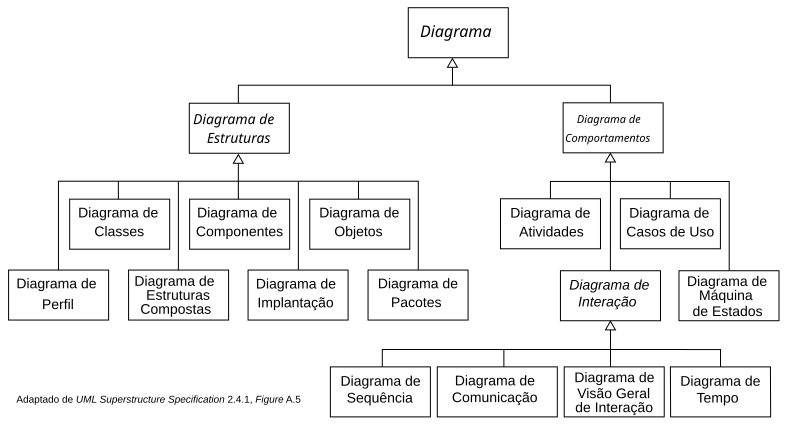
Modelagem UML

(Unified Modeling Language, Linguagem de Modelagem Unificada)

**Resumo**

É uma linguagem-padrão para a elaboração da estrutura de projetos de software. Ela poderá ser empregada para a visualização, a especificação, a construção e a documentação de artefatos que façam uso de sistemas complexos de software. Em outras palavras, na área de Engenharia de Software, a UML é uma linguagem de modelagem que permite representar um sistema de forma padronizada (com intuito de facilitar a compreensão de pré-implementação).

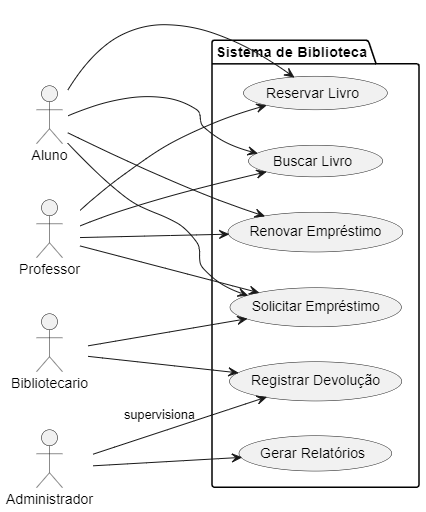


UML é adequada para a modelagem de sistemas, cuja abrangência poderá incluir desde sistemas de informação corporativos a serem distribuídos a aplicações baseadas na Web e até sistemas complexos embutidos de tempo real. É uma linguagem muito expressiva, abrangendo todas as visões necessárias ao desenvolvimento e implantação desses sistemas.

**Para que serve cada diagrama UML**

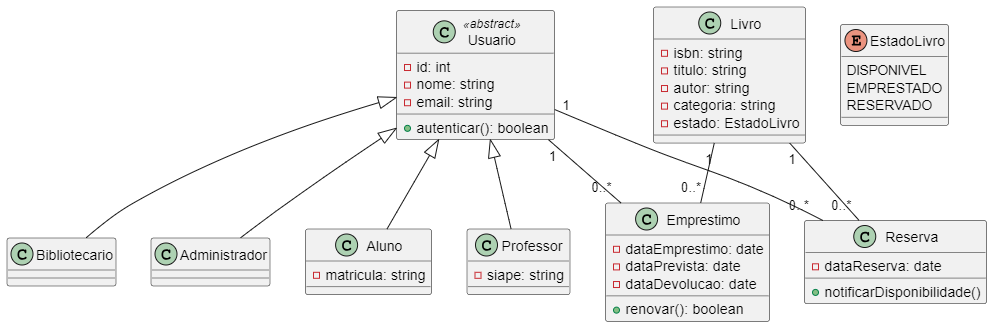
**1. Diagrama de Casos de Uso (Use-Case Diagram)**

* **Propósito**: Capturar *o que* o sistema fará do ponto de vista do usuário, não *como*.
* **Perguntas que responde**
  + Quem interage com o sistema? (atores)
  + Quais objetivos esses atores desejam atingir? (casos de uso)
  + Há dependências obrigatórias ou opcionais entre esses objetivos? (<<include>>, <<extend>>).
* **Quando usar**: logo no início do projeto, durante a elicitação de requisitos; excelente para conversar com stakeholders não técnicos.
* **Valor agregado**
  + Define a fronteira do sistema de forma visual.
  + Permite priorizar funções antes de mergulhar na arquitetura.
* **Exemplo prático**: “Aluno” reserva livro; “Bibliotecário” registra devolução – ambos são facilmente vistos no diagrama, tornando claras as responsabilidades.



