Manual Técnico – ArenaUSAC (Práctica 2)

1. Requerimientos de la aplicación

Requerimientos funcionales

- Gestión de personajes: agregar, editar, eliminar y visualizar personajes.
- Validación de datos: asegurar que los atributos de los personajes (HP, ataque, velocidad, defensa, agilidad) estén dentro de rangos válidos.
- Simulación de batallas:
    - Cada personaje ataca desde un hilo independiente.
    - La velocidad determina la frecuencia de ataque.
    - La agilidad define la probabilidad de esquivar.
    - La defensa reduce el daño recibido.
    - La batalla termina automáticamente cuando un personaje llega a HP ≤ 0.
- Historial de batallas: registrar fecha, participantes, bitácora de turnos y resultado.
- Persistencia: guardar y cargar el estado del sistema (personajes e historial) en un archivo de texto plano.
- Interfaz gráfica: debe permitir al usuario interactuar con menús, formularios y tablas de personajes.

Requerimientos no funcionales

- Lenguaje: Java SE 17 o superior.
- Librerías: Swing (javax.swing), sin dependencias externas.
- IDE recomendado: NetBeans o IntelliJ.
- El código debe estar documentado y organizado de forma modular.

- Los datos deben almacenarse en estructuras de listas (ArrayList).
- Concurrencia implementada con hilos y SwingWorker.

---

2. Estructura del sistema

Organización de clases

- **personaje**: representa a un combatiente con atributos como nombre, arma, HP, ataque, defensa, velocidad, agilidad, victorias y derrotas.
- **batalla**: representa un enfrentamiento entre dos personajes, guarda fecha, participantes, ganador y bitácora de eventos.
- **historial**: estructura que almacena todas las batallas realizadas.
- **trabajadorBatalla**: hilo encargado de ejecutar la simulación de una batalla entre dos personajes.
- **persistencia**: módulo para guardar y cargar datos desde archivo de texto.
- **main**: clase principal con interfaz gráfica (Swing), contiene la lógica de interacción con el usuario.

3. Descripción de clases y métodos principales

Clase Personaje

```java
package model;

import java.util.concurrent.atomic.AtomicInteger;

public class personaje {
    private static final AtomicInteger ID_GEN = new AtomicInteger(1);

    private final int id;
    private String nombre;
    private String arma;
    private int hp;
    private int ataque;
    private int velocidad;
    private int agilidad;
    private int defensa;

    // estadísticas
    private int wins = 0;
    private int losses = 0;

    public personaje(String nombre, String arma, int hp, int ataque, int velocidad, int agilidad, int defensa) {
        this.id = ID_GEN.getAndIncrement();
        this.nombre = nombre.trim();
        this.arma = arma.trim();
        this.hp = hp;
        this.ataque = ataque;
        this.velocidad = velocidad;
        this.agilidad = agilidad;
        this.defensa = defensa;
    }
```

- **Atributos**: id, nombre, arma, hp, ataque, velocidad, agilidad, defensa, victorias, derrotas.
- **Métodos**:
    - `estaVivo()`: devuelve true si el personaje tiene HP > 0.
    - `recibirDano(int rawDamage)`: aplica defensa y reduce los puntos de vida.
    - `addWin() / addLoss()`: actualizan las estadísticas de combates.
    - `toPersistString()`: exporta los datos a texto para guardado.

Clase Batalla

- **Atributos**: id de la batalla, fecha de inicio, personaje A, personaje B, ganador, lista de eventos.

- **Métodos**:

    - `addLog(String s)`: agrega un evento a la bitácora.

    - `toPersistHeader()`: genera el encabezado de la batalla para guardado en archivo.

    - `toPersistLog(int battleId)`: guarda la bitácora de eventos.

Clase Historial

```java
package model;

import java.util.ArrayList;
import java.util.List;

public class historial {
    private final List<batalla> batallas = new ArrayList<>();

    public synchronized void add(batalla b) { batallas.add(b); }
    public synchronized List<batalla> getAll() { return new ArrayList<>(batallas); }
}
```

- **Métodos**:

    - `add(Batalla b)`: agrega una batalla al historial.

    - `getAll()`: devuelve la lista de todas las batallas.

