

Trabalho AP2 POO IBMEC

2025.2

Tema: RPG de Turno

Resumo: O jogo é um RPG de turnos onde cada personagem realiza uma ação por vez, podendo atacar, defender, usar habilidades ou itens. Cada personagem possui vida, ataque e defesa definidos pela classe escolhida. Quando um ataque acontece, o alvo recebe este dano a menos que esteja defendendo. Ao defender, o personagem ganha defesa temporária apenas para o próximo ataque. Habilidades podem causar dano, curar ou ter efeitos específicos conforme a classe. Jogadores possuem um inventário com itens que podem ser usados para recuperar vida durante o combate. Inimigos também atacam e defendem, seguindo regras semelhantes. O jogo avança alternando turnos até que um dos lados perca toda a vida, e se todos os seus personagens morrerem, game over.

Segue abaixo todas as exigências requisitadas

1- 6 classes concretas de domínio + [] 1 enum do domínio

Arqueiro.java

Guerreiro.java














Mago.java

Inimigo.java

Item.java

Jogador.java

TipoHabilidade.java (enum)

 Acoes.java
 Arqueiro.java
 Batalha.java
 Entidade.java
 Guerreiro.java
 Habilidade.java
 Inimigo.java
 Inventario.java
 Item.java
 Jogador.java
 Mago.java
 Personagem.java
 TipoHabilidade.java

2-1 hierarquia com classe abstrata (c/ métodos abstratos e concretos)

```
public abstract class Entidade {  no usages  👤 Pedro de Castro Kurtz
    protected String nome;  2 usages
    protected int vidaMaxima;  4 usages
    protected int vidaAtual;  9 usages
```

```

public Entidade(String nome, int vidaMaxima) { no usages  🧑 Pedro de Castro Kurtz
    this.nome = nome;
    this.vidaMaxima = vidaMaxima;
    this.vidaAtual = vidaMaxima;
}

public boolean estaViva() { return vidaAtual > 0; }

public void receberDano(int dano) { no usages  🧑 Pedro de Castro Kurtz
    vidaAtual -= dano;
    if (vidaAtual < 0) vidaAtual = 0;
}

public void curar(int qtd) { no usages  🧑 Pedro de Castro Kurtz
    vidaAtual += qtd;
    if (vidaAtual > vidaMaxima) vidaAtual = vidaMaxima;
}

public String getNome() { return nome; } no usages  🧑 Pedro de Castro Kurtz
public int getVidaAtual() { return vidaAtual; } no usages  🧑 Pedro de Castro Kurtz
public int getVidaMaxima() { return vidaMaxima; } no usages  🧑 Pedro de Castro Kurtz

public abstract void mostrarStatus(); no usages  🧑 Pedro de Castro Kurtz
}

```

3-Polimorfismo (sobrecarga + sobrescrita + chamadas polimórficas)

```

public class Mago extends Personagem { no usages  🧑 Pedro de Castro Kurtz
    public Mago(String nome) { super(nome, 80, 15, 5); }
}

```

A classe Personagem possui usarHabilidade:

```

public abstract void usarHabilidade(Habilidade h, Entidade alvo);

```

A classe Mago utiliza-se de usarHabilidade:

```

@Override no usages  🧑 Pedro de Castro Kurtz
public void usarHabilidade(Habilidade h, Entidade alvo) {
    if (h.getTipo() == TipoHabilidade.CURA) {
        curar(h.getPoder());
        System.out.println(nome + " se cura em " + h.getPoder() + " pontos!");
    } else {
        alvo.receberDano(h.getPoder());
        System.out.println(nome + " conjura " + h.getNome() + " e causa " + h.getPoder() + " de dano!");
    }
}
}

```

4-Cardinalidades: 1:1, 1:N, N:N

Cardinalidade 1:N

Jogador -> Personagem

```
private List<Personagem> personagens = new ArrayList<>();
```

```
public class Jogador { no usages  Pedro de Castro Kurtz
    private String nome; 4 usages
    private List<Personagem> personagens = new ArrayList<>(); 4 usages
    private Inventario inventario = new Inventario(); 1 usage
    private int nivel = 1; 4 usages
    private int xp = 0; 4 usages
```

