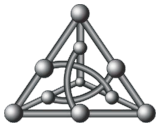


Universidade Federal de Mato Grosso do Sul - UFMS
Faculdade de Computação - FACOM

Arthur Silva Dantas
Graziela Santos de Araújo

Dato: 17 de Novembro de 2024

Relatório: Trabalho Prático - Linguagem de Programação Orientada a Objetos

Olá,

Este relatório destina-se ao trabalho prático da disciplina de Linguagem de Programação Orientada a Objetos, onde o projeto simula um jogo de RPG utilizando os princípios de Orientação a Objetos e aplicando o conteúdo abordado ao longo da disciplina.

Para obter mais informações sobre o código completo e consultar uma parte adicional da documentação, o projeto está disponível no GitHub.
Disponível em: `https://github.com/Arthur-SD15/LPOO-RolePlayingGame`.

Atenciosamente,
Arthur Dantas

Sumário

1. Projeto Geral

O projeto, de maneira geral, consiste em códigos e documentações de um trabalho prático da disciplina de Linguagem de Programação Orientada a Objetos, desenvolvido na Universidade Federal de Mato Grosso do Sul. Ele simula um Role Playing Game (RPG), um jogo via linha de comando, no qual foram aplicados os fundamentos da Orientação a Objetos utilizando a linguagem Java. O código contém comentários explicativos sobre as operações realizadas e inclui um relatório, onde se lê, tal documento, o objetivo é documentar o projeto com maiores detalhes.

2. Organização Geral

Nesta seção, começamos a detalhar e explorar a estrutura do projeto e, consequentemente, do código. No entanto, a análise detalhada do código ficará para a seção posterior. O objetivo desta seção é abordar a estrutura e alguns conceitos teóricos, a fim de facilitar a compreensão da próxima seção.

Antes de tudo, o projeto segue rigorosamente os princípios da Orientação a Objetos, contendo 15 arquivos .java com **classes abstratas, concretas e hierarquias de herança**. Antes que eu me esqueça, acho importante destacar que, de maneira geral, todos os atributos são privados. Durante a minha implementação o código em si, não foi necessário, e consequentemente não foi feito uso de sobrecarga, nem mesmo nos construtores. Além disso, não foi utilizado o método @Override para sobrescrita. De maneira geral, foram trabalhados, em especial, o polimorfismo, o encapsulamento e a abstração.

Como apoio, foi utilizado um Diagrama de Classes da UML (Unified Modeling Language), que havia sido disponibilizado pela professora e serviu como base ao longo do desenvolvimento.

Durante o projeto, os conceitos de Orientação a Objetos são evidentes, permitindo identificar relações de **herança**, como a herança da classe Personagem para as classes Clérigo, Paladino e Mago, que são suas subclasses. Vale destacar, que a herança tem por objetivo a reutilização de código permitindo que as subclasses herdem atributos e métodos das classes pai, além de poder adicionar ou modificar funcionalidades específicas.

Uma observação é que não existe apenas essa herança dentro do projeto. Assim como também não existe apenas a **classe abstrata** Personagem, que tem por objetivo servir como molde para outras **classes concretas**.

3. Organização Detalhada

Confira a organização detalhada, com base em alguns conceitos teóricos e entrando a fundo em cada código implementado.

3.1. Personagem.java

Neste arquivo, temos a classe abstrata Personagem, que serve como classe pai para todas as outras classes de personagens e é utilizada como molde.

A classe contém atributos privados relacionados ao personagem, acompanhados de seus respectivos métodos getters e setters. Também inclui o construtor da classe, o método printStatus, que imprime o status do personagem e o método atacar baseado em diversas condições, que será detalhado na seção 4, ao tratar das dificuldades encontradas. O método calculaDano, que calcula o dano causado pelo personagem; o receberDano, que calcula o dano recebido pelo personagem e por fim o estaMorto, que verifica se o personagem está morto.

