Universidad de San Carlos de Guatemala

Facultad de Ingeniería

Introducción a la Programación y Computación 1

Sección F

Ing. William Escobar

Aux. Zenaida Chacón

Práctica 2 – Manual teórico

ArenaUSAC

Eldan André Escobar Asturias

Carné 202303088

04/10/2025

INTRODUCCIÓN

El presente proyecto consiste en el desarrollo de un programa en Java con interfaz gráfica de usuario (GUI) implementada mediante la librería Swing, cuyo objetivo principal es administrar y simular batallas entre personajes. La aplicación fue diseñada para ofrecer una experiencia completa al usuario, abarcando tanto la gestión de datos como la simulación de enfrentamientos, integrando en un mismo entorno múltiples funcionalidades que demuestran la versatilidad y potencia de la programación orientada a objetos.

Desde el menú principal, el programa permite realizar operaciones esenciales como agregar nuevos personajes, modificar sus atributos, eliminar registros existentes, buscar información específica, imprimir listados completos e incluso exportar datos a un archivo de texto. Estas opciones convierten al sistema en una herramienta flexible para la administración de entidades, ya que los personajes son tratados como objetos con propiedades definidas (nombre, ataque, defensa, vida, velocidad, entre otros atributos).

Adicionalmente, el programa cuenta con un módulo de simulación de batalla, donde dos personajes seleccionados se enfrentan bajo ciertas reglas predefinidas. Durante este proceso, se hace uso de la bitácora de eventos, que permite al usuario observar cómo se desarrolla el combate paso a paso, registrando ataques, defensas y resultados parciales hasta determinar un vencedor. Esta funcionalidad ilustra de manera práctica cómo los conceptos de la programación pueden aplicarse para recrear situaciones dinámicas e interactivas.

En cuanto al diseño técnico, la aplicación pone en práctica conceptos fundamentales de la programación en Java, como el uso de clases y objetos, la aplicación de la herencia y el encapsulamiento, el manejo de eventos, y la interacción entre distintos componentes gráficos. También se explora la gestión de archivos mediante operaciones de lectura y escritura, lo que refuerza la utilidad del proyecto para aplicaciones del mundo real, donde el almacenamiento de datos es indispensable.

En resumen, este programa no solo cumple la función de entretener al simular batallas, sino que también representa un ejercicio formativo de gran valor académico. Su construcción permitió integrar de manera práctica los conocimientos adquiridos en el área de programación, reforzando la comprensión de estructuras lógicas, control de flujo, diseño modular y trabajo con interfaces gráficas. Así, se convierte en un ejemplo claro del potencial que ofrece Java para el desarrollo de aplicaciones interactivas y escalables.

GLOSARIO DE TÉRMINOS CLAVE

package: Define un conjunto de clases agrupadas bajo un mismo espacio de nombres.

import: Permite usar clases de otras librerías o paquetes en el programa.

public: Modificador de acceso que indica que una clase, método o variable es accesible desde cualquier otra clase.

private: Modificador de acceso que limita la visibilidad únicamente dentro de la misma clase.

String: Clase en Java que representa cadenas de texto.

extends: Palabra clave para indicar herencia entre clases.

super: Permite acceder a métodos o constructores de la clase padre.

void: Tipo de retorno que indica que un método no devuelve ningún valor.

static: Permite acceder a un método o variable sin necesidad de crear una instancia de la clase.

new: Operador que crea un objeto en memoria.

this: Referencia al objeto actual de la clase.

DICCIONARIO DE MÉTODOS

setVisible(true): Muestra la ventana gráfica en pantalla.

addActionListener(): Asocia un evento (como clic en un botón) a un método que ejecuta una acción.

getSelectedItem(): Obtiene el valor actual seleccionado en un JComboBox.

append(): Agrega texto al final de un JTextArea.

setText(): Establece el contenido de un campo de texto o etiqueta.

getText(): Recupera el contenido escrito en un campo de texto.

setEnabled(): Activa o desactiva un componente gráfico.

setTitle(): Cambia el título de la ventana principal.

dispose(): Cierra una ventana sin finalizar toda la aplicación.

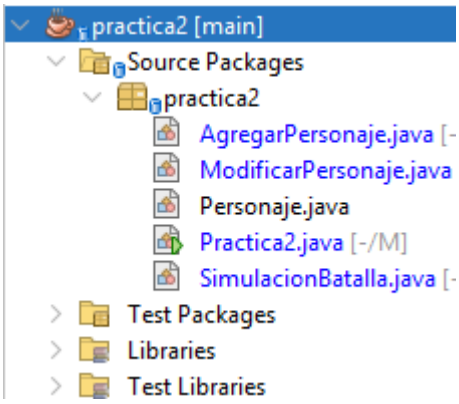
println(): Imprime un mensaje en consola.

write(): Escribe datos en un archivo de texto.

readLine(): Lee una línea de texto de un archivo o flujo de entrada.

