Trabalho de Programação Orientada a Objetos em Java

Prazo: 01/12/2025

1. Contexto geral

Este trabalho tem como objetivo desenvolver, de forma **individual**, um **MVP (Minimum Viable Product)** ou **protótipo funcional** em Java, aplicando os principais conceitos de **Programação Orientada a Objetos (POO)**.

O tema do sistema é livre (ex.: biblioteca, controle de estoque, agenda de contatos, sistema acadêmico, locadora, vendas simples, etc.), desde que atenda aos requisitos mínimos abaixo.

2. Escopo mínimo do sistema (funcional)

Para ser compatível com o prazo, o sistema deve ter, no mínimo:

1. **Duas entidades principais** (classes de domínio) relacionadas entre si.
   - Exemplos:
     - `Cliente` e `Pedido`
     - `Aluno` e `Disciplina`
     - `Livro` e `Autor`
2. **CRUD básico para pelo menos 2 entidades**:
   - Criar (cadastrar)
   - Listar (consultar)
   - Atualizar (editar)
   - Excluir (remover)
3. **Pelo menos 1 fluxo de uso completo**, coerente do início ao fim. Exemplos:
   - Cadastrar cliente → cadastrar pedido para esse cliente → listar pedidos do cliente.
   - Cadastrar aluno → matricular em disciplina → listar disciplinas de um aluno.
4. **Interface** (pode ser simples):
   - **Obrigatório**: Interface de **linha de comando (console)** com menus claros

- **Opcional (para se quiser ir além)**: interface gráfica simples (JavaFX/Swing) ou API REST e/ou Web.

3. Requisitos de Orientação a Objetos

O projeto deve demonstrar, obrigatoriamente:

1. **Encapsulamento**
   - Atributos privados ou protegidos.
   - Métodos públicos para acesso/alteração quando necessário (getters/setters ou métodos de negócio).
2. **Composição / Agregação**
   - Uma classe contendo outras como atributos.
   - Ex.: `Pedido` contendo uma lista de `ItemPedido`; `Turma` contendo uma lista de `Aluno`.
3. **Herança**
   - Pelo menos uma hierarquia simples, por exemplo:
     - `Pessoa` → `Cliente`, `Funcionario`
     - `Produto` → `ProdutoFisico`, `ProdutoDigital`
4. **Polimorfismo**
   - Uso de polimorfismo em pelo menos um ponto claro, por exemplo:
     - Lista de `Pagamento` contendo `PagamentoCartao` e `PagamentoPix`.
     - Método que recebe um tipo mais genérico (superclasse ou interface) e trata as subclasses.
5. **Interface e/ou classe abstrata**
   - Pelo menos **uma interface** *ou* **uma classe abstrata** com implementação concreta.
6. **Coleções**
   - Uso de `List`, `Set` ou `Map` para armazenar objetos de domínio.
7. **Tratamento de exceções**
   - Uso de `try/catch` em pontos críticos (entrada de dados do usuário, operações que podem falhar).
   - Opcional: exceção customizada (conta ponto extra).

4. Requisitos técnicos mínimos

1. **Organização em pacotes**
   - Mínimo recomendado:
     - `model` ou `domain` – classes de negócio
     - `service` – regras de negócio
     - `view` ou `ui` – interface com o usuário
     - `repository` – armazenamento (mesmo que seja em memória)
2. **Persistência de dados (ajustada para 3 semanas)**
   - **Obrigatório (mínimo):**
     - Persistência em **arquivo** (texto, CSV ou serialização simples) **ou**
     - Armazenamento em memória, mas com estrutura organizada (coleções), permitindo fácil inclusão futura de persistência.
   - **Opcional / ponto extra:** uso de banco de dados simples (ex.: SQLite, H2, MySQL com JDBC).
3. **Boas práticas**
   - Nomes claros para classes, métodos e variáveis.
   - Métodos com tamanho razoável.
   - Código organizado, sem blocos gigantes de lógica duplicada.

5. Documentação mínima

1. **README.md** (ou documento equivalente) contendo:
   - Nome do projeto.
   - Nome do aluno.
   - Descrição resumida do sistema.
   - Tecnologias utilizadas (Java versão X, etc.).
   - Como compilar e executar o projeto.
   - Um exemplo de fluxo de uso (passo a passo).
2. **Diagrama de classes simplificado**
   - Mostrando:
     - Principais classes
     - Relações (associações/composição)
     - Heranças
   - Pode ser em imagem (PlantUML, draw.io, etc.).
3. **Descrição de pelo menos 3 casos de uso**, em texto simples:

- Ex.:
  - Caso de uso 1: “Cadastrar cliente”
  - Caso de uso 2: “Registrar nova venda”
  - Caso de uso 3: “Listar vendas de um cliente”

6. Critérios de avaliação (sugeridos)

- **Modelagem OO (encapsulamento, herança, polimorfismo, composição, interfaces)** – 35%
- **Qualidade do código e organização em pacotes** – 25%
- **Funcionalidade do sistema (MVP funcionando)** – 25%
- **Documentação (README, diagrama, casos de uso)** – 15%