Código completo, com comentários:

Listing 1: Classe abstrata `Personagem`

```
/*
 * Importamos as classes necessarias: BigDecimal e RoundingMode.
 * BigDecimal: permite trabalhar com numeros decimais com precisao.
 * RoundingMode: permite trabalhar com arredondamento de numeros decimais.
*/
import java.math.BigDecimal;
import java.math.RoundingMode;

// Classe abstrata Personagem.
public abstract class Personagem {
    // Atributos da classe Personagem.
    private String nomeTipo;
    private double saude;
    private double forca;
    private double destreza;
    private Arma arma;

    // Construtor da classe Personagem.
```

```
19      public Personagem(String nomeTipo, double saude, double forca, double
            destreza, Arma arma) {
20          this.nomeTipo = nomeTipo;
21          this.saude = saude;
22          this.forca = forca;
23          this.destreza = destreza;
24          this.arma = arma;
25      }
26
27      // Metodos getters e setters da classe Personagem.
28      public String getNomeTipo() { return this.nomeTipo; }
29      public double getSaude() { return this.saude; }
30      public double getForca() { return this.forca; }
31      public double getDestreza() { return this.destreza; }
32      public Arma getArma() { return this.arma; }
33      public void setNomeTipo(String nomeTipo) { this.nomeTipo = nomeTipo; }
34      public void setSaude(double saude) { this.saude = saude; }
35      public void setForca(double forca) { this.forca = forca; }
36      public void setDestreza(double destreza) { this.destreza = destreza; }
37      public void setArma(Arma arma) { this.arma = arma; }
38
39      /*
40       * Metodo printStatus que imprime o status do personagem.
41       * Utiliza getters para acessar os atributos privados.
42      */
43      public void printStatus() {
44          if (estaMorto()) {
45              System.out.printf(getNomeTipo() + " [Morto, Forca: %.1f,
                    Destreza: %.1f, %s]\n",
46                   getForca(), getDestreza(), getArma().getNome());
47          } else {
48              System.out.printf(getNomeTipo() + " [Saude: %.1f, Forca: %.1f,
                    Destreza: %.1f, %s]\n",
49                   getSaude(), getForca(), getDestreza(), getArma().getNome())
                         ;
50          }
51      }
52
53      /*
54       * Metodo atacar que realiza um ataque a outro personagem.
55       * Possui condicoes baseadas na destreza dos personagens.
56      */
57      public void atacar(Personagem b) {
58          if (estaMorto()) {
59              System.out.println("O " + getNomeTipo() + " nao pode atacar,
                    pois esta morto.");
60              return;
61          } else {
62              System.out.println("O " + getNomeTipo() + " ataca o " + b.
                    getNomeTipo() + " com " + getArma().getNome() + ".");
63          }
64
65          if (b.estaMorto()) {
```

```
66              System.out.println("Pare! O " + b.getNomeTipo() + " ja esta
                    morto!");
67              return;
68          }
69
70          if (getDestreza() > b.getDestreza()) {
71              double auxDanoSofrido = calculaDano();
72              System.out.printf("O ataque foi efetivo com %.1f pontos de dano
                    !\n", auxDanoSofrido);
73              b.receberDano(auxDanoSofrido);
74          } else if (getDestreza() < b.getDestreza()) {
75              double auxDanoSofrido = b.calculaDano();
76              System.out.printf("O ataque foi inefetivo e revidado com %.1f
                    pontos de dano!\n", auxDanoSofrido);
77              receberDano(auxDanoSofrido);
78          } else {
79              System.out.println("O ataque foi defendido, ninguem se machucou
                    !");
80          }
81      }
82
83      /*
84       * Metodo calculaDano que calcula o dano causado.
85       * Utiliza BigDecimal para precisao no calculo.
86      */
87      private double calculaDano() {
88          BigDecimal dano = new BigDecimal(getForca() * getArma().getModDano
                ());
89          return dano.setScale(1, RoundingMode.HALF_UP).doubleValue();
90      }
91
92      /*
93       * Metodo receberDano que aplica o dano recebido ao personagem.
94      */
95      private double receberDano(double pontosDano) {
96          this.saude -= pontosDano;
97          return this.saude;
98      }
99
100     /*
101      * Metodo estaMorto que verifica se o personagem esta morto.
102      * Retorna true se a saude for menor que 1, caso contrario, false.
103     */
104     private boolean estaMorto() {
105         return getSaude() < 1;
106     }
107 }
```

3.2. Arma.java

A classe Arma possui semelhanças com a classe Personagem, sendo uma classe abstrata que serve como classe molde para outras classes. Ela contém dois atributos privados, que são acessados por meio de métodos getters. Além disso, a classe possui um construtor que recebe dois parâmetros: o nome da arma e o modificador.

