📝 Programación Orientada a Objetos 2025

Proyecto: Juego de Batalla por Turnos

🎯 Objetivo

Poner en práctica lo visto en clases y comprender donde aplicamos cada concepto, ya sea, **constructores, herencia, métodos abstractos, sobrescritura de métodos**, entre otros, mediante la creación de un pequeño juego de batalla.

📌 Enunciado

En este proyecto deberás completar la clase abstracta `Personaje` y luego crear **tres subclases**: `Guerrero`, `Mago` y `Arquero`.

Cada personaje tendrá **ataque normal, defensa y una habilidad especial distinta**. Luego, estos personajes serán utilizados en un juego con interfaz gráfica ya implementada.

🔹 Parte 1: Clase `Personaje`

Se te entrega la clase incompleta y deberas completar lo siguiente:

1. Completar modificadores de atributos y Constructor de clase.
2. Completar **getters y setters** (`getNombre`, `getVida`, `setVida`, `getPoder`).

3. Completar el método **estaVivo()** → devuelve `true` si el personaje aún tiene `vida`.

4. Implementar **setDefensa()** → activa el estado de defensa (Hint: ver que tipo es el atributo `defendiendo`).

5. Implementar **recibirDanio(int danio)**:

    - Si el personaje está defendiendo, el daño recibido se divide entre 4.
    - Luego se resta a la vida.
    - Un personaje sin vida no puede seguir recibiendo daño.

🔹 Parte 2: Subclases

Debes crear 3 clases que extiendan de `Personaje` y sobrescriban los métodos abstractos:

1. `Guerrero`

- **Atributos:**
  - **Escudo: Representa la cantidad de escudo que tiene el guerrero**
- **Ataque normal:** devuelve `poder`.
- **Defensa:** Resta 20 de escudo y activa defensa.
- **Habilidad especial:** golpe crítico → devuelve el **doble del poder**.

2. `Arquero`

- **Atributos:**
  - **Flechas: Representa la cantidad de flechas que tiene el arquero**
- **Ataque normal:** Resta una flecha y devuelve `poder`.
- **Defensa:** activa defensa.
- **Habilidad especial:** flecha múltiple → Resta dos flechas ataca a **dos enemigos a la vez**.
  (por ahora devuelve el daño total, que será igual a el cuadruple del `poder`).

3. `Mago`

- **Atributos:**
  - **Maná: Representa la cantidad de mana que tiene el mago**
- **Ataque normal:** Resta 25 de maná y devuelve `poder`.
- **Defensa:** Aumenta 20 de maná y activa defensa.
- **Habilidad especial:** regeneración → Resta 75 de maná y en lugar de hacer daño, **recupera 50 puntos de vida**. (En vez de devolver el daño, devolvemos la curacion realizada).

🚀 Guía para ejecutar el juego en Java (VS Code + Terminal)

🔹 Paso 1: Abrir el proyecto

1. Abrir **Visual Studio Code**.

Ir a **File** → **Open Folder…** y seleccionar la carpeta del proyecto (la que contiene la carpeta `RPG` con los `.java`). Asegurarse de crear los archivos de las subclases dentro de: `src/`
Ejemplo:

```
RPG/
      ├── BatallaGUI.class
      ├── Main.class
      ├── src/
      |    ├── Personaje.java
      |    ├── Guerrero.java
      |    ├── Mago.java
      |    └──  Arquero.java
      |
      └── images/
            ├── Guerrero.png
            ├── Guerrero-enemigo.png
            └── …
```

🔹 Paso 2: Abrir la terminal en VS Code

- En VS Code, ir a **Terminal** → **New Terminal**.
- Se abrirá una consola en la carpeta raíz del proyecto.

🔹 Paso 3: Compilar el proyecto

En la terminal, escribir:

```
javac src/*.java
```

👉 Esto compila **todos los archivos** `.java` dentro de la carpeta `src\` y genera archivos `.class` (el bytecode que corre Java).

🔹 Paso 4: Ejecutar el juego

Una vez compilado, correr:

```
java Main
```

👉 Esto ejecuta la clase `Main`, que inicia el juego y abre la ventana con la interfaz gráfica.

🔹 Borrar archivos compilados

Si cambian el código y algo no anda, puede ser útil borrar los `.class` antes de recompilar.

