Imagine que você está desenvolvendo um sistema de gerenciamento de uma biblioteca. Neste sistema, é necessário modelar livros, autores, usuários, empréstimos e devoluções. Cada livro possui informações como título, autor e gênero. Os autores têm nome, nacionalidade e obras publicadas. Os usuários da biblioteca têm nome, idade e histórico de empréstimos. Os empréstimos registram a data de retirada e devolução, além do livro e usuário envolvidos.

Com base no cenário descrito, analise qual paradigma de programação seria mais indicado para implementar o sistema de gerenciamento da biblioteca:

a) programação estruturada
b) programação imperativa
c) programação orientada a objetos ou
d) programação funcional

Justifique sua escolha considerando a estrutura e as interações entre os objetos do sistema.
Para facilitar, considere pontos como:
Estrutura e interação das entidades do sistema (Livros, autores, usuários e demais)
Reuso
Qual paradigma facilitaria a evolução do sistema, no futuro
Eficiência, dado o cenário apresentado

Dica: A combinação dos paradigmas de programação pode ser viável em certos contextos, mas geralmente não é recomendada devido a diferenças fundamentais entre os paradigmas.

A mais indicada é a Programação orientada a objetos (POO).

Por quê:

Estrutura e interação das entidades: O domínio tem objetos “naturais” — Livro, Autor, Usuario, Emprestimo/Devolucao. Em POO, cada um vira uma classe com estado (atributos) e comportamentos (métodos). As relações são claras:

Livro (título, gênero) ↔ composição com Autor

Usuário (nome, idade, histórico) ↔ associa Empréstimo

Empréstimo encapsula regras: retirar(), devolver(), cálculo de multa/atraso. Isso espelha o modelo conceitual da biblioteca e reduz “acoplamento solto” entre estruturas e regras.

Reuso: Padrões POO (herança/composição) e interfaces favorecem reuso:

Publicação → Livro, Revista no futuro.

Estratégias de multa: RegraMulta (polimórfica) para adulto/infantil/academic.

Serviços reutilizáveis: CatalogoService, EmprestimoService, NotificacaoService.

Evolução do sistema: A POO isola mudanças:

Adicionar reservas, filas de espera, vários exemplares por título → novas classes (Exemplar, Reserva) e pequenas mudanças nos serviços.

Trocar persistência (arquivo → banco → API) mantendo interfaces (RepositorioLivro, RepositorioUsuario).

Eficiência (no cenário): Diferenças de desempenho entre paradigmas são irrelevantes perto de E/S de banco e rede. O ganho real é eficiência de desenvolvimento/manutenção: POO reduz bugs ao encapsular invariantes (ex.: não permitir devolver() um empréstimo já devolvido).

Por que não os outros como principal:

Estruturada/Imperativa: funcionam para scripts pequenos, mas escalam mal. A lógica de empréstimos, validações, políticas e estados tende a virar um "procedural bowl of spaghetti" com dados e regras espalhados.

Funcional: brilha em transformações puras (consultas, relatórios), mas o domínio aqui é orientado a estado e efeitos colaterais (criar empréstimo, atualizar estoque, registrar devolução). Dá para usar funções puras dentro do desenho OO (p.ex., validadores, cálculos de multa), mas como paradigma principal complica o manejo de estado.

Você deve ter percebido que nossas aulas têm vários diagramas. Eles ajudam na correta compreensão dos conceitos e, quando projetamos sistemas reais, esses recursos nos ajudam a entender alguns detalhes de implementação e viabilidade do que estamos fazendo. Neste exercício, você vai criar seu primeiro diagrama!