Código completo, com comentários:

Listing 2: Classe Abstrata Arma

```
// Classe abstrata Arma.
public abstract class Arma{
    // Atributos da classe Arma.
    private String nome;
    private double modDano;

    // Construtor da classe Arma.
    public Arma(String nome, double modDano){
        this.nome = nome;
        this.modDano = modDano;
    }

    // Metodos getters da classe Arma.
    public String getNome(){
        return this.nome;
    }

    public double getModDano(){
        return this.modDano;
    }
}
```

3.3. Mago.java

Este arquivo é responsável por criar a classe Mago, que é uma subclasse de Personagem. Dentro dessa classe, temos o construtor da classe Mago, que recebe os atributos do personagem e a arma. O construtor da superclasse é utilizado para inicializar os atributos do personagem.

Código completo, com comentários:

Listing 3: Classe Mago, subclasse de Personagem

```
/*
 * Criamos a classe Mago que e uma subclasse de Personagem, herdando os atributos e metodos de personagem.
 * Essa classe e responsavel por criar um personagem do tipo Mago.
 * Dentro dessa classe, temos o construtor da classe Mago que recebe os valores de saude, forca, destreza e a arma.
 * Utilizamos o construtor da superclasse para inicializar os atributos de personagem.
*/
public class Mago extends Personagem{
    public Mago(double saude, double forca, double destreza, ArmaMago arma) {
        super("Mago", saude, forca, destreza, arma);
    }
}
```

3.4. Paladino.java

Este arquivo é responsável por criar a classe Paladino, que é uma subclasse de Personagem. Dentro dessa classe, temos o construtor da classe Paladino, que recebe os atributos do personagem e a arma. O construtor da superclasse é utilizado para inicializar os atributos do personagem.

Código completo, com comentários:

Listing 4: Classe Paladino, subclasse de Personagem

```
/*
 * Criamos a classe Paladino que e uma subclasse de Personagem, herdando os atributos e metodos de personagem.
 * Essa classe e responsavel por criar um personagem do tipo Paladino.
 * Dentro dessa classe, temos o construtor da classe Paladino que recebe os valores de saude, forca, destreza e a arma.
 * Utilizamos o construtor da superclasse para inicializar os atributos de personagem.
*/
public class Paladino extends Personagem {
    public Paladino(double saude, double forca, double destreza, ArmaPaladino arma) {
        super("Paladino", saude, forca, destreza, arma);
    }
}
```

3.5. Clerigo.java

Este arquivo é responsável por criar a classe Clérigo, que é uma subclasse de Personagem. Dentro dessa classe, temos o construtor da classe Clérigo, que recebe os atributos do personagem e a arma. O construtor da superclasse é utilizado para inicializar os atributos do personagem.

Código completo, com comentários:

Listing 5: Classe Clerigo, subclasse de Personagem

```
1  /*
2   * Criamos a classe Clerigo que e uma subclasse de Personagem, herdando os
        atributos e metodos de personagem.
3   * Essa classe e responsavel por criar um personagem do tipo Clerigo.
4   * Dentro dessa classe, temos o construtor da classe Clerigo que recebe os
        valores de saude, forca, destreza e a arma.
5   * Utilizamos o construtor da superclasse para inicializar os atributos de
        personagem.
6  */
7  public class Clerigo extends Personagem{
8      public Clerigo(double saude, double forca, double destreza, ArmaClerigo
           arma){
9          super("Clerigo", saude, forca, destreza, arma);
10     }
11 }
```

3.6. ArmaMago.java

Este arquivo é responsável por criar a classe ArmaMago, que é uma subclasse de Arma. Dentro dessa classe, temos o construtor da classe ArmaMago, que recebe os atributos da arma. O construtor da superclasse é utilizado para inicializar os atributos da arma.

