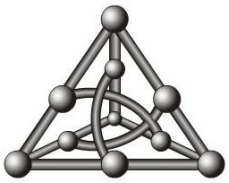

Universidade Federal de Mato Grosso do Sul
Faculdade de Computação

Disciplina: Linguagem de Programação Orientada a Objetos
Professor: Renan Albuquerque Marks

Descrição do Trabalho Prático

1 Introdução

Neste documento estão detalhados os procedimentos que devem ser seguidos para o desenvolvimento do Trabalho Prático e constituirá como parte da nota final da disciplina de LPOO.

É fortemente recomendado que os estudantes acessem com freqüência este documento para esclarecer possíveis dúvidas, estar ciente do cronograma e estar a par de possíveis atualizações/alterações no trabalho.

2 Objetivo

O objetivo deste trabalho é implementar em **linguagem de programação Java** um sistema de combates de um jogo de RPG (*Role Playing Game*). Para isso, deve ser implementado a possibilidade de escolha entre três personagens principais, cada um possuindo armas e habilidades únicas.

3 Personagens

Inicialmente, estarão presentes para escolha três tipos de personagens: **Mago**, **Paladino** e **Clérigo**. Todos possuem três status: saúde, força e destreza com pontos variando de 0.0 a 10.0. Quando os pontos de saúde de um personagem se tornam menor que 1.0, ele é declarado morto. Além disso, também são capazes de desferir ataques e se defender quando atacados.

3.1 Armas

Cada personagem é capaz de carregar uma arma compatível com seu perfil, isto é: um Mago não pode carregar uma arma de um Paladino ou Clérigo (e vice-versa). Além disso, cada arma possui um **modificador de força de ataque** cujo valor varia de 0.0 a 1.0, ou seja: de 0 a 100%. A seguir, listamos as armas disponíveis para cada classe com os respectivos códigos de cada uma:

1. Mago:
    1. **Magia da transmutação:** reduz o cálcio dos ossos do oponente, deixando-os mais porosos. Modificador: 0.25.
    2. **Psi-kappa:** com o poder de telecinese, arremessa o oponente à distância. Modificador: 0.5.
2. Paladino:
    1. **Espada:** possui uma lamina afiada mas é frágil. Modificador: 0.3.
    2. **Lança:** Ataques em longa distância. Modificador: 0.5.

3. Clérigo:

    1. **Martelo:** apesar de pesado, é uma arma letal. Modificador: 0.6.
    2. **Maça:** por ser mais leve, é mais fácil de manusear. Modificador: 0.4.

O diagrama UML que descreve a relação das classes projetadas para este sistema de RPG pode ser visualizado na Figura 1. As classes que possuem nomes em *itálico* são classes *abstratas*, isto é, classes que não podem ser instanciadas e servem apenas como classes-base (ou superclasses) que devem ser herdadas por outras classes concretas, representadas pelo **nome em negrito**.

3.2 Batalha

Ao ser criado, o personagem deve receber o nome de seu tipo ("Mago", "Paladino" ou "Clérigo"), o valor de seus status iniciais (saúde, força e destreza) e carregar uma arma compatível para usar (Arma de Mago, de Paladino ou de Clérigo). Essas invariantes devem ser definidas e validadas **construtor** das classes concretas dos personagens. De forma a simplificar a lógica da batalha, temos:

1. Quando um personagem A ataca um personagem B:

    - Verifique se o personagem **A está vivo**:
        - **Se o personagem A estiver vivo**, deve ser impressa a mensagem "`O <personagem A> ataca o <personagem B> com <arma>.`", por exemplo: "`O Mago ataca o Paladino com Psi-kappa.`"
        - **Se o personagem A estiver morto**, deve ser impressa a mensagem "`O <personagem A> não consegue atacar, pois está morto.`";
    - O ataque é **bem sucedido** (**não é defendido** por B) quando A possui **força e destreza maiores** que as respectivas força e destreza de B:
        - **Se B estiver vivo**:
            * Então a saúde do personagem B deve ser subtraída pela quantidade de pontos de força do personagem A × modificador de ataque da arma de A.
            * Deve ser impressa a mensagem: "`O ataque foi efetivo com X pontos de dano!`".
        - **Se B estiver morto**, deve ser impressa a mensagem: "`Pare! O <personagem B> já está morto!`".
    - O ataque é **mal sucedido** (**é defendido e revidado** por B) quando A possui **força ou destreza menores** que as respectivas força ou destreza de B:
        - **Se B estiver vivo**:
            * Então a saúde do personagem A deve ser subtraída pela quantidade de pontos de força do personagem B × modificador de ataque da arma de B.
            * Deve ser impressa a mensagem: "`O ataque foi inefetivo e revidado com X pontos de dano!`".
        - **Se B estiver morto**, deve ser impressa a mensagem: "`Pare! O <personagem B> já está morto!`".
    - O ataque é **defendido** (**é anulado** por B) quando A possui força **ou** destreza **iguais** às respectivas força **ou** destreza de B:
        - **Se B estiver vivo**:

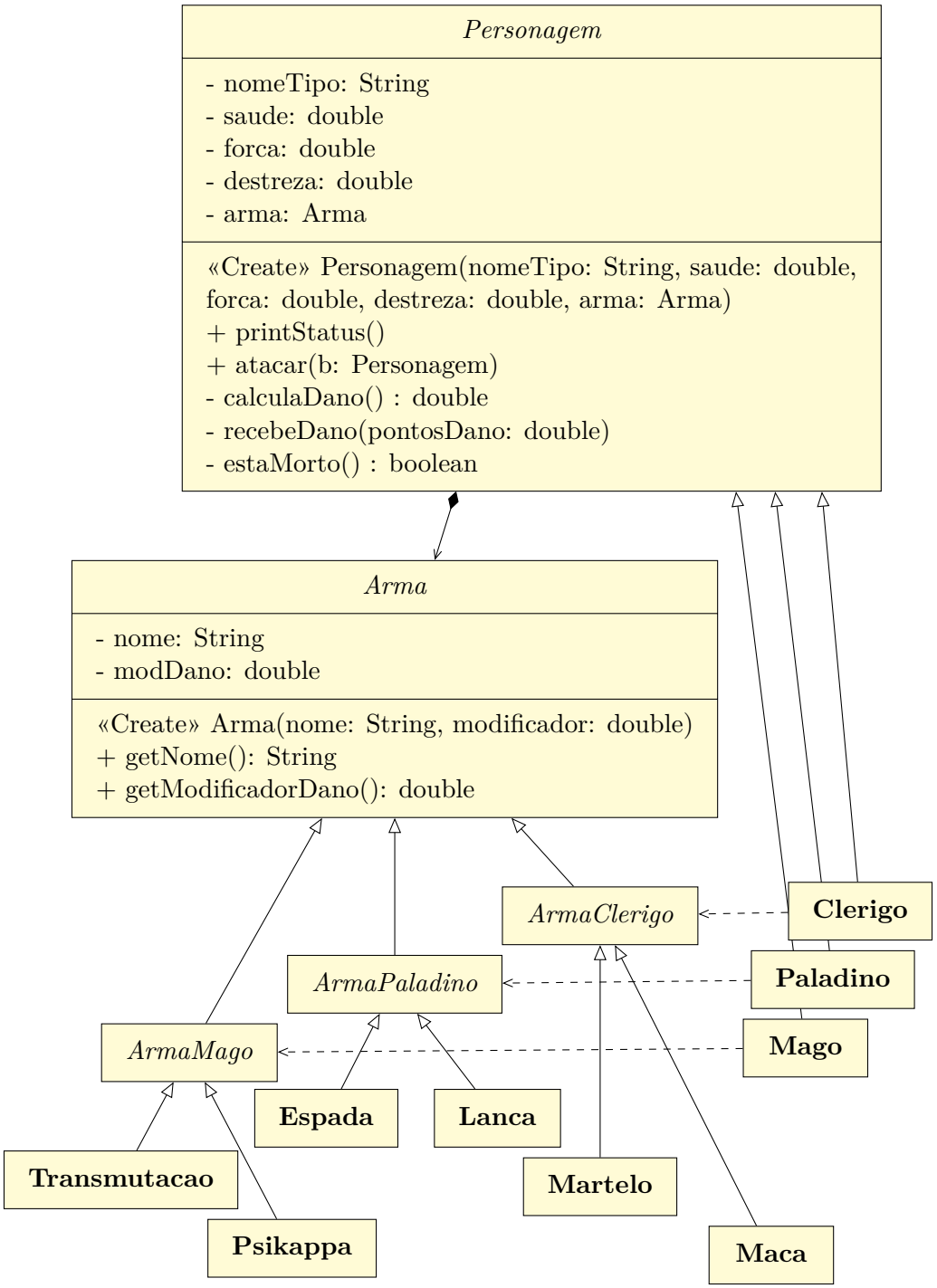

Figura 1: Diagrama de classe UML para implementação dos personagens de RPG.

        * Então nem A nem B sofrem danos nas suas respectivas saúdes.
        * Deve ser impressa a mensagem: "`O ataque foi defendido, ninguem se machucou!`"
    - **Se B estiver morto**, deve ser impressa a mensagem: "`Pare! O <personagem B> já está morto!`".

2. Após um personagem A atacar um personagem B:

    - Deve ser impresso na tela o status dos personagens A e B, por exemplo:

      ```
      Mago [Saude: 10.0, Forca: 6.0, Destreza: 4.0, Magia da Transmutacao]
      Paladino [Saude: 10.0, Forca: 5.0, Destreza: 6.0, Espada]
      ```

    - Caso um personagem tenha morrido, o status deve ser impresso como:

      ```
      Mago [Saude: 10.0, Forca: 6.0, Destreza: 4.0, Magia da Transmutacao]
      Paladino [Morreu, Forca: 5.0, Destreza: 6.0, Espada]
      ```

4 Entrada

O programa deve receber como entrada as informações necessárias para criar dois personagens com seus pontos de saúde, força, destreza e arma carregada. Os códigos de cada personagem e respectiva arma foram listados na Seção 3.1. Logo após, são especificados os índices dos personagens que serão o atacante e o defensor.

Por exemplo, para criar dois personagens:

- Mago com 10 pontos de saúde, 6 de força, 4 de destreza com Magia da Transmutação;
- Paladino com 10 pontos de saúde, 5 de força, 6 de destreza com Espada;

Teríamos como entrada:

```
1 10 6 4 1
2 10 5 6 1
```

E seria impresso:

```
Mago [Saude: 10.0, Forca: 6.0, Destreza: 4.0, Magia da Transmutacao]
Paladino [Saude: 10.0, Forca: 5.0, Destreza: 6.0, Espada]
```

A seguir, o programa esperaria como entrada a sequencia de ataques. A sequencia de ataques deve ser informada com dois números: o primeiro número do personagem atacante e o segundo número do personagem defensor. Por exemplo, para informar o ataque do Mago contra o Paladino, a entrada seria:

```
1 2
```

Para informar o fim da batalha, a entrada deve ser informada com o número zero:

```
0
```

5 Atribuição de grupos

O trabalho deverá ser feito individualmente.

6 Cronograma

- **Início dos trabalhos:** 23/05/2024;
- **Entrega:** 23/06/2024 — Submeter via AVA o código junto de um relatório que contenha:
  - Explicação da organização do código:
    - Clareza de comentários;
  - Dificuldades encontradas;
  - Soluções de implementação;

7 Avaliação do trabalho

- **Nota da Versão Final:** valor no intervalo [0,10] que será atribuído de acordo com com o cumprimento das atividades solicitadas para a versão da entrega final. Também serão considerados: organização e comentários no código.

Atenção: casos de plágio serão tratados com extremo rigor.

8 Dicas e Sugestões

- Inicie o trabalho o **quanto antes**. O tempo **voa**!
- Retire as dúvidas quanto ao entendimento dos elementos que compõem o trabalho. Isso possibilitará detectar possíveis falhas a tempo de corrigí-las.
- Cheque com frequência a documentação de ajuda do Java; Muitas informações úteis podem ser encontradas lá;