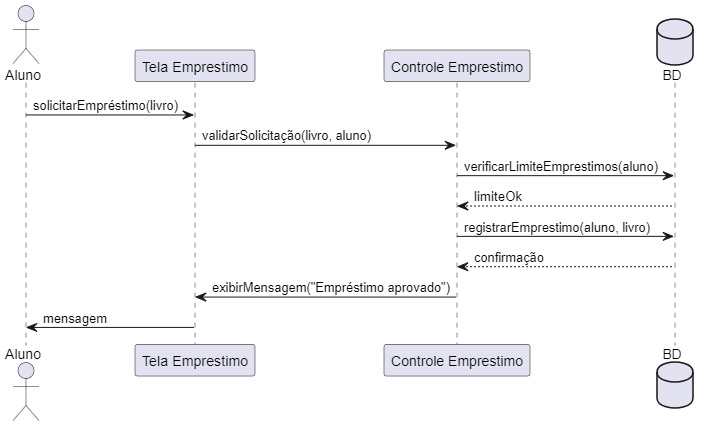
**2. Diagrama de Classes (Class Diagram)**

* **Propósito**: Descrever a estrutura estática do domínio: classes, atributos, métodos e relacionamentos (associação, herança, composição, agregação, dependência).
* **Perguntas que responde**
  + Quais entidades compõem o domínio?
  + Como elas se relacionam e quais regras de cardinalidade existem?
  + Que operações cada entidade oferece?
* **Quando usar**: durante análise de domínio e desenho da solução; serve de base para gerar código ou schema de banco de dados.
* **Valor agregado**
  + Torna explícitas as regras de negócio (ex.: “Empréstimo contém 1..\* Livros”).
  + Facilita detecção de ciclos e acoplamento excessivo.
* **Exemplo prático**: Usuario abstrato com subclasses Aluno, Professor; enum EstadoLivro prepara terreno para diagrama de estados.



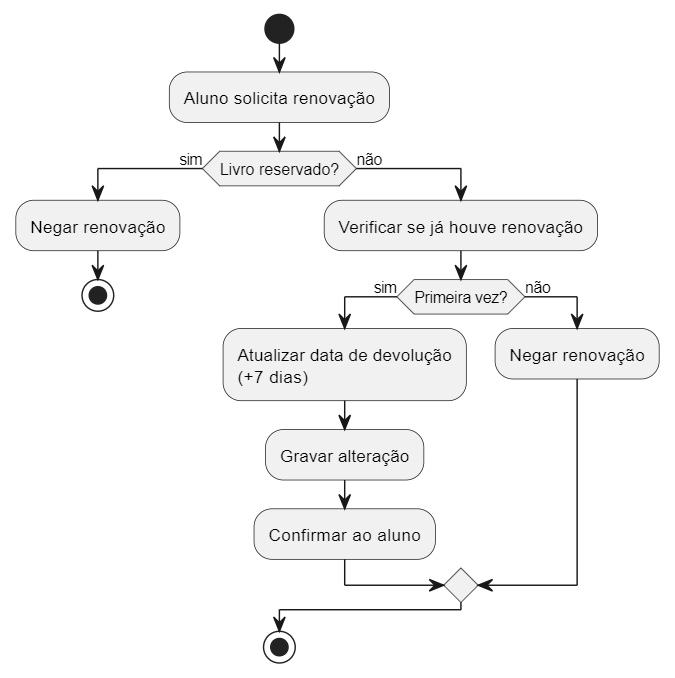
**3. Diagrama de Sequência (Sequence Diagram)**