CAPTURAS Y EXPLICACIÓN DE CÓDIGO

Se tienen 5 clases para que este programa funcione: la primera es la del menú principal, llamada “practica2” y en esta se encuentra la interfaz, y muchas funciones de cada botón; la segunda es llamada “personajes” y sirve únicamente para guardar todos los datos de personajes, y luego mostrarlos en las siguientes clases; la tercera se llama “agregar personaje”, y es una ventana nueva la cual contiene todos los textfield, label, y button para que se agreguen; la cuarta se llama “modificar personaje”, y es similar a la anterior, solo que aquí se imprimen automáticamente los datos del personaje que se busque previamente; y la última clase se llama “simular batalla”, y es la más compleja, ya que aquí es donde se hace uso de varios elementos para seleccionar los personajes que van a pelear, y también va mostrando cada acción de los personajes, hasta llegar al ganador.



Empezamos desde la clase del menú principal, y en primer lugar, se declaran las variables de los elementos a utilizar.

```
//variables para almacenamiento de personaje con vectores e importación de la clase
private Personaje[] personajes = new Personaje[100];
private int totalPersonajes = 0;

//variables con vectores para historial de batallas
private String[] historialBatallas = new String[100];
private int totalBatallas = 0;

//variables para creación de botones en la interfaz grafica
private JButton btnAgregar, btnModificar, btnEliminar, btnVerTodos, btnBuscar, btnBatalla, btnHistorial, btnGuardarDatos, btnEstudiante,

//ruta de almacenamiento de la bitacora
private static final String DESKTOP_PATH = System.getProperty("user.home") + File.separator + "Desktop";
private static final String BITACORA_FILE = DESKTOP_PATH + File.separator + "bitacora.txt";
```

Luego se crea una subclase pública para el diseño del menú principal, ya que todo es programado, y no es arrastrado con JFrame.

```java
public Practica2() {
    //diseño de ventana principal
    setTitle("Practica2");
    setSize(300, 500);
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setLayout(null);
    setLocationRelativeTo(null);

    //creacion de botones y distancia entre ellos
    int y = 20;
    btnAgregar = crearBoton("Agregar personaje", y); y += 40;
    btnModificar = crearBoton("Modificar personaje", y); y += 40;
    btnEliminar = crearBoton("Eliminar personaje", y); y += 40;
    btnVerTodos = crearBoton("Ver todos los personajes", y); y += 40;
    btnBuscar = crearBoton("Buscar personaje por ID", y); y += 40;
    btnBatalla = crearBoton("Simular batalla", y); y += 40;
    btnHistorial = crearBoton("Historial de batallas", y); y += 40;
    btnGuardarDatos = crearBoton("Guardar/Cargar datos", y); y += 40;
    btnEstudiante = crearBoton("Datos del estudiante", y); y += 40;
    btnSalir = crearBoton("Salir", y);

    registrarBitacora("Sistema iniciado");
}
```

La siguiente subclase es la que realiza las acciones de los botones creados previamente, y almacena los datos en la bitácora.

```java
private JButton crearBoton(String texto, int y) {
    JButton boton = new JButton(texto);
    boton.setBounds(50, y, 180, 30);

    boton.addActionListener(e -> registrarBitacora("Se presiono el boton: " + texto));

    if (texto.equals("Agregar personaje")) {
        boton.addActionListener(e -> {
            new AgregarPersonaje(Practica2.this).setVisible(true); //abrir clase para agregar personaje
            registrarBitacora("Se abrio la ventana para agregar personaje");
            dispose();
        });
    } else if (texto.equals("Modificar personaje")) {
        boton.addActionListener(e -> {
            new ModificarPersonaje(Practica2.this).setVisible(true); //abrir ventana para agregar personaje
            registrarBitacora("Se abrio la ventana para modificar personaje");
            dispose();
        });
    } else if (texto.equals("Eliminar personaje")) {
        boton.addActionListener(e -> eliminarPersonaje());
    } else if (texto.equals("Ver todos los personajes")) {
        boton.addActionListener(e -> verTodosLosPersonajes());
    } else if (texto.equals("Buscar personaje por ID")) {
        boton.addActionListener(e -> buscarPersonajePorID());
    } else if (texto.equals("Simular batalla")) {
        boton.addActionListener(e -> {
            new SimulacionBatalla(Practica2.this).setVisible(true); //abrir ventana para simular batalla
            registrarBitacora("Se abrio la ventana para simular batalla");
        });
    } else if (texto.equals("Historial de batallas")) {
        boton.addActionListener(e -> mostrarHistorialBatallas());
    } else if (texto.equals("Guardar/Cargar datos")) {
        boton.addActionListener(e -> guardarDatos());
    } else if (texto.equals("Datos del estudiante")) {
        boton.addActionListener(e -> datosEstudiante());
    } else if (texto.equals("Salir")) {
        boton.addActionListener(e -> {
            registrarBitacora("Gracias por usar el sistema");
            System.exit(0);
        });
    }

    add(boton);
```

