# Projeto “Ludoteca .NET” – descrição oficial

## 1. Objetivo geral

Construir, em **C# .NET 9**, um aplicativo **de console** que controle o **empréstimo de jogos de tabuleiro** de um clube universitário.

O programa deve:

- **Interatividade pelo terminal**
  - Exibir um menu textual contínuo (`while` principal).
  - Aceitar a opção digitada pelo usuário e executar o comando correspondente.
  - Exemplo mínimo (pode ser expandido):

```
=== LUDOTECA .NET ===
1  Cadastrar jogo
2  Cadastrar membro
3  Listar jogos
4  Emprestar jogo
5  Devolver jogo
6  Gerar relatório
0  Sair
Opção: _
```

- **Funções obrigatórias**
  1. Cadastro de jogos, membros, empréstimos e devoluções.
  2. Bloqueio de retirada se o jogo já estiver emprestado.
  3. Cálculo e pagamento de multa em caso de atraso (Pix ou dinheiro).
  4. Persistência: leitura e gravação em `biblioteca.json` (pasta *data*).
  5. Relatório em `relatorio.txt`.
  6. Log de erros em `debug.log` e assertivas de consistência.

## 2. Organização em grupo e entrega

- Até **4 integrantes** por equipe (nomes COMPLETOS e também MATRÍCULAS no `README.md`).
- Repositório **PÚBLICO no GitHub** contendo: código-fonte, diagramas, vídeos e artefatos.
- Dois branches obrigatórios (`av1`, `av2`) e duas tags (`v1.0`, `v2.0`).
- Para cada entrega, uma na AV1 e outra na AV2, grave um **vídeo de até 10 min** (link no README) demonstrando e explicando o que foi feito.

- Grupos formados na AV1 deve ser mantidos até o final da entrega do trabalho da AV1. Se houver a necessidade extrema de troca de membros entre grupos, poderá acontecer logo após a entrega do trabalho de AV1, mas então deve ser mantido até o final da entrega do trabalho para AV2.

# 3. Entregas detalhadas

## 3.1 AV1 – Fundamentos de POO + UML (capítulos 1-5)

| Item | O que deve existir | Observações sobre código-fonte |
|---|---|---|
| 1 | **Diagrama UML** (PNG) das classes `Jogo`, `Membro`, `Emprestimo`, `BibliotecaJogos`. | Indicar no README o arquivo da figura. |
| 2 | Implementação das quatro classes com **encapsulamento** e **validações**. | No README, liste cada classe e as linhas onde ficam os construtores e propriedades validadas. |
| 3 | Métodos `Salvar()` e `Carregar()` usando **System.Text.Json**. | Comente `// [AV1-3]` nas linhas de serialização. |
| 4 | **Menu console** que executa as funções mínimas. | Marque no arquivo do menu as linhas de cada opção (ex.: `// [AV1-4-Listar]`). |
| 5 | **Tratamento de exceções** coerente (`InvalidOperationException`, `ArgumentException`). | Adicione comentário `// [AV1-5]` ao bloco `try` / `catch`. |
| 6 | **Evidências** – quatro screenshots descritas na apostila. | Coloque na pasta `/evidencias/av1`. |
| 7 | **Vídeo** (≤ 10 min) mostrando fluxo completo. | Link no README. |

### 3.1.2 AV1 — Critérios de avaliação

| Nº | Critério avaliável | Pontos | Como o avaliador comprova |
|---|---|---|---|
| 1 | Projeto compila e executa com `dotnet build && dotnet run` | **1,0** | Build e execução sem erro; se falhar, itens 2-7 recebem nota 0. |

| Nº | Critério avaliável | Pontos | Como o avaliador comprova |
|---|---|---|---|
| 2 | **Menu** em laço contínuo + validação de opção inválida | **1,0** | Avaliador digita opção inexistente → mensagem clara; laço não encerra. |
| 3 | Diagrama UML (PNG) corresponde às **quatro** classes implementadas | **1,0** | Inspeção visual do diagrama e dos arquivos `.cs`. |
| 4 | Encapsulamento e validações em construtor (`private set`, exceções coerentes) | **1,0** | Revisão das linhas marcadas `// [AV1-2]`; nenhum campo sensível com `public set`. |
| 5 | Persistência JSON com **System.Text.Json** | **1,0** | Menu Salvar → Sair → Abrir → Listar recupera dados; linhas `// [AV1-3]` mostram serialização. |
| 6 | Tratamento de exceções funcional (`InvalidOperationException`, `ArgumentException`) | **1,0** | Tentativa de duplo empréstimo gera mensagem e programa segue. |
| 7 | Cinco funções mínimas operam: cadastrar jogo, cadastrar membro, listar jogos, emprestar, devolver | **2,0** | Avaliador executa todas; saídas corretas. |
| 8 | Qualidade de código: nomes auto-explicativos, métodos ≤ 30 linhas, **sem** `var` | **0,5** | Inspeção rápida; uso de tipos explícitos. |
| 9 | Evidências: 4 screenshots exigidas em `/evidencias/av1` | **0,5** | Verificação de arquivos. |
| 10 | Vídeo (≤ 10 min) com todos do grupo demonstrando o sistema | **1,0** | Link no `README`; duração conferida. |

