ESTA SOLUCIÓN DEBE SER ARMADA EN BLUEJ y **REALIZAR LOS AJUSTES CORRESPONDIENTES**

```
// Clase base abstracta para cualquier personaje del juego
// Es abstracta porque no crearemos instancias de un "Personaje" genérico,
// sino de tipos específicos como Guerrero o Mago.
public abstract class Personaje {
    // Atributos comunes a todos los personajes (encapsulados)
    private String nombre;
    private int nivel;
    private int puntosVida; // Salud actual del personaje
    private String estado; // Por ejemplo: "Normal", "Envenenado"

    // Constructor para inicializar un Personaje.
    // Es llamado por los constructores de las clases hijas usando 'super()'.
    public Personaje(String nombre, int nivelInicial, int vidaInicial) {
        this.nombre = nombre;
        this.nivel = nivelInicial;
        this.puntosVida = vidaInicial; // Asigna la vida inicial al crear el personaje
        this.estado = "Normal"; // Estado inicial por defecto
    }

    // --- Métodos Getters (para acceder a los atributos desde fuera) ---
    public String getNombre() {
        return nombre;
    }

    public int getNivel() {
        return nivel;
    }

    public int getPuntosVida() {
        return puntosVida;
    }

    public String getEstado() {
        return estado;
    }

    // --- Métodos Setters (para modificar algunos atributos desde fuera) ---
    public void setEstado(String estado) {
        this.estado = estado;
    }

    // --- Métodos de Lógica Interna (NO imprimen ni leen) ---
```

```
    // Método para que el personaje suba de nivel.
    // SOLO modifica el estado interno (nivel y vida).
    public void subirNivel() {
        this.nivel++;
        // Al subir de nivel, aumentamos la vida máxima (simulado aumentando la vida actual)
        this.puntosVida += 10;
        // NO se imprime nada aquí. El main se encargará de notificar que subió de nivel.
    }

    // Método para que el personaje reciba daño.
    // Calcula el daño efectivo, modifica la vida y DEVUELVE el daño que realmente se aplicó.
    public int recibirDaño(int cantidadDaño) {
        int dañoEfectivo = cantidadDaño;
        // Lógica simple de estado afectando el daño
        if (this.estado != null && this.estado.equals("Envenenado")) {
            dañoEfectivo += 5; // Daño extra por veneno
        }

        this.puntosVida -= dañoEfectivo;

        // Asegurarse de que la vida nunca sea negativa
        if (this.puntosVida < 0) {
            this.puntosVida = 0;
        }

        // Devolvemos cuánto daño se aplicó realmente para que el main lo pueda mostrar
        return dañoEfectivo;
    }

    // Método estándar de Java para obtener una representación en String del objeto.
    // Es usado por el main para obtener la información del estado del personaje.
    // NO imprime, SOLO devuelve un String.
    @Override
    public String toString() {
        return "[Tipo: " + this.getClass().getSimpleName() +
               ", Nombre: " + nombre +
               ", Nivel: " + nivel +
               ", Vida: " + puntosVida +
               ", Estado: " + estado + "]";
    }

    // Método ABSTRACTO: La habilidad especial de cada personaje.
    // Cada subclase DEBE implementar este método.
    // Ahora DEVUELVE el valor numérico (daño potencial, curación, etc.) que resulta de la
habilidad.
    // NO imprime ni realiza acciones complejas sobre otros objetos aquí.
    public abstract double realizarAtaqueEspecial(); // Devuelve el resultado numérico de la
habilidad
```

```
}
```

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

```
// Clase concreta que representa a un Guerrero
// Hereda de Personaje y añade atributos y lógica específicos de Guerrero.
public class Guerrero extends Personaje {
    // Atributo específico de Guerrero
    private int fuerza;

    // Constructor para crear un Guerrero
    public Guerrero(String nombre, int nivelInicial, int vidaInicial, int fuerzaInicial) {
        // Llama al constructor de la clase padre (Personaje) para inicializar los atributos
heredados
        super(nombre, nivelInicial, vidaInicial);
        this.fuerza = fuerzaInicial; // Inicializa el atributo propio de Guerrero
    }

    // Getter específico para fuerza
    public int getFuerza() {
        return fuerza;
    }

    // Setter específico para fuerza
    public void setFuerza(int fuerza) {
        this.fuerza = fuerza;
    }

    // Implementación OBLIGATORIA del método abstracto realizarAtaqueEspecial().
    // Calcula el daño potencial del ataque del Guerrero y lo DEVUELVE.
    // NO imprime ni realiza acciones sobre otros objetos aquí.
    @Override
    public double realizarAtaqueEspecial() {
        // Calcula el daño potencial basado en la fuerza y lo devuelve.
        // Usamos 2.0 para asegurar que el resultado sea un double.
        return this.fuerza * 2.0;
    }

    // Sobrescribe el método toString() para incluir la información específica de Guerrero.
    // Llama al toString de la superclase y añade la fuerza.
    @Override
    public String toString() {
        // Obtiene la representación base y le añade la fuerza antes del corchete final
        return super.toString().replace("]", ", Fuerza: " + fuerza + "]");
    }
}
```