Antes de começar a desenhar, tenha em mãos as seguintes respostas:
Quais serão os elementos (classes) do diagrama?
Com os conhecimentos que você já tem em programação, consegue pensar no tipo de cada propriedade? Por exemplo, no caso do livro, qual seria o tipo da propriedade "Título"? Seria uma string, um número, ou outro tipo?
Qual seria o relacionamento entre os elementos? Pense em relacionamentos 1 para 1 e 1 para muitos

Regras:
- Seu diagrama deve estar de acordo com o que você definiu no exercício anterior, em Elementos de diagrama e Relações entre elementos

Classes do diagrama e tipos de propriedades:

Pessoa:
    Nome: string.
    Lista de livros: Livros[] (array de objetos do tipo livro).

Emprestimo:
    data de retirada: Date.
    data de devolução: Date.
    livro: Livro.
    usuário: Usuario.

Usuario:
    idade: int.
    histórico de empréstimos: Emprestimo[] (array de objetos do tipo Emprestimo).

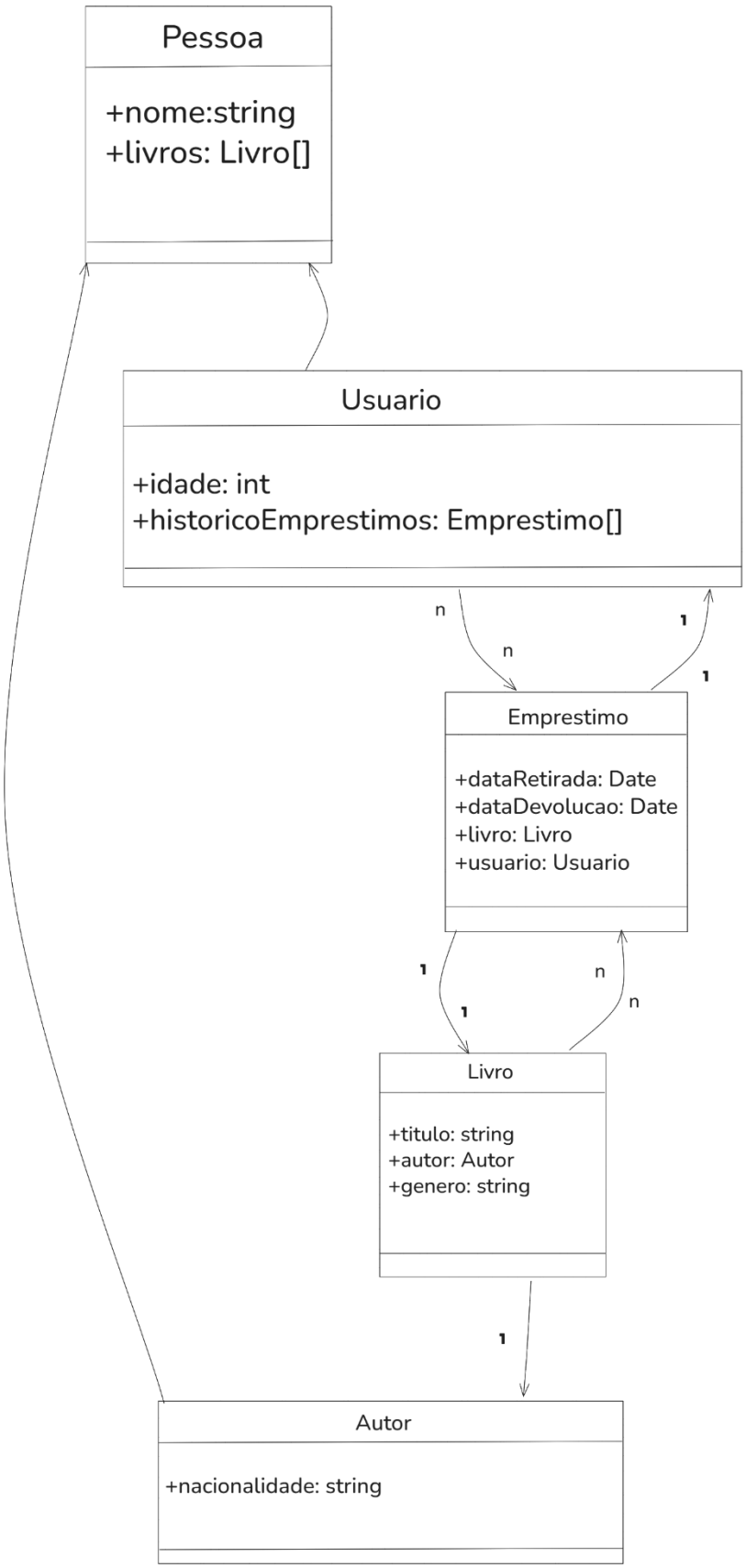
Livro:
    tìtulo: string
    autor: Autor.
    gênero: string