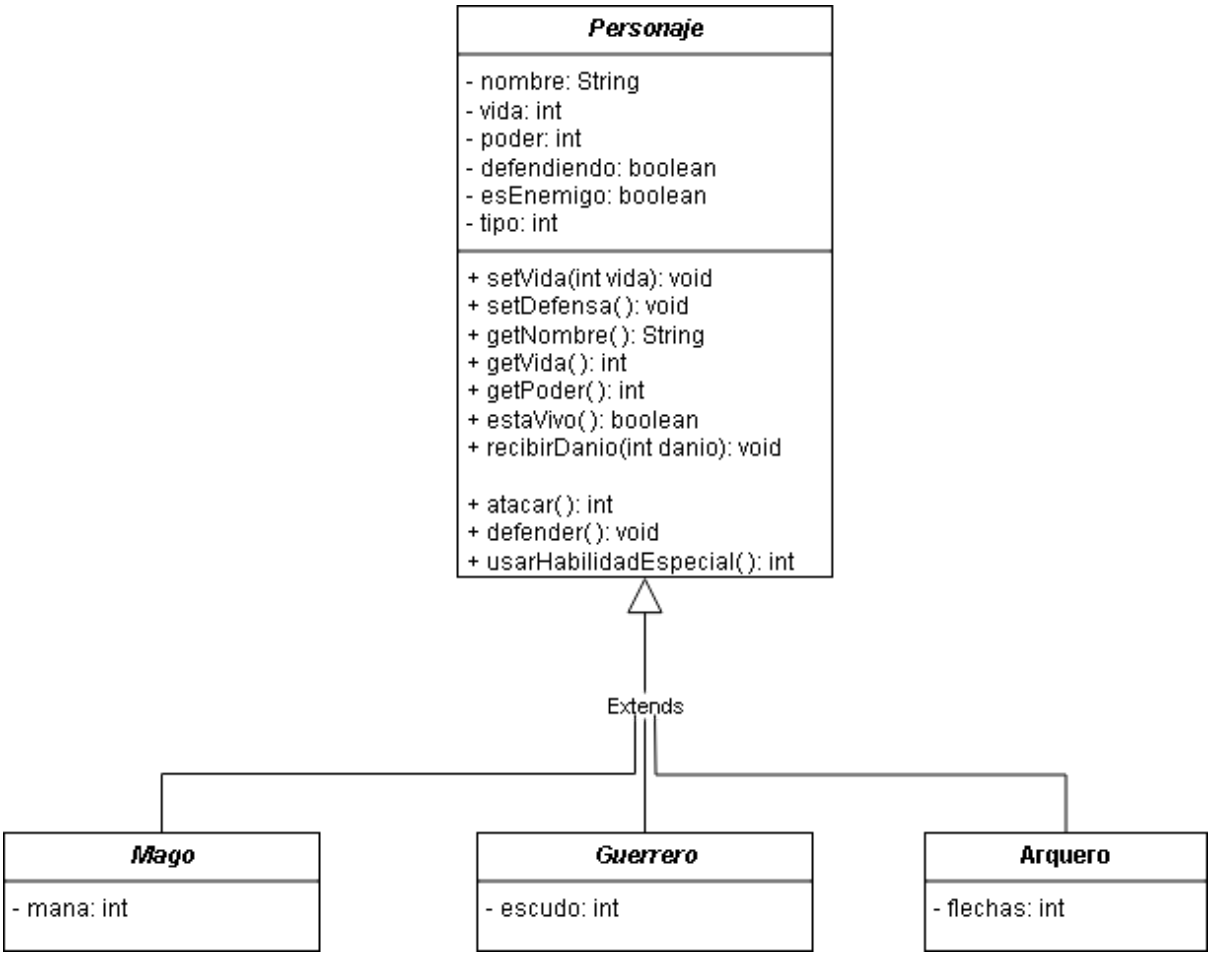
En Linux/Mac:

```
rm src/(NombreArchivo).class
```

En Windows (PowerShell):

```
del src\(nombreArchivo).class
```

🚀 Diagrama de Clases



📊 Evaluación

El trabajo será evaluado en base a los siguientes criterios:

1. **Estructura y calidad del código**
    - Uso correcto de **POO** (herencia, sobrescritura de métodos, encapsulamiento).
    - Organización del código, claridad en nombres de variables y métodos.
    - Cumplimiento de las consignas (atributos, constructores, getters/setters, implementación de métodos abstractos).
2. **Defensa oral**
    - Cada alumno deberá realizar una **defensa oral de 5 minutos** explicando:
        - Cómo implementaron cada clase y qué decisiones tomaron.
        - Qué dificultades encontraron y cómo las resolvieron.
        - Si usaron Inteligencias Artificiales; Para que se usaron y cómo se usaron.
