Universidad San Carlos de Guatemala

Introduccion a la Computación y Programacion

PRACTICA 2

Nombre: Pablo Gabriel Ordoñez Escobar
Carné: 202407178
Seccion: “B”
Fecha: 26/09/2025

Descripción del programa:

ArenaUSAC es una aplicación Java de escritorio que simula un sistema de batallas entre personajes. La aplicación permite gestionar personajes con atributos específicos, simular batallas en tiempo real usando hilos, y mantener un historial de combates.

Requerimientos del Sistema

- Lenguaje: Java 8 o superior
- Interfaz: Swing/AWT
- Persistencia: Archivos de texto plano (.dat)
- Concurrencia: Múltiples hilos para batallas simultáneas
- Sistema Operativo: Multiplataforma (Windows, Linux, macOS)

Arquitectura de la Aplicación

Descripción Detallada de Clases y Métodos

1. Clase Personaje

Funcion: Representa un personaje del juego con todos sus atributos y comportamientos.

- **Atributos:**

```
//Atributos del personaje
private int id;
private String nombre;
private String arma;
private int hp;
private int ataque;
private int velocidad;
private int agilidad;
private int defensa;
private int batallasGanadas;
private int batallasPerdidas;
```

- **Métodos Principales**
  - Método para calcular el daño recibido

```
//Metodo para calcular dano recibido
public int recibirDano(int danoBase){
    int danoReal = danoBase - this.defensa;
    if(danoReal < 0) danoReal = 0;
    this.hp -= danoReal;
    if(this.hp < 0) this.hp = 0;
    return danoReal;
}
```

  - Método para verificar si está vivo

```
//Método para verificar si esta vivo
public boolean estaVivo(){
    return this.hp > 0;
}
```

- Método para clonar personajes

```java
//Método para clonar personaje (útil para batallas)
public Personaje clonar(){
    Personaje clon = new Personaje(this.id, this.nombre, this.arma, this.hp, this.ataque,
            this.velocidad, this.agilidad, this.defensa);
    clon.setBatallasGanadas(this.batallasGanadas);
    clon.setBatallasPerdidas(this.batallasPerdidas);
    return clon;
}
```

- Método para incrementar estadísticas de batallas

```java
//Método para incrementar estadisticas de batallas
public void incrementarBatallasGanadas(){
    this.batallasGanadas++;
}

public void incrementarBatallasPerdidas(){
    this.batallasPerdidas++;
}
```

2. Clase GestorPersonajes

Funcion: Gestiona la colección de personajes y valida las operaciones CRUD.

- **Métodos Principales**
  - Método para agregar personajes

```java
public boolean agregarPersonaje(String nombre, String arma, int hp, int ataque, int velocidad, int agilidad, int defensa) {
    if(!validarRangos(hp, ataque, velocidad, agilidad, defensa)){
        return false;
    }
    if(buscarPorNombre(nombre) != null){
        return false;
    }
    if(cantidad >= personajes.length){
        return false;
    }
    Personaje nuevo = new Personaje(siguienteId++, nombre, arma, hp, ataque, velocidad, agilidad, defensa);
    personajes[cantidad] = nuevo;
    cantidad++;
    return true;
}
```

  - Método que valida que los atributos estén dentro de rangos permitidos

```java
private boolean validarRangos(int hp, int ataque, int velocidad, int agilidad, int defensa){
    return (hp >= 100 && hp <= 500) &&
           (ataque >= 10 && ataque <= 100) &&
           (velocidad >= 1 && velocidad <= 10) &&
           (agilidad >= 1 && agilidad <= 10) &&
           (defensa >= 1 && defensa <= 50);
}
```

- Método que busca personaje por nombre

```
public Personaje buscarPorNombre(String nombre){
    for (int i = 0; i < cantidad; i++) {
        if (personajes[i].getNombre().equalsIgnoreCase(nombre)) {
            return personajes[i];
        }
    }
    return null;
}
```

3. Clase Batalla

Función: Controla la simulación de batallas entre dos personajes usando hilos.

- **Métodos Principales**
  - Método para iniciar la simulación de batalla

```
public void iniciar(){
```

  - Método para lógica de ataque en un hilo separado

```
private void atacar(Personaje atacante, Personaje defensor){
```

4. Clase GestorArchivos

Función: Maneja la persistencia de datos en archivos de texto.

- **Métodos Principales**
  - Metodo para guardar todos los personajes en archivo

```
public static void guardarPersonajes(GestorPersonajes gestor, String archivo) {
    try (PrintWriter writer = new PrintWriter(new FileWriter(archivo))) {
        Personaje[] personajes = gestor.getPersonajes();
        int cantidad = gestor.getCantidad();

        for (int i = 0; i < cantidad; i++) {
            Personaje p = personajes[i];
            writer.println(p.getId() + "," + p.getNombre() + "," + p.getArma() + "," +
                           p.getHp() + "," + p.getAtaque() + "," + p.getVelocidad() + "," +
                           p.getAgilidad() + "," + p.getDefensa() + "," +
                           p.getBatallasGanadas() + "," + p.getBatallasPerdidas());
        }
    }catch (IOException e) {
        System.out.println("Error guardando personajes: " + e.getMessage());
    }
}
```

  - Método para cargar personajes desde archivo

```
public static void cargarPersonajes(GestorPersonajes gestor, String archivo) {
```

5. Clase Main

Función: Interfaz gráfica principal que coordina todas las funcionalidades.

- **Métodos de Interfaz**
  - Método de router de eventos para botones de la interfaz

```
private void manejarBoton(String opcion){
```

- **Métodos de Gestión**
  - ***agregarPersonaje():*** Diálogos para entrada de datos
  - ***simularBatalla():*** Configura y lanza batalla en hilo separado
  - ***verHistorial():*** Muestra estadísticas acumuladas