **Ajuda no design OO** ao representar conceitos como **composição, associação e agregação**.

2️⃣ Por que é importante?

✅ **Facilita a compreensão do sistema** – Mostra a organização das classes antes da implementação.

✅ **Ajuda na comunicação** entre programadores, analistas e arquitetos.

✅ **Evita erros de implementação** – Serve como um "esqueleto" antes da codificação.

3️⃣ Quando se usa?

📌 Durante a **fase de design OO**, para estruturar as classes antes da implementação.

📌 Para **documentação técnica**, facilitando a manutenção e evolução do software.

4️⃣ Como construir um Diagrama de Classes?

🔷 Passo 1: Identificar as Classes

1️⃣ As classes são derivadas do **Modelo de Domínio** e **Casos de Uso**.

2️⃣ As classes devem representar **conceitos importantes** do sistema.

3️⃣ Nomear as classes no **singular e com inicial maiúscula** (ex.: `Cliente`, `Produto`).

🔷 Passo 2: Definir Atributos e Métodos

📌 **Atributos** → São as características da classe.

📌 **Métodos** → São as ações que a classe pode executar.

```
@startuml
class Produto {
  - id: int
  - nome: String
  - preco: double
  + calcularDesconto(): double
}
@enduml
```

✅ **Atributos em minúsculas** e **métodos começam com verbo no infinitivo**.

✅ **O sinal indica visibilidade**:

- `+` **Public** (acessível por qualquer classe).
- `-` **Private** (acessível apenas dentro da própria classe).

  → Métodos privados não são representados no diagrama de classes. O diagrama de classes serve para comunicação entre desenvolvedores e arquitetos, então foca-se na estrutura pública da classe. Os métodos privados são detalhes internos da implementação, e não impactam outras classes diretamente.

- `#` **Protected** (acessível pela classe e classes dentro do mesmo package).

🔷 Passo 3: Definir Relações entre Classes

Como o Acoplamento é Representado no Diagrama de Classes?

Embora o **diagrama de classes** não seja explicitamente projetado para mostrar o grau de **acoplamento** de forma numérica, ele representa as **relações entre as classes**, o que pode indicar onde o acoplamento existe. Existem diferentes tipos de relacionamentos que indicam acoplamento entre as classes:

📌 As classes interagem entre si de várias formas:

Relação	Símbolo UML	Explicação
Associação	`--`	Uma classe usa a outra. Ex.: `Pedido` tem `Cliente`.
Agregação	`<>--`	Relação "tem um", mas os objetos podem existir separadamente.
Composição	`◆--`	Relação "contém um", onde um objeto não pode existir sem o outro.
Herança	`--`	>`
Implementação	`--`	>` (tracejado)

📌 Segundo os princípios do **GRASP**, **relações bidirecionais devem ser evitadas sempre que possível**, pois **aumentam o acoplamento entre as classes** e tornam o sistema mais difícil de modificar e manter.

Quando Usar Relações Unidirecionais?

✅ Se **uma classe não precisa realmente de conhecer a outra**, evita-se a relação bidirecional.

✅ Se **uma consulta reversa pode ser feita por um serviço ou repositório**, evitamos a referência direta.

✅ Se **uma das classes já tem acesso à outra indiretamente**, não há necessidade de referência mútua.

➡️ `Livro` conhece `Autor`, mas `Autor` não precisa conhecer `Livro`.