Código completo, com comentários:

Listing 6: Classe ArmaMago, subclasse de Arma

```
1  /*
2   * Criamos uma classe abstrata ArmaMago que e uma subclasse de Arma,
        herdando os atributos de arma.
3   * Essa classe e responsavel por criar uma ArmaMago do tipo Arma.
4   * Dentro dessa classe, temos o construtor da classe ArmaMago que recebe o
        nome e o modificador de dano.
5   * Utilizamos o construtor da superclasse para inicializar os atributos de
        arma.
6  */
7  public abstract class ArmaMago extends Arma{
8      public ArmaMago(String nome, double modDano){
```

```
9           super(nome, modDano);
10      }
11  }
```

3.7. ArmaPaladino.java

Este arquivo é responsável por criar a classe ArmaPaladino, que é uma subclasse de Arma. Dentro dessa classe, temos o construtor da classe ArmaPaladino, que recebe os atributos da arma. O construtor da superclasse é utilizado para inicializar os atributos da arma.

Código completo, com comentários:

Listing 7: Classe ArmaPaladino, subclasse de Arma

```
1  /*
2   * Criamos uma classe abstrata ArmaPaladino que e uma subclasse de Arma,
        herdando os atributos de arma.
3   * Essa classe e responsavel por criar uma ArmaPaladino do tipo Arma.
4   * Dentro dessa classe, temos o construtor da classe ArmaPaladino que
        recebe o nome e o modificador de dano.
5   * Utilizamos o construtor da superclasse para inicializar os atributos de
        arma.
6   */
7  public abstract class ArmaPaladino extends Arma{
8      public ArmaPaladino(String nome, double modDano){
9          super(nome, modDano);
10     }
11 }
```

3.8. ArmaClerigo.java

Este arquivo é responsável por criar a classe ArmaClerigo, que é uma subclasse de Arma. Dentro dessa classe, temos o construtor da classe ArmaClerigo, que recebe os atributos da arma. O construtor da superclasse é utilizado para inicializar os atributos da arma.

Código completo, com comentários:

Listing 8: Classe ArmaClerigo, subclasse de Arma

```
/*
 * Criamos uma classe abstrata ArmaClerigo que e uma subclasse de Arma, herdando os atributos de arma.
 * Essa classe e responsavel por criar uma ArmaClerigo do tipo Arma.
 * Dentro dessa classe, temos o construtor da classe ArmaClerigo que recebe o nome e o modificador de dano.
 * Utilizamos o construtor da superclasse para inicializar os atributos de arma.
*/
public abstract class ArmaClerigo extends Arma{
    public ArmaClerigo(String nome, double modDano){
        super(nome, modDano);
    }
}
```

3.9. Transmutação.java

Este arquivo é responsável por criar a classe Transmutacao, que é uma subclasse de ArmaMago. Dentro dessa classe, temos o construtor da classe Transmutacao, que recebe os atributos da arma, como o nome e o modificador de dano. O nome da arma é "Magia da Transmutação"e o modificador de dano é 0.25. O construtor da superclasse ArmaMago é utilizado para inicializar esses atributos.

Código completo, com comentários:

Listing 9: Classe Transmutacao, subclasse de ArmaMago

```
/*
 *  Criamos a classe Transmutacao que e uma subclasse de ArmaMago, herdando os atributos de ArmaMago.
 *  Essa classe e responsavel por criar uma Arma de Mago do tipo Transmutacao.
 *  Dentro dessa classe, temos o construtor da classe Transmutacao que recebe o nome e o dano da arma.
 *  Utilizamos o construtor da superclasse para inicializar os atributos de ArmaMago.
*/
public class Transmutacao extends ArmaMago {
    public Transmutacao() {
        super("Magia da Transmutacao", 0.25);
    }
}
```

3.10. Psikappa.java

Este arquivo é responsável por criar a classe Psikappa, que é uma subclasse de ArmaMago. Dentro dessa classe, temos o construtor da classe Psikappa, que recebe os atributos da arma, como o nome e o modificador de dano. O nome da arma é "Psi-kappa"e o modificador de dano é 0.5. O construtor da superclasse ArmaMago é utilizado para inicializar esses atributos.