Luego se tiene la subclase para eliminar personaje, la cual muestra un panel que pedirá al usuario ingresar el ID, y una vez validado que exista, mostrará un mensaje con la información del personaje, y pedirá una confirmación para eliminar; también se validará si no existe el personaje, o si se ingresó un valor inválido.

```java
private void eliminarPersonaje() {
    if (totalPersonajes == 0) {
        JOptionPane.showMessageDialog(this, "No hay personajes para eliminar"); //valodacion de existencia de personajes para eliminar
        return;
    }

    try { //buscar personaje para eliminar
        String input = JOptionPane.showInputDialog(this, "Ingrese el ID del personaje a eliminar (1-" + totalPersonajes + "):");
        if (input == null) return;

        int id = Integer.parseInt(input);

        if (id < 1 || id > totalPersonajes) { //validacion de id
            JOptionPane.showMessageDialog(this, "Ingrese un ID valido");
            return;
        }

        Personaje personaje = personajes[id-1];
        String mensaje = "¿Está seguro que desea eliminar este personaje?\n\n" +
                         "ID: " + id + "\n" +
                         "Nombre: " + personaje.getNombre() + "\n" +
                         "Arma: " + personaje.getArma() + "\n" +
                         "HP: " + personaje.getHp();

        int confirmacion = JOptionPane.showConfirmDialog(this, mensaje, "Confirmar Eliminación", JOptionPane.YES_NO_OPTION);
        //confirmacion de eliminacion de personaje
        if (confirmacion == JOptionPane.YES_OPTION) {
            registrarBitacora("Personaje eliminado: " + personaje.getNombre() + " (ID: " + id + ")");

            for (int i = id-1; i < totalPersonajes - 1; i++) {
                personajes[i] = personajes[i + 1];
            }
            personajes[totalPersonajes - 1] = null;
            totalPersonajes--;

            JOptionPane.showMessageDialog(this, "Personaje eliminado exitosamente");
        }

    } catch (NumberFormatException e) {
        JOptionPane.showMessageDialog(this, "Ingrese un número valido");
    }
}
```

Para la subclase de ver personajes y ver historial, simplemente se importarán los datos existentes de la clase de personajes, y se imprimirán con un mensaje.

```java
private void verTodosLosPersonajes() {
    if (totalPersonajes == 0) {
        JOptionPane.showMessageDialog(this, "No hay personajes guardados"); //valodacion de existencia de personajes para mostrar
        return;
    }

    StringBuilder sb = new StringBuilder();
    sb.append("Lista de personajes:\n\n");

    //imprimir personajes existentes
    for (int i = 0; i < totalPersonajes; i++) {
        Personaje p = personajes[i];
        sb.append("ID: ").append(i + 1).append("\n")
          .append("Nombre: ").append(p.getNombre()).append("\n")
          .append("Arma: ").append(p.getArma()).append("\n")
          .append("HP: ").append(p.getHp()).append("\n")
          .append("Ataque: ").append(p.getAtaque()).append("\n")
          .append("Velocidad: ").append(p.getVelocidad()).append("\n")
          .append("Agilidad: ").append(p.getAgilidad()).append("\n")
          .append("Defensa: ").append(p.getDefensa()).append("\n")
          .append("-------------------------\n");
    }

    JTextArea textArea = new JTextArea(sb.toString(), 20, 40);
    textArea.setEditable(false);
    JScrollPane scrollPane = new JScrollPane(textArea);

    JOptionPane.showMessageDialog(this, scrollPane, "Todos los Personajes", JOptionPane.INFORMATION_MESSAGE);

    registrarBitacora("Vista de todos los personajes");
}
```

```java
//mostrar historial de batallas en un área de texto
private void mostrarHistorialBatallas() {
    if (totalBatallas == 0) {
        JOptionPane.showMessageDialog(this, "No hay batallas registradas aun");
        return;
    }

    StringBuilder sb = new StringBuilder();
    sb.append("Historial de batallas:\n\n");
    for (int i = 0; i < totalBatallas; i++) {
        sb.append(historialBatallas[i]).append("\n\n");
    }

    JTextArea textArea = new JTextArea(sb.toString(), 20, 50);
    textArea.setEditable(false);
    JScrollPane scrollPane = new JScrollPane(textArea);
    JOptionPane.showMessageDialog(this, scrollPane, "Historial de Batallas", JOptionPane.INFORMATION_MESSAGE);

    registrarBitacora("Vista del historial de batallas");
}
```

