TRABALHO DE PROGRAMAÇÃO ORIENTADA A OBJETOS I (POO I)

Tema: Desenvolvimento de Jogos com Interface Textual em Java

Objetivo principal:

O principal objetivo deste trabalho é aplicar de forma prática os principais conceitos de Programação Orientada a Objetos, por meio do desenvolvimento de um jogo simples em Java. Neste trabalho, você deverá desenvolver seu jogo com interface exclusivamente textual. Isso não é uma limitação - é uma escolha pedagógica intencional. Utilizar interface gráfica, além de ainda não fazer parte do conteúdo da disciplina, pode desviar o foco para detalhes visuais e técnicos que não são o objetivo de POOI. A proposta é que você se concentre em dominar os fundamentos da POO.

- O que será valorizado neste trabalho é a clareza da modelagem, a estrutura do código e a aplicação correta dos conceitos de POO. Uma interface textual bem organizada, com menus claros e boa interação, demonstra o seu cuidado com o usuário.
- Use essa oportunidade para pensar como um programador profissional: planeje, modele, teste, refatore. A parte visual virá no tempo certo.

Requisitos Funcionais:

O jogo pode seguir qualquer tema escolhido pelo grupo (RPG, aventura interativa, quiz com níveis, batalha de criaturas, jogo de palavras, entre outros ou seguir os temas sugeridos no item **Sugestões de Jogos**), **desde que atenda aos requisitos técnicos obrigatórios e considerações descritas a seguir**.

Requisitos Técnicos (Obrigatórios):

O jogo deve contemplar de forma clara e funcional os seguintes conceitos de POO:

- Encapsulamento e Ocultação de Informações.
- Herança.
- Interface.
- Classe abstrata.
- Polimorfismo.
- Agregação/Composição: uso de agregação e/ou composição.
- Tratamento de Exceções (TE): acrescentar TE no código de seu aplicativo incluindo ao menos duas exceções personalizadas.
- Estruturas de dados: uso de no mínimo um tipo de estrutura de dados como ArrayList, HashMap, entre outros.
- Arquivos: uso de arquivos para entrada de dados e/ou saída de dados.
- Documentação: uso do Javadoc.

Considerações:

1. Definir e entregar o diagrama de classes UML do aplicativo.
2. Construir a interface de Usuário de um modo simples (só textual).
3. Certifique-se de que a lógica do jogo possa ser reusada independentemente da interface com o usuário.

4. Construir as classes/interfaces do jogo em pacotes.
5. Propor e implementar uma funcionalidade extra ao aplicativo. Essa funcionalidade adicional deve ser discutida e definida pelos membros da equipe, aproveitando suas habilidades e criatividade. Cada equipe deve incluir a funcionalidade extra na versão final do projeto e documentar o processo de desenvolvimento, incluindo: descrição detalhada da funcionalidade extra; justificativa da escolha da funcionalidade e passos de implementação.
6. Não utilizar bibliotecas gráficas (Swing, JavaFX, etc).
    a. Interface deve ser textual
7. O aplicativo deve ser documentado utilizando as tags do JavaDoc
    a. Documentação deve ter @author, @version, @param, @return, @exception.
8. Siga as boas práticas de programação do Java:
    a. Nomes de pacotes, classes, interfaces, atributos e métodos devem seguir o padrão de nomenclatura do Java.
9. Os arquivos entregues devem incluir:
    - Código-fonte completo em Java (.java)
    - Arquivo README.txt
        a. Descrição do jogo
        b. Como executar
        c. Conceitos aplicados
        d. Detalhamento do item criativo
10. O projeto deve ser entregue no Moodle por meio do link do repositório no GitHub.

Informações importantes:

- *Data e horário de entrega do trabalho: 16/06 até às 23h59min*
- *Data de apresentação do trabalho: 23/06 (segunda-feira), 25/06(quarta-feira). Formação de equipes: equipes de até três estudantes.*
- ***Critérios para validade do trabalho**:*
    - *Informações até dia 28 de maio (quarta-feira)* ***- 100% do valor do trabalho**.*
        - *Membros da equipe.*
        - *Jogo a ser desenvolvido.*
    - *Informações após 28 de maio (quarta-feira) –* ***90% do valor do trabalho.***
    - *Postagem do trabalho no Moodle por meio do link do repositório no GitHub:*
        - Código postado até o dia 16/06/2025 – **100% do valor do trabalho.**
        - Código postado após o dia 16/06/2025 e/ou trabalho apresentado em datas diferentes das programadas – **60% do valor do trabalho**.
    - *Apresentar individualmente o trabalho para a professora da disciplina na data e horário agendado.*

Avaliação:

A nota será atribuída com base nos seguintes critérios:

- Implementação dos conceitos obrigatórios de POO: 60%
- Funcionamento do jogo e lógica geral: 10%
- Organização do código e boas práticas: 10%
- Criatividade e funcionalidade diferenciada: 20%

Sugestões de Jogos

A seguir estão quatro sugestões de jogos. Todos eles devem ser desenvolvidos em Java com foco total na aplicação dos conceitos de Programação Orientada a Objetos e em conformidade com as orientações descritas nas páginas anteriores. Cada sugestão de jogo contém:

- Descrição geral do trabalho
- Requisitos mínimos obrigatórios (com foco nos conceitos de POO)
- Sugestão de estrutura (opcional)

1. Jogo RPG por Turnos

Descrição Geral: implemente um jogo de RPG em texto no qual o jogador escolhe uma classe (Mago, Guerreiro ou Arqueiro), enfrenta inimigos em batalhas por turnos, usa itens e evolui seu personagem com experiência. O jogo deve permitir salvar e carregar o progresso.

