



Universidad de San Carlo de Guatemala
Facultad de Ingeniería
Escuela de Ciencias y Sistemas
Introducción a la programación y computación 1
Ing. Moisés Velásquez
Auxiliar. Pablo Oliva

PRACTICA No.2: MANUAL TECNICO

Guillermo Enrique Marroquin Morán

202103527

Guatemala 26 de sep. de 25

INDICE

MANUAL TÉCNICO	3
REQUERIMIENTOS MÍNIMOS DEL PROGRAMA.	3
EXPLICACIÓN DE LAS PARTES DEL PROYECTO.	4
LIBRERÍAS USADAS	4
EXPLICACION DEL CODIGO.....	6
CLASE DEL MENU PRINCIPAL:.....	6
CLASE PERSONAJES	7
<i>Modulo buscar personajes y personajes:</i>	7
<i>Modulo de guardar personajes:</i>	8
CLASE Y MODULO DE MODIFICAR PERSONAJE	9
<i>Modulo mostrar personajes:</i>	10
<i>Módulo de guardar personajes:</i>	10
CLASE DE ELIMINAR PERSONAJE.....	12
<i>Módulo personaje eliminado.</i>	13
<i>Módulo confirmar eliminación y eliminar personaje.</i>	14
CLASE Y MÓDULO DE VER PERSONAJE Y MÓDULO DE TABLA DE PERSONAJES.....	15
CLASE SIMULAR COMBATES	16
<i>Módulo simular combates.....</i>	16
<i>Módulo cargar personajes</i>	17
<i>Módulo iniciar combate.....</i>	17
<i>Modulo combate.....</i>	18
<i>Módulo actualizar historial</i>	19
CLASE DE HISTORIAL COMBATE.....	20
<i>Módulo historial de combates.</i>	20
<i>Módulo cargar historial.</i>	21
CLASE BUSCAR PERSONAJES POR NOMBRE	22
<i>Modulo buscar personajes.....</i>	23
CLASE GESTOR DE ARCHIVOS	24
<i>Módulo de guardar historial de personajes.</i>	24
<i>Modulo de cargar historial de personajes.....</i>	24
<i>Módulo de borrar datos.....</i>	25
<i>Clase validar acción</i>	25
<i>Modulo Registrar acción.</i>	26
<i>Módulo mostrar borrado de bitácora.</i>	26
CLASE Y MÓDULO DE DATOS DEL ESTUDIANTE	27

Manual Técnico

Requerimientos mínimos del programa.

Para instalar apache NetBeans los requisitos mínimos son:

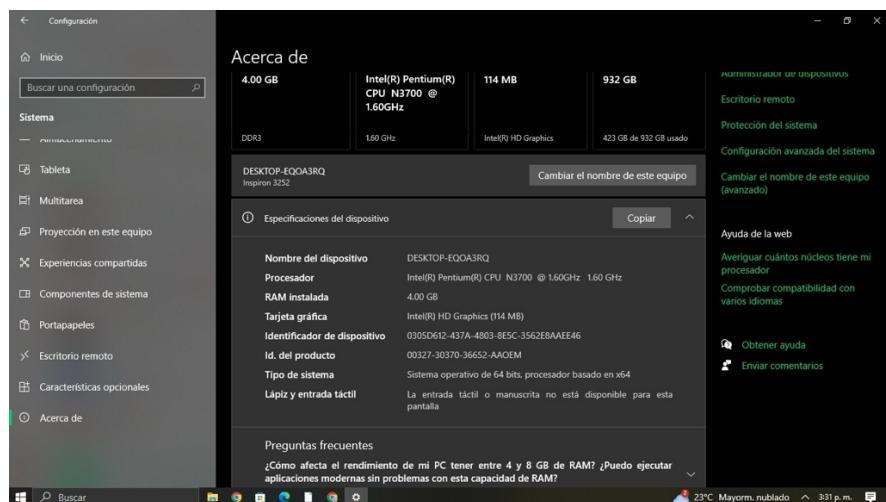
- 781 MB de Espacio Libre en el Disco Duro.
- 512 MB de RAM.
- Procesador Intel Pentium III a 800 MHz.
- Compatible con Windows, macOS y Linux.

(Janl, 2022)

Requisitos del ordenador usado para el proyecto:



Requisitos del ordenador con sistema Windows:



Explicación de las partes del proyecto.