Para buscar personaje, es igual que para eliminar, que se ingresa el ID, y si existe, se imprimirán todos los datos, esto por medio de un try catch.

```java
try { //buscar personaje por id
    String input = JOptionPane.showInputDialog(this,
        "Ingrese el ID del personaje a buscar: ");
    if (input == null) return;

    int id = Integer.parseInt(input);

    if (id < 1 || id > totalPersonajes) { //validacion de datos
        JOptionPane.showMessageDialog(this, "ID invalido");
        return;
    }

    //imprimir datos
    Personaje p = personajes[id-1];
    StringBuilder info = new StringBuilder();
    info.append("Personaje:\n\n")
        .append("ID: ").append(id).append("\n")
        .append("Nombre: ").append(p.getNombre()).append("\n")
        .append("Arma: ").append(p.getArma()).append("\n")
        .append("HP: ").append(p.getHp()).append("\n")
        .append("Ataque: ").append(p.getAtaque()).append("\n")
        .append("Velocidad: ").append(p.getVelocidad()).append("\n")
        .append("Agilidad: ").append(p.getAgilidad()).append("\n")
        .append("Defensa: ").append(p.getDefensa()).append("\n\n");

    //mostrar historial de batallas del personaje
    info.append("Historial de peleas del personaje:\n");
    boolean tieneBatallas = false;
    for (int i = 0; i < totalBatallas; i++) {
        if (historialBatallas[i].contains(p.getNombre())) {
            info.append(historialBatallas[i]).append("\n");
            tieneBatallas = true;
        }
    }
    if (!tieneBatallas) {
        info.append("El personaje no tiene batallas registradas aun\n");
    }

    JOptionPane.showMessageDialog(this, info.toString(), "Personaje ID: " + id, JOptionPane.INFORMATION_MESSAGE);

    registrarBitacora("Busqueda de personaje: " + p.getNombre() + " (ID: " + id + ")");
```

En cada acción se guardan datos en la bitácora, los cuales se mostrarán en la consola, así que aquí está el código para lograrlo.

```java
//registro de batalla para mostrar en la bitacora
public void registrarHistorialBatalla(String registro) {
    if (totalBatallas < historialBatallas.length) {
        String timestamp = new SimpleDateFormat("dd/MM/yyyy HH:mm:ss").format(new Date());
        historialBatallas[totalBatallas] = "[" + timestamp + "] " + registro;
        totalBatallas++;
        registrarBitacora("Batalla registrada: " + registro);
    }
}

public void registrarBitacora(String accion) {
    String timestamp = new SimpleDateFormat("dd/MM/yyyy HH:mm:ss").format(new Date());
    String linea = "[" + timestamp + "] " + accion;

    //mostrar en consola
    System.out.println(linea);

    //guardar en archivo bitácora en escritorio
    try (BufferedWriter writer = new BufferedWriter(new FileWriter(BITACORA_FILE, true))) {
        writer.write(linea);
        writer.newLine();
    } catch (IOException e) {
        JOptionPane.showMessageDialog(this, "Error al escribir en la bitacora: " + e.getMessage());
    }
}
```

Otro botón del menú principal, es para guardar o cargar datos, y este generará archivos ,txt los cuales se guardarán en una ruta previamente programada, en este caso, el escritorio, y los tres archivos que generará serán para los personajes, bitácora, y batallas. Para esto se necesita la librería de filewritter.

```java
private void guardarDatos() {
    try {
        //guardar archivo .txt de personajes en el escritorio
        FileWriter fw = new FileWriter(DESKTOP_PATH + File.separator + "personajes.txt");
        for (int i = 0; i < totalPersonajes; i++) {
            Personaje p = personajes[i];
            fw.write("ID: " + (i+1) + ", Nombre: " + p.getNombre() + ", Arma: " + p.getArma() +
                     ", HP: " + p.getHp() + ", Ataque: " + p.getAtaque() +
                     ", Velocidad: " + p.getVelocidad() + ", Agilidad: " + p.getAgilidad() +
                     ", Defensa: " + p.getDefensa() + "\n");
        }
        fw.close();

        //guardar archivo .txt de batallas en el escritorio
        FileWriter fw2 = new FileWriter(DESKTOP_PATH + File.separator + "batallas.txt");
        for (int i = 0; i < totalBatallas; i++) {
            fw2.write(historialBatallas[i] + "\n");
        }
        fw2.close();

        JOptionPane.showMessageDialog(this, "Datos guardados en el escritorio");
        registrarBitacora("Datos guardados en el escritorio");
    } catch (IOException e) {
        JOptionPane.showMessageDialog(this, "Error al guardar los datos: " + e.getMessage());
    }
}
```