Autor:
    nacionalidade: string.

Relações

1 autor é 1 pessoa.
1 usuário é uma pessoa.
1 Livro possui 1 autor principal – levando em consideração que cada livro tenha apenas 1 autor.
Cada Autor pode escrever N Livros.
Cada Livro pode ser emprestado N vezes, mas não pode estar em amis de 1 emprestimo por vez.
Cada Usuário pode realizar N Empréstimos por vez.
Cada empréstimo envolve pelo menos 1 livro e apenas 1 Usuário.

DESENHO DO DIAGRAMA


Pessoa
+nome:string
+livros: Livro[]
Usuario
+idade: int
+historicoEmprestimos: Emprestimo[]
n
n
1
1
Emprestimo
+dataRetirada: Date
+dataDevolucao: Date
+livro: Livro
+usuario: Usuario
1
1
n
n
Livro
+titulo: string
+autor: Autor
+genero: string
1
Autor
+nacionalidade: string

4. Agora, vamos acrescentar mais alguns detalhes ao seu diagrama! Já desenhamos o diagrama de classes, contendo propriedades e relacionamentos entre os objetos.

Agora, você vai receber os fluxos do nosso sistema de gerenciamento! Precisamos dessas informações para completar nosso exercício, que vai ser adicionar os métodos às nossas classes, de acordo com o que o sistema deve fazer. Para facilitar, vamos deixar os fluxos mais simples para o nosso projeto de aula.

Fluxo de Empréstimo de Livros

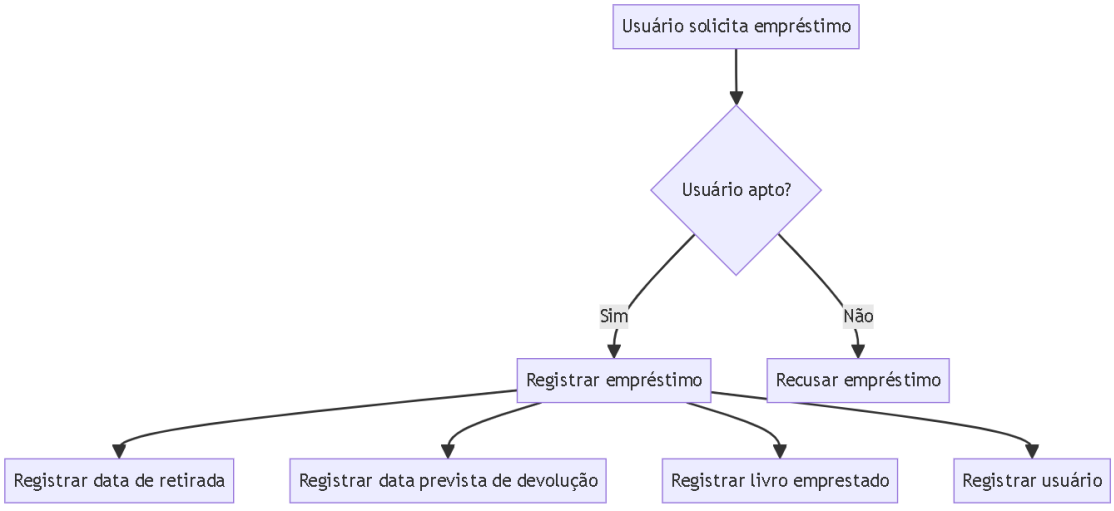
1. O usuário solicita o empréstimo de um ou mais livros.
2. O sistema verifica se o usuário está apto a realizar o empréstimo:

a. Consulta o nome do usuário para identificá-lo
b. Verificar o histórico de empréstimos do usuário para ver se ele não possui livros vencidos.
c. Verificar a idade do usuário, caso o livro solicitado seja de um gênero impróprio para menores de 18 anos.

3. Se o usuário estiver apto, o sistema:

a. Registra a data de retirada do livro.
b. Registra a data prevista de devolução.
c. Associar o livro emprestado e o usuário que realizou o empréstimo.

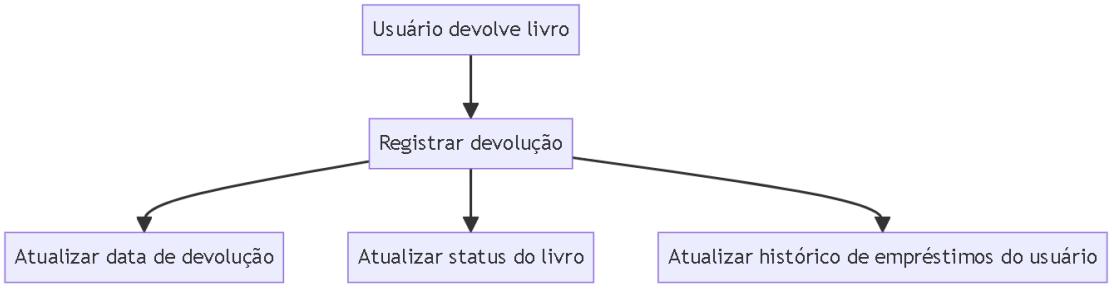
4. Caso o usuário não esteja apto, o sistema recusa o empréstimo



Fluxo de Devolução de Livros

1. O usuário devolve um ou mais livros.
2. O sistema registra a devolução, atualizando:

a. A data de devolução do livro.
b. O status do livro como disponível na biblioteca.
c. O histórico de empréstimos do usuário.

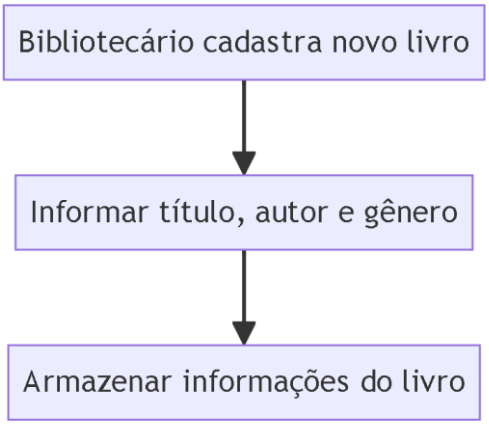


Fluxo de Cadastro de Livros

1. O bibliotecário cadastra um novo livro no sistema, informando:

a. O título do livro.
b. O autor do livro.
c. O gênero do livro.

2. O sistema armazena as informações do novo livro.

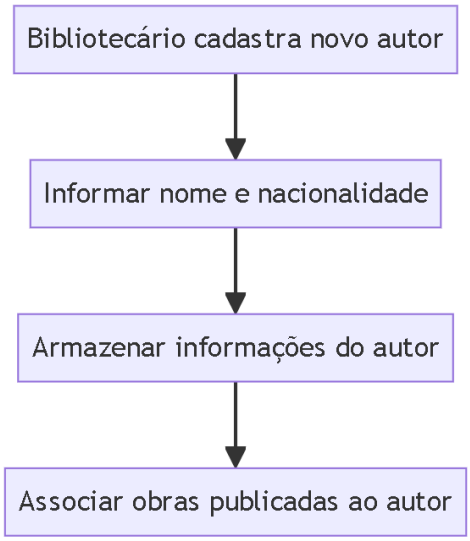


Fluxo de Cadastro de Autores

1. O bibliotecário cadastra um novo autor no sistema, informando:

a. O nome do autor.
b. A nacionalidade do autor.