Librerías usadas

```
import java.awt.BorderLayout;
import java.awt.FlowLayout;
import java.awt.Font;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.Random;
import javax.swing.JButton;
import javax.swing.JComboBox;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JTextArea;
import javax.swing.SwingConstantsConstants;
import javax.swing.SwingUtilities;
```

Las librerías se explicarán por funciones, así más fácil:

Para dibujar, los botones, las ventanas y texto en las mismas

- javax.swing.JPanel: ayuda agrupar botones o textos en una ventana
- javax.swing.JFrame: básicamente un lienzo en blanco donde vamos a poner los botones el texto etc.
- javax.swing.JLabel: ayuda para poner un texto estático que no sea mover y básicamente para poner las bienvenidas, o los datos de los personajes.
- javax.swing.JButton: nos ayuda a crear un botón que luego podemos agregarle una función.
- javax.swing.JTextArea: es un área de texto básicamente para que el usuario pueda escribir dentro de él.
- javax.swing.JComboBox: una lista desplegable, la use para la parte de simular combates.
- javax.swing.JScrollPane: ayuda que si en dado caso es demasiado contenido la pantalla automáticamente vaya bajando o desplazándose.

Para acciones y organización:

- java.awt.BorderLayout y java.awt.FlowLayout: ayuda a que se pueda repositionar todos los bloques de texto botones y texto como tal en la ventana, BorderLayout, nos permite poner

cosas de arriba abajo izquierda o derecha, y FlowLayout nos deja poner cosas una tras otra sin que estén solapadas.

- `java.awt.Font`: nos ayuda a cambiar el tipo de letra o sea la fuente o el tamaño.
- `javax.swing.SwingConstantsConstants`: nos ayuda a decir si el texto va a ir a la izquierda o la derecha en su marido lo usé para que todos los textos estén centrados.

Acciones de los botones y las ventanas del Joptionpane:

- `java.awt.event.ActionListener` y `java.awt.event.ActionEvent`: nos ayuda a indicar si se hace una cosa llame al método o la acción que nosotros queramos.
- `javax.swing.JOptionPane`: nos ayudas en mensajes emergentes o ventanas de confirmación o selección de dos opciones.

Hora, fecha y lo demás

- `java.util.Date` y `java.text.SimpleDateFormat`: se debe establecer la fecha y la hora y ponerle un formato que queramos
- `javax.swing.SwingUtilities`: no se ayuda a que la parte gráfica se actualice sin ninguna dificultad.

Para los archivos:

- `java.io.File`: nos ayuda para representar un archivo o una carpeta dentro del ordenador.
- `java.io.FileWriter`: nos ayuda a escribir dentro del
- `java.io.PrintWriter`: al igual que la Librería anterior nos permite escribir dentro del archivo de manera más cómoda.
- `java.io.IOException`: una condición si en dado caso el archivo no se va a encontrar para evitar que el programa se cierre.

EXPLICACION DEL CODIGO.

Clase del menu principal:

Esta clase se encarga principalmente de albergar en un panel o ventana, los botones de acción correspondientes a su nombre con sus debida llamada al método en clases separadas, y al final se puede visualizar que cada botón se está agregando al panel principal en orden. Esto gracias a JPanel, para la creación de la ventana, JButton, para la creacion de botones, y ActionListener para hacer un llamado a un método para así que el botón ejecute la acción deseada.

```
public menuPrincipal() {
    setTitle("Area USAC----- Menú Principal");
    //el tamaño de la venta
    setSize(500, 600);
    setLocationRelativeTo(null);
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); //se cierra la app se termina

    //un panel con un GridLayout de 10 filas y una columna (por las opciones)
    JPanel panel = new JPanel(new GridLayout(10, 1, 10, 10));

    //un titulo bonito para el menu
    JLabel titulo = new JLabel("Menu Principal", SwingConstants.CENTER);
    titulo.setFont(new Font("Arial", Font.BOLD, 25));
    panel.add(titulo);

    //creamos TODOS LOS BOTONES PARA CADA OPCION
    JButton btAgregar = new JButton("1. Agregar Personaje");
    //accion del boton agregar personaje
    btAgregar.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) {
            Personajes agregarVentana = new Personajes();
            agregarVentana.setVisible(true);
        }
    });
    JButton btModificar = new JButton("2. Modificar Personaje");
    //accion del boton de modificar personajes
    btModificar.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) {
            modificarPersonaje modificarVentana = new modificarPersonaje();
            modificarVentana.setVisible(true);
        }
    });
    JButton btHistorial = new JButton("6. Ver Historial de Batallas");
    //Acción del boton de historial
    btHistorial.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) {
            historialCombate VentanaHistorial = new historialCombate();
            VentanaHistorial.setVisible(true);
        }
    });

    JButton btBuscar = new JButton("7. Buscar Personaje");
    //accion del boton este...
    btBuscar.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) {
            BuscarPersonaje VentanaBuscar = new BuscarPersonaje();
            VentanaBuscar.setVisible(true);
        }
    });

    JButton btGuardar = new JButton("8. Guardar Estado del Sistema");
    //accion del boton guardar
    btGuardar.addActionListener(e -> gestorArchivo.GuardarPersonaje());

    JButton btCargar = new JButton("9. Cargar Estado del sistema");
    //accion bon cargar
    btCargar.addActionListener(e -> gestorArchivo.CargarPersonajes());

    JButton btLimpiar = new JButton("10. Limpiar Estado del Sistema");
    //Acción del boton de limpiar
    btLimpiar.addActionListener(e -> gestorArchivo.BorrarDatos ());

    JButton btBorrar = new JButton("11. Limpiar bitacora de acciones");
    //accion boton de bitacora
    btBorrar.addActionListener(e -> validarAccion.MostrarBit ());

    JButton btEliminar = new JButton("3. Eliminar Personaje");
    //Acción del boton de eliminar
    btEliminar.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) {
            eliminarPersonaje eliminarVentana = new eliminarPersonaje();
            eliminarVentana.setVisible(true);
        }
    });

    JButton btVer = new JButton("4. Ver Personajes Registrados");
    //Acción del boton de ver personajes :3
    btVer.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) {
            verPersonaje verVentana = new verPersonaje();
            verVentana.setVisible(true);
        }
    });

    JButton btSimular = new JButton("5. Simulacion de Combates");
    //Acción boton de MADrazos
    btSimular.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) {
            SimPelea combateVentana = new SimPelea();
            combateVentana.setVisible(true);
        }
    });

    JButton btYO = new JButton("12. Ver Datos del Estudiante ");
    btYO.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) {
            YOYYYYO ventanaYO = new YOYYYYO(); // si soy yo, que mas xd
            ventanaYO.setVisible(true);
        }
    });

    //en mac el menu se ve bonito la verdad, VIVA MAC y sus precios no
    panel.add(btAgregar);
    panel.add(btModificar);
    panel.add(btEliminar);
    panel.add(btVer);
    panel.add(btSimular);
    panel.add(btHistorial);
    panel.add(btBuscar);
    panel.add(btGuardar);
    panel.add(btCargar);
    panel.add(btLimpiar);
    panel.add(btBorrar);
    panel.add(btYO);
    add(panel);
}
```

Clase Personajes

Acá lo que hacemos es declarar variables que podremos usar tanto en esta clase como para llamarlas en los demás métodos para diferentes usos, usamos JTextField, para crear áreas de texto donde el usuario podrá escribir tanto el nombre arma puntos de vida ataque velocidad agilidad y defensa del personaje, y creamos la matriz de personajes que será de 100×9 lo que indicará 100 personajes y nueve atributos diferentes.

```
* @author kiquemarroquin
*/
public class Personajes extends JFrame {

    //una matriz de 50*9 para guardar a los personajes
    public static String[][] personaje = new String[100][9];

    //Un contador estatico para que generar el ID sea unico y poderlos rastrear
    public static int ContPersonajes = 0;
    public static int UID = 0;
    public static int contadorID = 1;
    // contantes para la matriz
    public static final int ID = 0;
    public static final int NOMBRE = 1;
    public static final int ARMA = 2;
    public static final int HP = 3;
    public static final int ATAQUE = 4;
    public static final int VELOCIDAD = 5;
    public static final int AGILIDAD = 6;
    public static final int DEFENSA = 7;
    public static final int VICTERR = 8;//Victorias y derrotas
    public static final int NUMCAMPOPERSONAJE = 9;

    //componentes de esta interfaz
    private JTextField txtNombre;
    private JTextField txtArma;
    private JTextField txtHp;
    private JTextField txtAtaque;
    private JTextField txtVelocidad;
    private JTextField txtAgilidad;
    private JTextField txtDefenso;
```

Modulo buscar personajes y personajes:

Éste módulo usa un ciclo For para buscar en la matriz de personajes el nombre que queremos encontrar o el ID del mismo, retornando el valor de -1 si lo encuentra, en el módulo/método de personajes crearemos una ventana de 900×900 , con la opción de qué cuando se cierre la ventana se cierra el proceso, en esta misma ventana agregaremos áreas de texto donde el usuario va a escribir los apartados de los atributos del personaje que quiera, y se agregó un botón para guardar al personaje haciendo llamado a el método que explicaremos a continuación.

```
//una clase publica para hacer llamados en todas partes mejor, ya me irritede hacer lo mismo ashuddaaaaaa
public static int buscarPer(String buscado) {
    if (buscado == null || buscado.isEmpty())
        return -1;
}
String buscar = buscado.trim().toLowerCase();

for (int i = 0; i < ContPersonajes; i++) {
    String id = personaje[i][ID];
    String nombre = personaje[i][NOMBRE].toLowerCase();

    if (id.equals(buscar) || nombre.equals(buscar)) {
        return i; //devolveremos el indice del personaje que encontramos
    }
}
return -1;
}

//ahora si la clase para guardar el personaje
public Personajes() {
    setTitle("Agregar Personaje");
    setSize(900, 900);
    setLocationRelativeTo(null); //para CENTRARLA
    //solo se va a ocultar para poder reusarla, viva el reciclaje
    setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);

    JPanel panel = new JPanel(new GridLayout(9, 2, 10, 10));
    //el título del panel NO ARRIBITA si no que... bueno DENTRO del panel
    JLabel titulo = new JLabel("Agregue un nuevo personaje", SwingConstants.CENTER);
    titulo.setFont(new Font("Arial", Font.BOLD, 18));
    panel.add(titulo);
    panel.add(new JLabel("")); //un espacio en blanco como el System.out.println();
```