Y por último en el menú principal, se pueden mostrar los datos del estudiante, aquí solo se escribió cada cosa, y se imprimen en un mensaje,

```java
private void datosEstudiante(){
    String nombre = "Eldan André Escobar Asturias";
    String carnet = "202303088";
    String curso = "IPC1-F";

    String mensaje= "Nombre: " + nombre + "\nCarnet: " + carnet + "\nCurso: " + curso;

    JOptionPane.showMessageDialog(this, mensaje, "Datos del estudiante", JOptionPane.INFORMATION_MESSAGE);
    registrarBitacora("Vista de los datos del estudiante");
}
```

La clase de personajes solo crea las variables, y usa getters y setters que se llamarán en otras clases.

```java
public class Personaje {
    private String nombre, arma;
    private int hp, ataque, velocidad, agilidad, defensa;

    public Personaje(String nombre, String arma, int hp, int ataque, int velocidad, int agilidad, int defensa) {
        this.nombre = nombre;
        this.arma = arma;
        this.hp = hp;
        this.ataque = ataque;
        this.velocidad = velocidad;
        this.agilidad = agilidad;
        this.defensa = defensa;
    }

    public String getNombre() {
        return nombre;
    }

    public void setNombre(String nombre) {
        this.nombre = nombre;
    }

    public String getArma() {
        return arma;
    }

    public void setArma(String arma) {
        this.arma = arma;
    }

    public int getHp() {
        return hp;
    }

    public void setHp(int hp) {
        this.hp = hp;
    }

    public int getAtaque() {
        return ataque;
    }
```

En la siguiente clase, se agregarán los personajes, y primero se declaran las variables, para luego generar la interfaz gráfica.

```
this.principal = principal;
setTitle("Agregar Personaje");
setSize(400, 420);
setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
setLayout(null);
setLocationRelativeTo(null);

int y = 20;
txtNombre = crearCampo("Nombre", y); y+=35;
txtArma = crearCampo("Arma", y); y+=35;
txtHp = crearCampo("Puntos de vida (100-500)", y); y+=35;
txtAtaque = crearCampo("Ataque (10-100)", y); y+=35;
txtVelocidad = crearCampo("Velocidad (1-10)", y); y+=35;
txtAgilidad = crearCampo("Agilidad (1-10)", y); y+=35;
txtDefensa = crearCampo("Defensa (1-50)", y); y+=35;

btnGuardar = new JButton("Guardar");
btnGuardar.setBounds(50, y, 120, 30);
btnGuardar.addActionListener(e -> guardarPersonaje());
add(btnGuardar);

btnMenu = new JButton("Menú Principal");
btnMenu.setBounds(200, y, 120, 30);
btnMenu.addActionListener(e -> {
    principal.setVisible(true);
    principal.registrarBitacora("Se regreso al menu principal");
    dispose();
});
add(btnMenu);
```

Como se crearon campos para guardar datos, en la subclase para guardar personaje se leerán estos datos y se almacenarán en sus respectivas variables, también se validará que los datos ingresados sean válidos y cumplan con lo solicitado.

```
try {
    String nombre = txtNombre.getText().trim();
    String arma = txtArma.getText().trim();
    int hp = Integer.parseInt(txtHp.getText().trim());
    int ataque = Integer.parseInt(txtAtaque.getText().trim());
    int velocidad = Integer.parseInt(txtVelocidad.getText().trim());
    int agilidad = Integer.parseInt(txtAgilidad.getText().trim());
    int defensa = Integer.parseInt(txtDefensa.getText().trim());

    //validaciones de datos
    if (nombre.isEmpty() || arma.isEmpty()) {
        throw new Exception("Llene los campos");
    }

    //validacion de existencia de nombre
    for (int i = 0; i < principal.getTotalPersonajes(); i++) {
        if (principal.getPersonajes()[i].getNombre().equalsIgnoreCase(nombre)) {
            throw new Exception("Ya existe un personaje con el nombre '" + nombre + "'.");
        }
    }

    if (hp < 100 || hp > 500) throw new Exception("HP debe estar entre 100 y 500");
    if (ataque < 10 || ataque > 100) throw new Exception("Ataque debe estar entre 10 y 100");
    if (velocidad < 1 || velocidad > 10) throw new Exception("Velocidad debe estar entre 1 y 10");
    if (agilidad < 1 || agilidad > 10) throw new Exception("Agilidad debe estar entre 1 y 10");
    if (defensa < 1 || defensa > 50) throw new Exception("Defensa debe estar entre 1 y 50");

    //crear y guardar personaje
    Personaje p = new Personaje(nombre, arma, hp, ataque, velocidad, agilidad, defensa);
    int id = principal.getTotalPersonajes();
    principal.getPersonajes()[id] = p;
    principal.setTotalPersonajes(id + 1);

    JOptionPane.showMessageDialog(this, "Personaje agregado con ID: " + (id+1));
    principal.registrarBitacora("Se agrego personaje: " + nombre + " con ID: " + (id+1));
    limpiarCampos();

} catch(NumberFormatException e) {
    JOptionPane.showMessageDialog(this, "Ingrese numeros validos");
    principal.registrarBitacora("Error al agregar personaje: campos numericos invalidos");
} catch(Exception e) {
    JOptionPane.showMessageDialog(this, "Error: " + e.getMessage());
    principal.registrarBitacora("Error al agregar personaje: " + e.getMessage());
```

