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)

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
```

```java
    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)

```java
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:

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

A classe Mago utiliza-se de usarHabilidade:

```java
    @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 {  no usa
    void atacar(Entidade alvo);
    void defender();  no usages
}
```

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;  1 usage
    protected int defesa;  2 usages
    protected List<Habilidade> habilidades = new ArrayList<>();  3 usa
    protected int defesaTemporaria = 0; // zera-se após absorver um
```

9-Ordenação demonstrada

```
public void ordenarPersonagensPorNome() {  no usages  new *
personagens.sort(Comparator.comparing(Personagem::getNome));}
```

10-App Interativo (menu com Scanner)

```
import ...                                                                    OFF

public class App {  no usages  Pedro de Castro Kurtz
    public static void main(String[] args) {  no usages  Pedro de Castro Kurtz
        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();
```

12-PDF com prints evidenciando cada exigência

Ta aqui!

13-README.md com como compilar e executar ambos os apps.