* **Propósito**: Modelar a interação temporal entre objetos/partes do sistema para realizar um cenário específico.
* **Perguntas que responde**
  + Qual é a ordem exata das mensagens?
  + Que objeto inicia a operação? Onde ocorrem validações?
  + Há paralelismo ou chamadas assíncronas?
* **Quando usar**: ao refinar casos de uso críticos, definir APIs, ou depurar fluxos complexos (login, pagamento, transação bancária).
* **Valor agregado**
  + Revela dependências ocultas e gargalos de performance.
  + Alinha desenvolvedores sobre quem chama quem e em que ordem.
* **Exemplo prático**: sequência “Aluno solicita empréstimo” → sistema checa limite → bibliotecário confirma → banco registra.



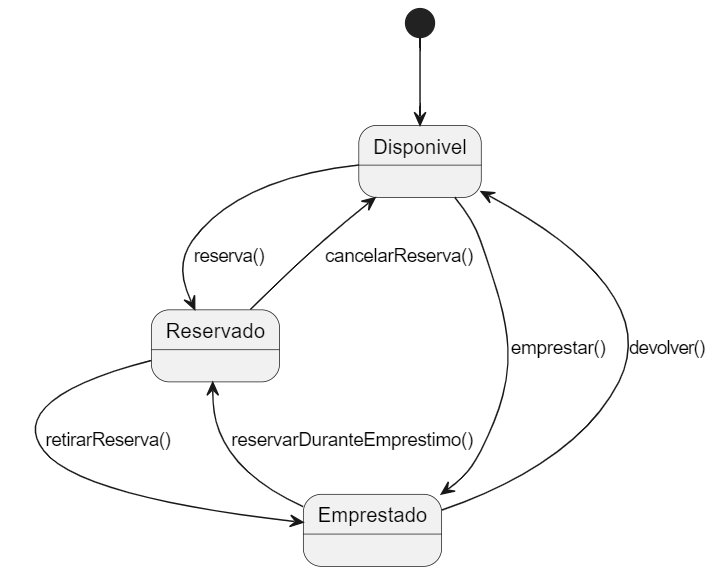
**4. Diagrama de Atividades (Activity Diagram)**

* **Propósito**: Representar fluxos de trabalho, algoritmos ou processos de alto nível, incluindo decisões, paralelismo e loops.
* **Perguntas que responde**
  + Quais passos compõem um processo?
  + Onde ocorrem bifurcações (decision nodes) e junções (merge/join)?
  + Quais atividades podem executar em paralelo?
* **Quando usar**: para documentar regras de negócio, processos de negócio BPM, ou lógica de métodos complexos.
* **Valor agregado**
  + Comunica claramente caminhos alternativos e exceções.
  + Serve de ponte entre análise de requisitos e modelagem de processo (BPMN).
* **Exemplo prático**: processo de renovação de empréstimo – vértice inicial, verificação de reserva, ramificação “permitido / negado”, vértice final.



**5. Diagrama de Estados (State-Machine Diagram)**

* **Propósito**: Mostrar os estados possíveis de um objeto (ou sistema) e os eventos que provocam transições entre esses estados.
* **Perguntas que responde**
  + Quais são os estados válidos?
  + Que eventos disparam mudanças?
  + Existem transições condicionais ou ações de entrada/saída?
* **Quando usar**: em entidades com ciclo de vida complexo (documentos, dispositivos IoT, ordens de compra) ou para validar lógica de controle.
* **Valor agregado**
  + Detecta estados inalcançáveis ou transições faltantes antes da implementação.
  + Auxilia na escrita de testes baseados em eventos.
* **Exemplo prático**: Livro → estados *Disponível*, *Reservado*, *Emprestado*; eventos reservar(), devolver(), emprestar().



**Síntese**

* **Casos de Uso** definem escopo funcional e atores.
* **Classes** formalizam a estrutura estática do domínio.
* **Sequência** detalha a dinâmica de um cenário específico.
* **Atividades** mapeiam processos com fluxo de controle.
* **Estados** descrevem a evolução temporal de um objeto.

Juntos, esses cinco diagramas oferecem visão complementar que vai do requisito ao funcionamento interno, garantindo entendimento compartilhado e redução de risco durante o desenvolvimento.