Requisitos obrigatórios:

- Classes com herança (exemplo: Personagem base e subclasses).
- Interface – exemplo para verificações de ações dos personagens.
- Encapsulamento com atributos privados e métodos públicos.
- Polimorfismo nas ações dos personagens e inimigos.
- Agregação ou composição (exemplo: Inventário, Itens).
- Tratamento de exceções, definição de no mínimo duas exceções personalizadas.
- Uso de collections (exemplo: listas de inimigos, itens, habilidades).
- Leitura/escrita em arquivos (exemplo: salvar o progresso do jogo em um arquivo).

Sugestão de estrutura (opcional):

- Classe Personagem (abstrata), subclasses Mago, Guerreiro, Arqueiro.
- Interface AcaoCombate com método executarAcao().
- Classe Inimigo, Item, Inventario.
- Classe Batalha para lógica de turno.
- Classe Jogo com menu e fluxo principal.
- Utilizar ArrayList para lista de itens e inimigos.
- Salvar em um arquivo de texto algumas informações do jogo.

2. Jogo de Embaralhamento de Palavras

Descrição Geral: implemente um jogo onde o sistema embaralha palavras e o jogador tenta acertar a forma correta. O jogo deve ter níveis de dificuldade e pontuação acumulada.

Requisitos obrigatórios:

- Herança – exemplo entre tipos de palavras ou desafios.
- Interface – exemplo para verificação de acertos.
- Encapsulamento de dados do jogador e das rodadas.
- Polimorfismo no embaralhamento (exemplo diferentes estratégias).

- Agregação: exemplo: Lista de rodadas jogadas associada ao jogador ou Composição: exemplo Classe Rodada composta por Palavra, Dificuldade, Resultado.
- Tratamento de exceções, definição de no mínimo duas exceções personalizadas.
- Uso de *Collection Framework* (exemplo lista de palavras e rodadas).
- Arquivos: exemplo leitura de palavras de arquivos.

Sugestão de estrutura (opcional):
- Classe Palavra, Rodada, Jogador.
- Interface Verificavel com método verificarResposta().
- Subclasses de Palavra para diferentes níveis.
- Classe Jogo com menu, pontuação e progresso.
- Lista de palavras em ArrayList, carregadas de arquivo .txt.

3. Aventura com Escolhas (Estilo Livro-Jogo)

Descrição Geral: implemente uma aventura interativa em que o jogador toma decisões que alteram o rumo da história. As cenas podem conter batalhas, enigmas ou diálogos. O jogo pode ter múltiplos finais.

Requisitos obrigatórios:
- Herança entre tipos de cenas (exemplo: Batalha, Enigma, Diálogo).
- Interface – exemplo para ações ou decisões interativas.
- Encapsulamento do progresso e variáveis do jogo.
- Polimorfismo nas cenas.
- Agregação de cenas formando uma história encadeada.
- Tratamento de exceções, definição de no mínimo duas exceções personalizadas.
- Uso de *Collection Framework* (exemplo: lista de cenas, caminhos).
- Arquivos: exemplo - salvar em arquivo o roteiro da aventura ou ler a estrutura da história via arquivo.

Sugestão de estrutura (opcional):
- Classe Cena (abstrata), com subclasses CenaBatalha, CenaEnigma
- Interface Interativa com método executarCena()
- Classe História com sequência de cenas e decisões
- Classe Jogador para guardar escolhas e progresso
- Estrutura de árvore ou lista encadeada para fluxo da história
- Arquivo .txt para armazenar o roteiro da aventura

4. Quiz com Níveis e Ranking
Descrição Geral: desenvolva um jogo de perguntas com múltiplos níveis. As perguntas podem ser de múltipla escolha ou abertas. O jogador acumula pontos e aparece em um ranking ao final.

Requisitos obrigatórios:

- Herança: exemplo entre tipos de perguntas (Pergunta base e subclasses PerguntaMultipla, PerguntaAberta).
- Interface: exemplo para validação de respostas.
- Encapsulamento: exemplo – controle das perguntas, respostas, pontuação.
- Polimorfismo: exemplo - validação de respostas diferentes para cada tipo de pergunta .
- Agregação: exemplo - Jogador com lista de respostas, perguntas associadas ao nível e/ou Composição: exemplo quiz composto por lista de perguntas.
- Tratamento de exceções, definição de no mínimo duas exceções personalizadas.
- Uso de *Collection Framework*: exemplo lista de perguntas, jogadores, ranking.
- Arquivos: exemplo de Leitura de perguntas, gravação de pontuação e ranking final.

Sugestão de estrutura (opcional):

- Classe Pergunta (abstrata), subclasses PerguntaMultipla, PerguntaAberta.
- Interface Validavel com método validarResposta().
- Classe Quiz, Jogador, Ranking.
- Utilizar ArrayList para perguntas e jogadores.
- Arquivos .txt para armazenar perguntas e pontuação final.