En la otra clase, para modificar personaje, se declaran las variables y se crea la interfaz igual que en la clase anterior, pero aquí se comienza creando un textbox y un botón para ingresar el Id a buscar, así luego se muestran en cada campo como en agregar personaje, para luego modificarlos.

```
try {
    int id = Integer.parseInt(txtId.getText().trim()) - 1;
    if(id < 0 || id >= principal.getTotalPersonajes()) {
        JOptionPane.showMessageDialog(this, "No existe personaje con ese ID"); //validacion de existencia de id
        principal.registrarBitacora("Se busco un ID inexistente");
        return;
    }
    Personaje p = principal.getPersonajes()[id];
    lblNombre.setText("Nombre: " + p.getNombre());
    txtArma.setText(p.getArma());
    txtHp.setText(""+p.getHp());
    txtAtaque.setText(""+p.getAtaque());
    txtVelocidad.setText(""+p.getVelocidad());
    txtAgilidad.setText(""+p.getAgilidad());
    txtDefensa.setText(""+p.getDefensa());

    principal.registrarBitacora("Personaje buscado para modificar: " + p.getNombre() + " (ID: " + (id+1) + ")");
} catch(Exception e) {
    JOptionPane.showMessageDialog(this, "Ingrese un ID valido");
    principal.registrarBitacora("Se ingreso un ID invalido");
}
```

Una vez mostrados los datos, se puede seleccionar qué se va a modificar, así que solo se modifica el texto en los campos; este procedimiento es similar al de agregar personaje.

```
private void guardarCambios() {
    try {
        int id = Integer.parseInt(txtId.getText().trim()) - 1;
        if(id < 0 || id >= principal.getTotalPersonajes()) {
            JOptionPane.showMessageDialog(this, "No existe personaje con ese ID");
            principal.registrarBitacora("Error por el ID al guardar datos (" + (id+1) + ")");
            return;
        }

        //actualizar los datos que se modificaron
        Personaje p = principal.getPersonajes()[id];
        p.setArma(txtArma.getText().trim());
        p.setHp(Integer.parseInt(txtHp.getText().trim()));
        p.setAtaque(Integer.parseInt(txtAtaque.getText().trim()));
        p.setVelocidad(Integer.parseInt(txtVelocidad.getText().trim()));
        p.setAgilidad(Integer.parseInt(txtAgilidad.getText().trim()));
        p.setDefensa(Integer.parseInt(txtDefensa.getText().trim()));

        JOptionPane.showMessageDialog(this, "Datos modificados correctamente");
        principal.registrarBitacora("Datos modificados del personaje: " + p.getNombre() + " (ID: " + (id+1) + ")");
    } catch(Exception e) {
        JOptionPane.showMessageDialog(this, "Revise que todos los campos sean correctos");
        principal.registrarBitacora("Error al guardar cambios en personaje: datos invalidos");
    }
}
```

Para la última clase, correspondiente a la de simulación batalla, se creará la interfaz, y se crearán unos combobox los cuales permitan seleccionar el personaje a usar, también se tiene un textarea el cual mostrará las acciones de la batalla.

```java
public SimulacionBatalla(Practica2 principal) {
    this.principal = principal;
    setTitle("Simulacion de batalla");
    setSize(600, 500);
    setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
    setLayout(null);
    setLocationRelativeTo(null);

    //combobox para selección de personajes
    JLabel lblPersonaje1 = new JLabel("Personaje 1:");
    lblPersonaje1.setBounds(20, 20, 100, 25);
    add(lblPersonaje1);

    cmbPersonaje1 = new JComboBox<>();
    cmbPersonaje1.setBounds(120, 20, 150, 25);
    add(cmbPersonaje1);

    JLabel lblPersonaje2 = new JLabel("Personaje 2:");
    lblPersonaje2.setBounds(20, 50, 100, 25);
    add(lblPersonaje2);

    cmbPersonaje2 = new JComboBox<>();
    cmbPersonaje2.setBounds(120, 50, 150, 25);
    add(cmbPersonaje2);

    // Botones
    btnIniciarBatalla = new JButton("Iniciar Batalla");
    btnIniciarBatalla.setBounds(300, 35, 120, 30);
    btnIniciarBatalla.addActionListener(e -> iniciarBatalla());
    add(btnIniciarBatalla);

    btnMenu = new JButton("Menú Principal");
    btnMenu.setBounds(430, 35, 120, 30);
    btnMenu.addActionListener(e -> {
        principal.setVisible(true);
        principal.registrarBitacora("Se regreso al menu principal");
        dispose();
    });
    add(btnMenu);
```