Cardinalidade 1:1

Jogador -> Inventário

```
private Inventario inventario = new Inventario();
```

```
public class Jogador { no usages  Pedro de Castro Kurtz
    private String nome; 4 usages
    private List<Personagem> personagens = new ArrayList<>(); 4 usages
    private Inventario inventario = new Inventario(); 1 usage
    private int nivel = 1; 4 usages
    private int xp = 0; 4 usages
```

5- Pelo menos 1 bidirecional (com manutenção consistente dos dois lados)

6-Composição/Agregação quando fizer sentido

Composição: Jogador → Inventario.

O inventário é parte do jogador e é criado junto com ele.

```
public class Jogador { no usages  Pedro de Castro Kurtz
    private String nome; 4 usages
    private List<Personagem> personagens = new ArrayList<>(); 4 usages
    private Inventario inventario = new Inventario(); 1 usage
    private int nivel = 1; 4 usages
    private int xp = 0; 4 usages
```

Agregação: Jogador → Personagem.

Personagens podem existir sem o jogador

```
public abstract class Personagem extends Entidade implements Acoes { no usages  Pedro de Castro Kurtz
    protected int ataque; 1 usage
    protected int defesa; 1 usage
    protected List<Habilidade> habilidades = new ArrayList<>(); 3 usages
```

7- Interface própria implementada por ≥ 2 classes;

```
public interface Acoes {
    void atacar(Entidade alvo);
    void defender();
}
```

8-Uso de Collections com verificação de duplicidade:

(List)

protected List<Habilidade> habilidades = new ArrayList<>();

```
public abstract class Personagem extends Entidade implements Acoes {
    protected int ataque;
    protected int defesa;
    protected List<Habilidade> habilidades = new ArrayList<>();
    protected int defesaTemporaria = 0; // zera-se após absorver um d
```

9-Ordenação demonstrada

```
public void ordenarPersonagensPorNome() {
    personagens.sort(Comparator.comparing(Personagem::getNome));
}
```

10-App Interativo (menu com Scanner)

```
import java.util.Scanner;

public class App {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        Random rand = new Random();
        Jogador jogador = new Jogador("Herói");

        Guerreiro guerreiro = new Guerreiro("Ragnar");
        Mago mago = new Mago("Merlin");
        Arqueiro arqueiro = new Arqueiro("Robin");

        jogador.adicionarPersonagem(guerreiro);
        jogador.adicionarPersonagem(mago);
        jogador.adicionarPersonagem(arqueiro);

        System.out.println("=== Bem-vindo ao RPG: Aventura no Reino ===");
    }
}
```

11-App Roteiro (sem Scanner, saídas determinísticas)

```

System.out.println("=== Bem-vindo ao RPG: Aventura no Reino ===");

boolean jogando = true;
while (jogando) {
    System.out.println("\nEscolha uma ação:");
    System.out.println("1 - Explorar área");
    System.out.println("2 - Ver inventário");
    System.out.println("3 - Ver status");
    System.out.println("4 - Sair");
    int escolha = sc.nextInt();
    sc.nextLine();

    switch (escolha) {
        case 1 -> {
            int evento = rand.nextInt(3);
            if (evento == 0) {
                System.out.println("Você encontrou um inimigo!");
                Inimigo inimigo = new Inimigo("Goblin", 60 + rand.nextInt(40), 10);
                System.out.println("Com qual personagem deseja lutar?");
                for (int i = 0; i < jogador.getPersonagens().size(); i++) {
                    System.out.println((i + 1) + " - " + jogador.getPersonagens().get(i).getNome());
                }
                int idx = sc.nextInt() - 1;
                sc.nextLine();
                Personagem p = jogador.getPersonagens().get(idx);
                Batalha b = new Batalha(p, inimigo);
                b.iniciar();
            }
        }
    }
}

```

Segue README.md no arquivo .zip

Representantes:

Pedro de Castro Kurtz (202407278752)

Gabriel Maccachero Araujo (202501441823)

Victor Coutinho