## 3.2 AV2 – Padrões, herança, polimorfismo, depuração (capítulos 6-12)

| Item | O que deve existir | Como identificar no código |
|---|---|---|
| 1 | **Refatoração**: `RelatorioService` assume impressão e arquivo de resumo. | Comente `// [AV2-1]` na classe nova. |
| 2 | **Herança**: `Jogo` (base), `JogoExpansao`, `JogoPremium`; métodos virtuais sobrescritos. | Marque cada `override` com `// [AV2-2]`. |
| 3 | **Interface** `IPagamentoMulta` + classes `PagamentoPix`, `PagamentoDinheiro`; uso sem `if` de tipo. | Comente `// [AV2-3]` em cada implementação; mostrar no README onde o polimorfismo é invocado. |
| 4 | **Relatório** em `relatorio.txt` com totais e multas. | Linha geradora marcada `// [AV2-4]`. |
| 5 | **Assertivas e log** (`Debug.Assert`, gravação em `debug.log`). | Comentários `// [AV2-5-Assert]` e `// [AV2-5-Log]` nas linhas correspondentes. |
| 6 | **Vídeo** (≤ 10 min) demonstrando refatorações, herança, polimorfismo, relatório e sessão de depuração. | Link no README. |

### 3.2.1 AV2 — Critérios de avaliação

| Nº | Critério avaliável | Pontos | Como o avaliador comprova |
|---|---|---|---|
| 1 | Projeto compila e inicia no branch `av2`; carrega dados JSON pré-existentes | **0,5** | Build ok; lista de jogos aparece. |
| 2 | Todas as funções da AV1 ainda funcionam (regressão zero) | **1,0** | Roteiro AV1 repetido sem falhas. |
| 3 | `RelatorioService` criado e únicas operações de relatório estão nele (`// [AV2-1]`) | **1,0** | Busca textual confirma ausência de relatório em outras classes; arquivo `relatorio.txt` gerado. |

| Nº | Critério avaliável | Pontos | Como o avaliador comprova |
|---|---|---|---|
| 4 | Herança + polimorfismo: `Jogo`, `JogoExpansao`, `JogoPremium` com métodos `virtual/override` (`// [AV2-2]`) | **2,0** | Avaliador cria `List<Jogo>` com instâncias variadas e vê método polimórfico agir. |
| 5 | Interface `IPagamentoMulta` usada sem `if`/`switch` de tipo (`// [AV2-3]`) | **1,0** | Grep no código; instâncias manipuladas só pela interface. |
| 6 | Relatório `relatorio.txt` contém **5** métricas: total de jogos, membros, empréstimos ativos, devoluções em atraso, multas cobradas | **1,0** | Abrir arquivo e conferir valores com estado real do programa. |
| 7 | Assertiva impede quantidade negativa e `debug.log` grava pelo menos duas exceções (`// [AV2-5]`) | **1,0** | Forçar falha → `Debug.Assert` dispara em build *Debug*; log contém data e mensagem. |
| 8 | Qualidade de código: SRP respeitado, nomes claros, métodos ≤ 30 linhas, **sem** `var` | **0,5** | Revisão de arquivos `.cs`. |
| 9 | Cálculo de multa e prazo correto para cada subtipo de jogo | **1,0** | Avaliador empresta e devolve jogos básicos, premium e expansão em datas simuladas; valores batem regras nos métodos sobrescritos. |
| 10 | Vídeo (≤ 10 min) mostra refatorações, polimorfismo, relatório e sessão de depuração | **1,0** | Link no `README`; tempo conferido; demonstração de breakpoint ativo. |

# 4. Convenções de marcação no código

Para facilitar a conferência automática, adote comentários-chave, por exemplo:

```
// [AV1-3]  --- início da serialização
public void Salvar() { ... }
// [AV1-3]  --- fim
```

Descreva no `README.md` onde cada comentário aparece (arquivo + linha).

## 5. Boas práticas de implementação

- Use tipos explícitos (`List<Jogo>`), evite `var`.
- Não exponha listas mutáveis: `public IReadOnlyCollection<Jogos> Jogos => _jogos;`
- Gere JSON indentado (`WriteIndented = true`) para leitura humana.
- Mantenha commits pequenos e com mensagem significativa em português (“Adicionar IPagamentoMulta”).
- Dúvidas? Abra *Issue* no repositório; respostas ficam disponíveis a todos do grupo. Assim os membros podem se ajudar.