Se tiene una subclase para cargar los personajes en los combobox, y mostrar sus datos.

```
private void cargarPersonajes() {
    cmbPersonaje1.removeAllItems();
    cmbPersonaje2.removeAllItems();

    for (int i = 0; i < principal.getTotalPersonajes(); i++) {
        Personaje p = principal.getPersonajes()[i];
        String item = (i + 1) + ". " + p.getNombre() + " (HP: " + p.getHp() + ")";
        cmbPersonaje1.addItem(item);
        cmbPersonaje2.addItem(item);
    }
    principal.registrarBitacora("Lista de personajes cargados en la simulacion");
}
```

Una vez seleccionados los personajes, se creará una subclase para iniciar batalla, en la cuál se comenzará validando que ambos personajes estén listos para pelear, eso quiere decir que ambos tienen que ser diferentes, tienen que tener más de 0 de vida, no se puede seleccionar solo un personaje, y solo se puede realizar una batalla al mismo tiempo.

```
private void iniciarBatalla() {
    if (batallaEnCurso) {
        JOptionPane.showMessageDialog(this, "Error, ya hay una batalla en curso");
        principal.registrarBitacora("Intento de iniciar batalla fallido: ya hay una en curso");
        return;
    }

    int index1 = cmbPersonaje1.getSelectedIndex();
    int index2 = cmbPersonaje2.getSelectedIndex();

    if (index1 == -1 || index2 == -1) {
        JOptionPane.showMessageDialog(this, "Error, debe seleccionar dos personajes");
        principal.registrarBitacora("Intento de iniciar batalla fallido: no se seleccionaron ambos personajes"
        return;
    }

    if (index1 == index2) {
        JOptionPane.showMessageDialog(this, "Error, debe seleccionar personajes diferentes");
        principal.registrarBitacora("Intento de iniciar batalla fallido: mismos personajes seleccionados");
        return;
    }

    Personaje p1 = principal.getPersonajes()[index1];
    Personaje p2 = principal.getPersonajes()[index2];

    // Validar que estén vivos
    if (p1.getHp() <= 0 || p2.getHp() <= 0) {
        JOptionPane.showMessageDialog(this, "Error, ambos personajes deben estar vivos para combatir");
        principal.registrarBitacora("Intento de iniciar batalla fallido: uno de los personajes está muerto");
        return;
    }

    batallaEnCurso = true;
    btnIniciarBatalla.setEnabled(false);
    txtBitacora.setText("");
```

Haciendo uso de la librería de fecha y hora, se mostrarán mensaje que lleven el registro de las batallas y almacenarán todos los datos.

```
batallaEnCurso = true;
btnIniciarBatalla.setEnabled(false);
txtBitacora.setText("");

// Registrar inicio de batalla
String horaInicio = new SimpleDateFormat("dd/MM/yyyy HH:mm:ss").format(new Date());
String registro = "Batalla iniciada: " + p1.getNombre() + " vs " + p2.getNombre() + " - " + horaInicio;
txtBitacora.append(registro + "\n\n");
principal.registrarHistorialBatalla(registro);
principal.registrarBitacora("Inicio de batalla: " + p1.getNombre() + " vs " + p2.getNombre());

// Crear copias de los personajes para la batalla
Personaje copiaP1 = new Personaje(p1.getNombre(), p1.getArma(), p1.getHp(), p1.getAtaque(),
                                  p1.getVelocidad(), p1.getAgilidad(), p1.getDefensa());
Personaje copiaP2 = new Personaje(p2.getNombre(), p2.getArma(), p2.getHp(), p2.getAtaque(),
                                  p2.getVelocidad(), p2.getAgilidad(), p2.getDefensa());

// Crear hilos para la batalla
Thread hilo1 = new Thread(new Combatiente(copiaP1, copiaP2, "Jugador 1", principal, this));
Thread hilo2 = new Thread(new Combatiente(copiaP2, copiaP1, "Jugador 2", principal, this));

hilo1.start();
hilo2.start();


ic void agregarLogBatalla(String mensaje) {
SwingUtilities.invokeLater(() -> {
    txtBitacora.append(mensaje + "\n");
    txtBitacora.setCaretPosition(txtBitacora.getDocument().getLength());
});
```