Código completo, com comentários:

Listing 10: Classe Psikappa, subclasse de ArmaMago

```
/*
 *  Criamos a classe Psikappa que e uma subclasse de ArmaMago, herdando os
    atributos de ArmaMago.
 *  Essa classe e responsavel por criar uma Arma de Mago do tipo Psikappa.
 *  Dentro dessa classe, temos o construtor da classe Psikappa que recebe o
     nome e o dano da arma.
 *  Utilizamos o construtor da superclasse para inicializar os atributos de
     ArmaMago.
*/
public class Psikappa extends ArmaMago {
    public Psikappa() {
        super("Psi-kappa", 0.5);
    }
}
```

3.11. Espada.java

Este arquivo é responsável por criar a classe Espada, que é uma subclasse de ArmaPaladino. Dentro dessa classe, temos o construtor da classe Espada, que recebe os atributos da arma, como o nome e o modificador de dano. O nome da arma é "Espada"e o modificador de dano é 0.3. O construtor da superclasse ArmaPaladino é utilizado para inicializar esses atributos.

Código completo, com comentários:

Listing 11: Classe Espada, subclasse de ArmaPaladino

```
/*
 *  Criamos a classe Espada que e uma subclasse de ArmaPaladino, herdando
    os atributos de ArmaPaladino.
 *  Essa classe e responsavel por criar uma Arma de Paladino do tipo Espada
    .
 *  Dentro dessa classe, temos o construtor da classe Espada que recebe o
    nome e o dano da arma.
 *  Utilizamos o construtor da superclasse para inicializar os atributos de
     ArmaPaladino.
```

```
*/
public class Espada extends ArmaPaladino {
    public Espada() {
        super("Espada", 0.3);
    }
}
```

3.12. Lança.java

Este arquivo é responsável por criar a classe Lança, que é uma subclasse de ArmaPaladino. Dentro dessa classe, temos o construtor da classe Lança, que recebe os atributos da arma, como o nome e o modificador de dano. O nome da arma é "Lança"e o modificador de dano é 0.5. O construtor da superclasse ArmaPaladino é utilizado para inicializar esses atributos.

Código completo, com comentários:

Listing 12: Classe Lanca, subclasse de ArmaPaladino

```
/*
 *  Criamos a classe Lanca que e uma subclasse de ArmaPaladino, herdando os
     atributos de ArmaPaladino.
 *  Essa classe e responsavel por criar uma Arma de Paladino do tipo Lanca.
 *  Dentro dessa classe, temos o construtor da classe Lanca que recebe o
    nome e o dano da arma.
 *  Utilizamos o construtor da superclasse para inicializar os atributos de
     ArmaPaladino.
*/
public class Lanca extends ArmaPaladino {
    public Lanca() {
        super("Lanca", 0.5);
    }
}
```

3.13. Martelo.java

Este arquivo é responsável por criar a classe Martelo, que é uma subclasse de ArmaClerigo. Dentro dessa classe, temos o construtor da classe Martelo, que recebe os atributos da arma, como o nome e o modificador de dano. O nome da arma é "Martelo"e o modificador de dano é 0.6. O construtor da superclasse ArmaClerigo é utilizado para inicializar esses atributos.

Código completo, com comentários:

Listing 13: Classe Martelo, subclasse de ArmaClerigo

```
/*
 *  Criamos a classe Martelo que e uma subclasse de ArmaClerigo, herdando
    os atributos de ArmaClerigo.
 *  Essa classe e responsavel por criar uma Arma de Clerigo do tipo Martelo
    .
 *  Dentro dessa classe, temos o construtor da classe Martelo que recebe o
    nome e o dano da arma.
 *  Utilizamos o construtor da superclasse para inicializar os atributos de
     ArmaClerigo.
*/
public class Martelo extends ArmaClerigo {
    public Martelo(){
        super("Martelo", 0.6);
    }
}
```

3.14. Maça.java

Este arquivo é responsável por criar a classe Maca, que é uma subclasse de ArmaClerigo. Dentro dessa classe, temos o construtor da classe Maca, que recebe os atributos da arma, como o nome e o modificador de dano. O nome da arma é "Maça"e o modificador de dano é 0.4. O construtor da superclasse ArmaClerigo é utilizado para inicializar esses atributos.