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

```
// Clase concreta que representa a un Mago
// Hereda de Personaje y añade atributos y lógica específicos de Mago.
public class Mago extends Personaje {
    // Atributo específico de Mago
    private int poderMagico;

    // Constructor para crear un Mago
    public Mago(String nombre, int nivelInicial, int vidaInicial, int poderMagicoInicial) {
        // Llama al constructor de la clase padre (Personaje)
        super(nombre, nivelInicial, vidaInicial);
        this.poderMagico = poderMagicoInicial; // Inicializa el atributo propio de Mago
    }

    // Getter específico para poderMagico
    public int getPoderMagico() {
        return poderMagico;
    }

    // Setter específico para poderMagico
    public void setPoderMagico(int poderMagico) {
        this.poderMagico = poderMagico;
    }

    // Implementación OBLIGATORIA del método abstracto realizarAtaqueEspecial().
    // Calcula el daño potencial del hechizo del Mago y lo DEVUELVE.
    // NO imprime ni realiza acciones sobre otros objetos aquí.
    @Override
    public double realizarAtaqueEspecial() {
        // Calcula el daño potencial basado en el poder mágico y lo devuelve
        return this.poderMagico * 1.5;
    }

    // Sobrescribe el método toString() para incluir la información específica de Mago.
    // Llama al toString de la superclase y añade el poder mágico.
    @Override
    public String toString() {
         // Obtiene la representación base y le añade el poder mágico antes del corchete final
        return super.toString().replace("]", ", Poder Mágico: " + poderMagico + "]");
    }
}
```

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

```
// Clase principal con el método main para simular las interacciones.
// Esta clase es la ÚNICA responsable de mostrar información en la consola.
public class SimuladorJuego {

    // El método main es el punto de entrada del programa.
    public static void main(String[] args) {

        // --- Demostración de Instanciación y Uso de Array Polimórfico ---

        // Definimos el tamaño del array (en este caso, 2 personajes)
        int tamañoGrupo = 2;
        // Creamos un array de tipo Personaje (la superclase).
        // Este array puede almacenar objetos de CUALQUIER clase que herede de Personaje.
        Personaje[] grupoAventureros = new Personaje[tamañoGrupo];

        // Creamos instancias de las clases concretas (Guerrero y Mago).
        Guerrero conan = new Guerrero("Conan", 5, 150, 20);
        Mago merlin = new Mago("Merlin", 6, 80, 25);

        // Añadimos los objetos de las subclases al array en posiciones específicas.
        // Esto demuestra que una referencia de superclase puede apuntar a un objeto de
subclase.
        grupoAventureros[0] = conan; // Posición 0 guarda el objeto Guerrero
        grupoAventureros[1] = merlin; // Posición 1 guarda el objeto Mago

        System.out.println("--- Estado inicial del grupo de aventureros ---");
        // Iteramos sobre el array. Cada 'p' es una referencia de tipo Personaje.
        // El bucle for mejorado funciona con arrays.
        for (Personaje p : grupoAventureros) {
            // Usamos el método toString() del objeto para obtener su descripción
            // y luego la imprimimos en la consola desde el main.
            System.out.println(p.toString());
        }

        System.out.println("\n--- Los aventureros realizan sus ataques especiales ---");
        for (Personaje p : grupoAventureros) {
            // Llamamos al método polimórfico realizarAtaqueEspecial().
            // Este método calcula y DEVUELVE el daño potencial, NO lo imprime.
            double dañoPotencial = p.realizarAtaqueEspecial();

            // En el main, usamos el valor devuelto para mostrar la información.
            System.out.println(p.getNombre() + " realiza su ataque especial. Daño potencial
calculado: " + dañoPotencial);
```

```
        }

        System.out.println("\n--- Simulación de recibir daño para todos ---");
         for (Personaje p : grupoAventureros) {
            // Simulamos que reciben 20 de daño base
            int dañoBaseSimulado = 20;

            // Llamamos al método recibirDaño().
            // Este método modifica la vida interna y DEVUELVE el daño efectivo recibido.
            int dañoEfectivoRecibido = p.recibirDaño(dañoBaseSimulado);

            // En el main, usamos el valor devuelto y el estado actual del objeto
            // para mostrar la información relevante.
            System.out.println(p.getNombre() + " recibió " + dañoEfectivoRecibido + " puntos de
daño efectivo.");
            System.out.println("Vida restante de " + p.getNombre() + ": " + p.getPuntosVida());

            // En main, verificamos si el personaje fue derrotado y lo informamos.
            if (p.getPuntosVida() == 0) {
                System.out.println(p.getNombre() + " ha sido derrotado.");
            }
         }

        System.out.println("\n--- Simulacion de subir de nivel ---");
         for (Personaje p : grupoAventureros) {
            // Llamamos al método subirNivel().
            // Este método SOLO modifica el estado interno (nivel y vida).
            p.subirNivel();

            // Luego, en main, mostramos que subió de nivel y su nuevo estado usando los
getters y toString().
            System.out.println(p.getNombre() + " ha subido al nivel " + p.getNivel() + "!");
            System.out.println("Nuevo estado de " + p.getNombre() + ": " + p.toString());
         }


        System.out.println("\n--- Estado final del grupo ---");
        for (Personaje p : grupoAventureros) {
           System.out.println(p.toString());
        }
    }
}
```