Se crea una subclase llamada combatientes, y es la que almacena los datos de los personajes mientras están en una batalla, principalmente va a almacenar los registros de cada acción.

```java
class Combatiente implements Runnable {
    private Personaje atacante;
    private Personaje oponente;
    private String nombreHilo;
    private Practica2 principal;
    private SimulacionBatalla ventanaBatalla;

    public Combatiente(Personaje atacante, Personaje oponente, String nombreHilo,
                       Practica2 principal, SimulacionBatalla ventanaBatalla) {
        this.atacante = atacante;
        this.oponente = oponente;
        this.nombreHilo = nombreHilo;
        this.principal = principal;
        this.ventanaBatalla = ventanaBatalla;
    }

    @Override
    public void run() {
        try {
            while (oponente.getHp() > 0 && atacante.getHp() > 0) {
                int tiempoEspera = 1000 / atacante.getVelocidad();
                tiempoEspera = Math.min(Math.max(tiempoEspera, 200), 2000); // Entre 200ms y 2s
                Thread.sleep(tiempoEspera);

                if (oponente.getHp() <= 0) break;

                realizarAtaque();
            }

            if (atacante.getHp() > 0 && oponente.getHp() <= 0) {
                ventanaBatalla.batallaTerminada(atacante, oponente);
            }

        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
```

En la subclase anterior se guardan los registros, y en esta subclase llamada realizar ataque, se muestran las acciones.

```java
private void realizarAtaque() {
    double probabilidadGolpe = 0.7 + (atacante.getAgilidad() * 0.01);
    boolean golpeExitoso = Math.random() < probabilidadGolpe;

    if (golpeExitoso) {
        int daño = atacante.getAtaque() - (oponente.getDefensa() / 3);
        daño = Math.max(daño, 1);

        oponente.setHp(oponente.getHp() - daño);

        String log =  atacante.getNombre() + " ataca a " + oponente.getNombre() +
                      " - Golpe! (" + daño + " de daño) - HP restante: " +
                      (oponente.getHp() > 0 ? oponente.getHp() : "0");
        ventanaBatalla.agregarLogBatalla(log);
        principal.registrarBitacora("Ataque exitoso: " + atacante.getNombre() + " -> " + oponente.getNombre() + " (" + daño + " daño)");
    } else {
        String log = atacante.getNombre() + " ataca a " + oponente.getNombre() + " - fallo";
        ventanaBatalla.agregarLogBatalla(log);
        principal.registrarBitacora("Ataque fallido: " + atacante.getNombre() + " -> " + oponente.getNombre());
    }
}
```

POSIBLES MEJORAS A FUTURO

- Agregar más atributos a los personajes (como magia, resistencia o habilidades especiales).
- Incorporar animaciones gráficas que representen los ataques y defensas.
- Guardar y cargar personajes desde un archivo externo (XML, JSON o base de datos).
- Agregar un sistema de niveles y experiencia, para que los personajes evolucionen tras cada batalla.
- Mejorar la interfaz gráfica con iconos, imágenes de los personajes y menús más dinámicos.
- Implementar inteligencia artificial básica para que un personaje controlado por la máquina tome decisiones.
- Modo multijugador en red, donde dos usuarios puedan conectarse y simular la batalla desde diferentes computadoras.

CONCLUSIÓN

El desarrollo de este proyecto representó un desafío integral que abarcó desde el diseño conceptual hasta la implementación de una aplicación completa en Java, capaz de gestionar información y simular procesos dinámicos como batallas entre personajes. El programa no se limita a una única función, sino que incorpora un sistema de gestión de personajes con opciones de agregar, modificar, eliminar, buscar e imprimir datos, lo que refleja la importancia de ofrecer al usuario un entorno flexible y adaptable a distintas necesidades. A su vez, la posibilidad de crear archivos de texto en los que se almacena la información fortalece la dimensión práctica del proyecto, pues vincula la aplicación con el manejo de persistencia de datos, un aspecto clave en el desarrollo de software real.

Uno de los puntos más relevantes del sistema es la simulación de batallas, que permite observar en tiempo real cómo interactúan los personajes a través de sus atributos y probabilidades. Esta funcionalidad, además de ser un atractivo para el usuario, evidencia la correcta aplicación de estructuras de control, ciclos, métodos y registros de información, los cuales se combinan para generar una experiencia coherente y dinámica.

El proyecto, además, abre la puerta a una serie de mejoras futuras que podrían potenciar su alcance. Entre ellas destacan la incorporación de imágenes y animaciones para enriquecer la experiencia visual, la implementación de un sistema de niveles o evolución de personajes, el uso de bases de datos en lugar de archivos de texto para un manejo más robusto de la información, y la posibilidad de habilitar un modo multijugador en red que permita a distintos usuarios conectarse entre sí.