Código completo, com comentários:

Listing 14: Classe Maça, subclasse de ArmaClerigo

```
/*
 *  Criamos a classe Maca que e uma subclasse de ArmaClerigo, herdando os
    atributos de ArmaClerigo.
 *  Essa classe e responsavel por criar uma Arma de Clerigo do tipo Maca.
 *  Dentro dessa classe, temos o construtor da classe Maca que recebe o
    nome e o dano da arma.
 *  Utilizamos o construtor da superclasse para inicializar os atributos de
     ArmaClerigo.
*/
public class Maca extends ArmaClerigo {
    public Maca(){
        super("Maca", 0.4);
    }
}
```

3.15. Main.java

A classe Main é a classe principal do programa, onde o fluxo do programa é executado. No método main, inicialmente, diante de um problema, defini que o padrão de formatação de números decimais seja o americano, já que o padrão brasileiro utiliza vírgulas como separadores decimais, enquanto o americano utiliza pontos.

Em seguida, é criado um vetor de personagens com tamanho 2, pois o RPG será jogado com dois personagens: o atacante e o defensor. Dentro de um laço de repetição, iremos percorrer o vetor de personagens, e lemos o número do personagem e, em seguida, seus atributos. De acordo com os números inseridos para o personagem e a arma, primeiro a arma é instanciada e, em seguida, o personagem.

Listing 15: Instanciação da Arma e do Personagem

```
case 1:
    ArmaMago armaMago;
    switch (numeroArma) {
        case 1:
            armaMago = new Transmutacao(); // A arma do mago e instanciada
                aqui com base no numero fornecido.
            break;
        case 2:
            armaMago = new Psikappa(); // Outra opcao de instancia para a
                arma do mago.
            break;
        default:
            throw new IllegalArgumentException("Arma inv lida para Mago.")
                ;
```

```
12      }
13      personagens[i] = new Mago(numeroSaude, numeroForca, numeroDestreza,
            armaMago); // O personagem Mago    instanciado aqui com os atributos
             e a arma escolhida.
14      break;
```

Após essa etapa, a batalha é iniciada. A batalha só é interrompida quando o número do personagem atacante ou do defensor for 0. Dentro do laço de repetição, são lidos os números dos personagens atacante e defensor, e então é realizado o ataque do personagem atacante contra o personagem defensor.

Listing 16: While da Batalha

```
1  while (true) {
2      // Le o numero do personagem atacante.
3      int ataque = sc.nextInt();
4      // Le o numero do personagem defensor.
5      int defesa = sc.nextInt();
6
7      // Caso o numero do personagem atacante ou defensor seja 0, o la o e
           interrompido.
8      if (ataque == 0 || defesa == 0) {
9          break;
10     }
11
12     /*
13      * Dentro de [] utilizamos o -1, pois o vetor comeca em 0
14      * e o numero do personagem come a em 1.
15     */
16     // Realizamos o ataque do personagem atacante no personagem defensor.
17     personagens[ataque - 1].atacar(personagens[defesa - 1]);
18     // Imprimimos o status do atacante.
19     personagens[ataque - 1].printStatus();
20     // Imprimimos o status do defensor.
21     personagens[defesa - 1].printStatus();
22 }
```

Código completo, com comentários:

Listing 17: Codigo Principal do Programa

```
1  /*
2   * Importamos as classes necessarias, sendo elas: Locale e Scanner.
3   * Locale: classe que permite a formatacao de numeros decimais, o padrao do
        Brasileiro e diferente do padrao Americano. O padrao brasileiro
       utiliza virgula como separador de casas decimais, enquanto o padrao
       americano utiliza ponto.
4   * Scanner: classe que permite a leitura de dados.
5   */
6  import java.util.Locale;
7  import java.util.Scanner;
```

```
8
9  // Classe Main que contem o metodo main, que e o metodo principal por
       executar o programa.
10 public class Main {
11     // Metodo main que e o metodo principal por executar o programa.
12     public static void main(String[] args) {
13         // Define o padrao de formatacao de numeros decimais para o padrao
               americano.
14         Locale.setDefault(Locale.US);
15
16         // Cria um objeto sc da classe Scanner que permite a leitura de
               dados.
17         Scanner sc = new Scanner(System.in);
18         // Cria um vetor de personagens com tamanho 2, ja que nosso RPG
               sera de dois personagens.
19         Personagem[] personagens = new Personagem[2];
20
21         // Laco de repeticao para ler os dados dos personagens.
22         for (int i = 0; i < 2; i++) {
23             // Le o numero do personagem (1: Mago, 2: Paladino, 3: Clerigo)
                   .
24             int numeroPersonagem = sc.nextInt();
25             // Le os numeros de saude, forca, destreza.
26             double numeroSaude = sc.nextDouble();
27             double numeroForca = sc.nextDouble();
28             double numeroDestreza = sc.nextDouble();
29             // Le o numero da arma (Cada arma tem numero 1 e 2, mas varia
                   entre os personagens).
30             int numeroArma = sc.nextInt();
31
32             /*
33              * Verificamos o numero do personagem. De acordo com esse
                    numero, ele entrara em um dos casos. Caso o numero seja
                    diferente de 1, 2 ou 3 uma excecao sera lancada.
34              * Todos os casos tem a mesma estrutura, mudando apenas a arma
                    de acordo com o personagem. Segue a estrutura:
35              * - De acordo com o numero do personagem, criamos inicialmente
                     uma variavel do tipo da arma do personagem.
36              * - Verificamos o numero da arma. De acordo com esse numero,
                    instanciamos a arma do personagem. Caso o numero seja
                    diferente de 1 ou 2, uma excecao sera lancada.
37              * - Instanciamos o personagem, de acordo com o numero do
                    personagem, passando os valores de saude, forca, destreza e
                     a arma.
38              * - Apos instanciarmos o personagem, ele e adicionado no vetor
                     de personagens.
39             */
40             switch (numeroPersonagem) {
41                 case 1:
42                     ArmaMago armaMago;
43                     switch (numeroArma) {
44                         case 1:
45                             armaMago = new Transmutacao();
```