2. O sistema armazena as informações do novo autor e associa as obras publicadas a ele.



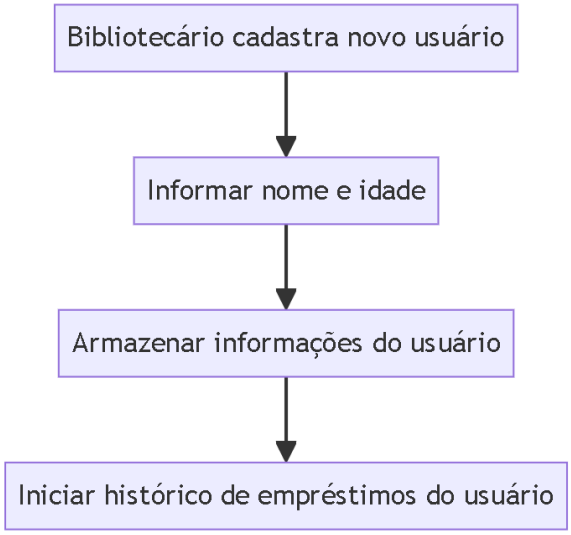
Fluxo de Cadastro de Usuários

1. O bibliotecário cadastra um novo usuário no sistema, informando:

a. O nome do usuário.

b. A idade do usuário.

2. O sistema armazena as informações do novo usuário e inicia seu histórico de empréstimos.



Com base em todos os fluxos e regras de negócio, quais seriam os métodos que precisamos acrescentar em nossas classes?

Métodos por Classe

Classe Livro:

Atributos já definidos: título (string), gênero (string), autor (Autor).
Novos métodos:
cadastrarLivro(titulo, autor, genero) → registra um novo livro no sistema.

atualizarStatus(disponivel: boolean) → marca o livro como disponível ou emprestado.
getInfo() → retorna detalhes do livro.

Classe Autor:

Atributos já definidos: nome (string), nacionalidade (string), obras publicadas (Lista<Livro>).
Novos métodos:
cadastrarAutor(nome, nacionalidade) → cria um autor.
adicionarObra(livro: Livro) → associa um livro ao autor.
listarObras() → retorna todos os livros do autor.

Classe Usuario:

Atributos já definidos: nome (string), idade (int), histórico de empréstimos (Lista<Emprestimo>).
Novos métodos:
cadastrarUsuario(nome, idade) → adiciona um usuário ao sistema.
verificarApto() → verifica se pode realizar novo empréstimo (sem atrasos, idade permitida).
listarHistorico() → mostra todos os empréstimos feitos pelo usuário.

Classe Emprestimo:
Atributos já definidos: dataRetirada (Date), dataDevolucao (Date), livro (Livro), usuario (Usuario).
Novos métodos:
realizarEmprestimo(usuario, livro) → cria um registro de empréstimo, se o usuário estiver apto.
registrarDevolucao(dataDevolucao) → atualiza status do livro e do histórico do usuário.
verificarAtraso() → retorna se o livro foi devolvido fora do prazo.

Classe Devolucao:

registrarDevolucao(usuario, livro, data) → efetua a devolução.
atualizarHistorico() → grava no histórico do usuário.

Relações com os Métodos

O usuário pede um empréstimo → método de Usuario chama Emprestimo.realizarEmprestimo().
O sistema verifica aptidão → Usuario.verificarApto().
O livro tem seu status atualizado → Livro.atualizarStatus().
Na devolução → Emprestimo.registrarDevolucao() atualiza o livro como disponível e grava no histórico.