Clase trabajadorBatalla

```java
package trabajadores;

import model.batalla;
import model.personaje;

import javax.swing.*;
import java.util.Random;

public class trabajadorBatalla extends SwingWorker<Void, String> {
    private final personaje p1;
    private final personaje p2;
    private final batalla batalla;
    private volatile boolean running = true;
    private final Random rand = new Random();

    public trabajadorBatalla(personaje p1, personaje p2, batalla b) {
        this.p1 = p1;
        this.p2 = p2;
        this.batalla = b;
    }

    @Override
    protected Void doInBackground() throws Exception {
        // cada personaje ataca en su propio "ritmo" basado en velocidad.
        Thread t1 = new Thread(() -> ataqueLoop(p1, p2));
        Thread t2 = new Thread(() -> ataqueLoop(p2, p1));
        t1.start();
        t2.start();

        // esperar a que uno muera
        while (running && p1.estaVivo() && p2.estaVivo()) {
```

- **Método principal**:
    - `doInBackground()`: corre la simulación de la batalla en hilos separados para cada combatiente.
    - `loopAttack(Personaje atacante, Personaje defensor)`: lógica de ataque con probabilidad de esquiva y cálculo de daño.

Clase Persistencia

```java
package util;

import model.batalla;
import model.historial;
import model.personaje;

import java.io.*;
import java.nio.file.Files;
import java.nio.file.Path;
import java.time.format.DateTimeFormatter;
import java.util.*;

public class persistencia {

    // Guarda personajes y batallas en formato simple
    public static void save(String path, List<personaje> personajes, historial historial) throws IOException {
        try (BufferedWriter bw = Files.newBufferedWriter(Path.of(path))) {
            for (personaje p : personajes) {
                // PERSON|id|nombre|arma|hp|ataque|velocidad|agilidad|defensa|wins|losses
                bw.write(String.format("PERSON|%d|%s|%s|%d|%d|%d|%d|%d|%d\n",
                        p.getId(),
                        escape(p.getNombre()),
                        escape(p.getArma()),
                        p.getHp(), p.getAtaque(), p.getVelocidad(), p.getAgilidad(), p.getDefensa(),
                        p.getWins(), p.getLosses()
                ));
            }
            for (batalla b : historial.getAll()) {
                bw.write(b.toPersistHeader() + "\n");
                bw.write(b.toPersistLog(b.getId()) + "\n");
            }
```

- **Métodos**:
    - `save(String path, List<Personaje>, Historial)`: guarda todos los datos en un archivo.
    - `load(String path)`: carga personajes y batallas desde archivo.

Clase main

```
package iu;

import model.batalla;
import model.historial;
import model.personaje;
import trabajadores.trabajadorBatalla;
import util.persistencia;
import javax.swing.Timer;
import javax.swing.*;
import javax.swing.event.ListSelectionEvent;
import javax.swing.table.DefaultTableModel;
import java.awt.*;
import java.awt.event.*;
import java.io.File;
import java.util.*;
import java.util.List;
import java.util.concurrent.ExecutionException;

public class main extends JFrame {
    private final DefaultTableModel tableModel = new DefaultTableModel(new String[]{"ID","Nombre","Arma","HP","ATK","DEF","AGI","SPD"},0) {
        public boolean isCellEditable(int r, int c){return false;}
    };
    private final JTable table = new JTable(tableModel);
    private final JTextArea logArea = new JTextArea(15, 50);
    private final JTextArea battleLog = new JTextArea(10,50);
    private final java.util.List<personaje> personajes = Collections.synchronizedList(new ArrayList<>());
    private final historial historial = new historial();
    private trabajadorBatalla currentWorker = null;
```

- **Métodos de la interfaz gráfica**:
    - `onAdd()`: agrega un nuevo personaje validando entradas.
    - `onEdit()`: permite modificar un personaje seleccionado.
    - `onDelete()`: elimina un personaje tras confirmación.
    - `onStartBattle()`: inicializa un combate con dos personajes seleccionados.
    - `onStopBattle()`: detiene una batalla en curso.
    - `refreshTable()`: actualiza la tabla de personajes en pantalla.
    - `refreshCombos()`: actualiza las listas desplegables de combatientes.

4. Requerimientos de hardware y software

- **Hardware mínimo**: procesador de dos núcleos, 4 GB de RAM.
- **Software necesario**:
    - Java JDK 17 o superior.
    - NetBeans, IntelliJ IDEA o Eclipse.
    - Sistema operativo Windows, Linux o macOS.

5. Posibles mejoras técnicas

- Migrar el sistema de persistencia a formato JSON para mayor flexibilidad.
- Sustituir la interfaz Swing por JavaFX para una experiencia más moderna.
- Implementar patrón de diseño MVC para separar lógica, vista y controlador.
- Agregar pruebas unitarias con JUnit.

6. Conclusiones técnicas

- La aplicación implementa de manera exitosa la concurrencia en Java con hilos.
- La estructura modular del código permite su fácil mantenimiento y escalabilidad.
- La interfaz gráfica en Swing cumple con los requisitos del enunciado, aunque puede mejorarse con JavaFX.
- El manejo de archivos de texto asegura que los datos de personajes e historial se conserven entre ejecuciones.