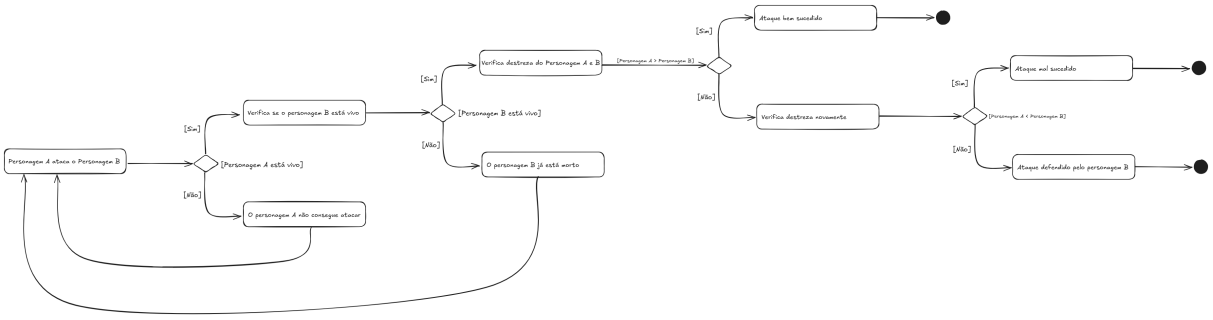
```
46                          break;
47                      case 2:
48                          armaMago = new Psikappa();
49                          break;
50                      default:
51                          throw new IllegalArgumentException("Arma
                                invalida para Mago.");
52                  }
53                  personagens[i] = new Mago(numeroSaude, numeroForca,
                        numeroDestreza, armaMago);
54                  break;
55
56              case 2:
57                  ArmaPaladino armaPaladino;
58                  switch (numeroArma) {
59                      case 1:
60                          armaPaladino = new Espada();
61                          break;
62                      case 2:
63                          armaPaladino = new Lanca();
64                          break;
65                      default:
66                          throw new IllegalArgumentException("Arma
                                invalida para Paladino.");
67                  }
68                  personagens[i] = new Paladino(numeroSaude, numeroForca,
                         numeroDestreza, armaPaladino);
69                  break;
70
71              case 3:
72                  ArmaClerigo armaClerigo;
73                  switch (numeroArma) {
74                      case 1:
75                          armaClerigo = new Martelo();
76                          break;
77                      case 2:
78                          armaClerigo = new Maca();
79                          break;
80                      default:
81                          throw new IllegalArgumentException("Arma
                                invalida para Clerigo.");
82                  }
83                  personagens[i] = new Clerigo(numeroSaude, numeroForca,
                        numeroDestreza, armaClerigo);
84                  break;
85
86              default:
87                  throw new IllegalArgumentException("Personagem invalido
                        .");
88          }
89      }
90
91      // Imprimimos o status inicial dos personagens.
```

```
92          personagens[0].printStatus();
93          personagens[1].printStatus();
94
95          // Laco de repeticao para ler os ataques e defesas dos personagens.
96          while (true) {
97              // Le o numero do personagem atacante.
98              int ataque = sc.nextInt();
99              // Le o numero do personagem defensor.
100             int defesa = sc.nextInt();
101
102             // Caso o numero do personagem atacante ou defensor seja 0, o
                    laco e interrompido.
103             if (ataque == 0 || defesa == 0) {
104                 break;
105             }
106
107             /*
108              * Dentro de [] utilizando o -1, pois o vetor comeca em 0 e o
                     numero do personagem comeca em 1.
109             */
110             // Realizamos o ataque do personagem atacante no personagem
                    defensor.
111             personagens[ataque - 1].atacar(personagens[defesa - 1]);
112             // Imprimimos o status do atacante.
113             personagens[ataque - 1].printStatus();
114             // Imprimimos o status do defensor.
115             personagens[defesa - 1].printStatus();
116         }
117         // Fecha o objeto sc da classe Scanner.
118         sc.close();
119     }
120 }
```

4. Dificuldades Encontradas

Não tive muita dificuldade em relação à estrutura em si. O Diagrama de Classes foi muito importante para entender como a estrutura do jogo funcionaria. Porém, algo que me levou um tempo considerável foi entender três aspectos.

O primeiro foi no entendimento de quando um personagem atacaria o outro. Mesmo estando descrito no trabalho, levei um tempo para organizar as ideias. O que me ajudou bastante foi o fluxograma que precisei desenhar, pois ele evitou que eu abstraísse demais as ideias das condicionais.



O segundo aspecto foi em relação às casas decimais, que estavam aparecendo com vírgula ao invés de ponto. Além disso, em um certo momento, as casas decimais estavam sendo exibidas de maneira errada, com muitos números após a vírgula.

Outro aspecto foi a demora para entender os status. Eu estava implementando incorretamente a parte responsável pela exibição da saída, o que resultava em algumas saídas invertidas, até notar que o status sempre é impresso inicialmente pelo status de quem está atacando.

Listing 18: Implementação Antes de Corrigir

```
personagens[0].printStatus();
personagens[1].printStatus();
// Mago [Morto, Forca: 10.0, Destreza: 5.0, Magia da Transmuta  o]
// Mago [Saude: 10.0, Forca: 10.0, Destreza: 6.0, Psi-kappa]
```

Listing 19: Implementação Após de Corrigir

```
personagens[ataque - 1].printStatus();
personagens[defesa - 1].printStatus();
// Mago [Saude: 10.0, Forca: 10.0, Destreza: 6.0, Psi-kappa]
// Mago [Morto, Forca: 10.0, Destreza: 5.0, Magia da Transmuta  o]
```

5. Soluções

Decidi agrupar as soluções, juntamente com os esclarecimentos e comentários, na seção anterior.