Modulo de guardar personajes:

En este módulo tenemos la característica de qué si en dado caso el usuario no llena todos los apartados mandará un mensaje de error indicando que tiene que llenar todos los apartados, junto a las validaciones de puntos de vida, ataque, velocidad, agilidad y defensa junto a sus ventanas de mensaje de error, lo siguiente al pasar todas las validaciones el personaje se guardará en la matriz de personajes de esta manera aumentando en uno la cantidad de personajes agregados a la matriz mandando un mensaje de qué el personaje se ha agregado al Rooster de personajes. Junto al llamado del método de validación de acciones.

```
private void guardarPersonaje() {
    if (CantPersonajes > 100) {
        JOptionPane.showMessageDialog(this, "El limite de personajes se alcanzo, borra alguno", "Error 001", JOptionPane.ERROR_MESSAGE);
        validarAcción.regisAcción("agregar personaje", false, "Salio mal");
        return;
    }
    //todo esto es para que no hayan espacios sin llenar :3 muy inteligente
    String nombre = txtNombre.getText();
    if (nombre.isEmpty() || txtArma.getText().isEmpty() || txtHp.getText().isEmpty() || txtAtaque.getText().isEmpty() || txtVelocidad.getText().isEmpty() || txtAgilidad.getText().isEmpty() || txtDefensa.getText().isEmpty()) {
        JOptionPane.showMessageDialog(this, "Por favor llena todos los apartados D:", "Error 002", JOptionPane.ERROR_MESSAGE);
        validarAcción.regisAcción("agregar personaje", false, "Salio mal APROPOSITO");
        return;
    }

    int hp, ataque, velocidad, agilidad, defensa;
    try {
        hp = Integer.parseInt(txtHp.getText());
        if (hp < 100 || hp > 500) {
            JOptionPane.showMessageDialog(this, "Los puntos de vida deben de ser un numero entero de 100 a 500", "Error 003", JOptionPane.ERROR_MESSAGE);
            validarAcción.regisAcción("agregar vida personaje", false, "Salio mal APROPOSITO");
            return;
        }

        ataque = Integer.parseInt(txtAtaque.getText());
        if (ataque < 10 || ataque > 100) {
            JOptionPane.showMessageDialog(this, "Los puntos de ataque deben de ser un numero entero de 10 a 100", "Error 004", JOptionPane.ERROR_MESSAGE);
            validarAcción.regisAcción("agregar ataque personaje", false, "Salio mal APROPOSITO");
            return;
        }

        velocidad = Integer.parseInt(txtVelocidad.getText());
        if (velocidad < 1 || velocidad > 10) {
            JOptionPane.showMessageDialog(this, "La velocidad debe de ser un numero entero de 1 a 10", "Error 005", JOptionPane.ERROR_MESSAGE);
            validarAcción.regisAcción("agregar velocidad personaje", false, "Salio mal APROPOSITO");
            return;
        }

        agilidad = Integer.parseInt(txtAgilidad.getText());
        if (agilidad < 1 || agilidad > 10) {
            JOptionPane.showMessageDialog(this, "La agilidad debe de ser un numero entero de 1 a 10", "Error 006", JOptionPane.ERROR_MESSAGE);
            validarAcción.regisAcción("agregar agilidad personaje", false, "Salio mal APROPOSITO");
            return;
        }

    } catch (NumberFormatException e) {
        JOptionPane.showMessageDialog(this, "Los apartados que son numeros deben ser llenados con eso... pos numeros", "Error 008", JOptionPane.ERROR_MESSAGE);
        validarAcción.regisAcción("agregar personaje", false, "Salio mal APROPOSITO");
        return;
    }

    personaje[CantPersonajes][ID] = String.valueOf(Personajes.contadorID);
    personaje[CantPersonajes][NOMBRE] = txtNombre.getText();
    personaje[CantPersonajes][ARMA] = txtArma.getText();
    personaje[CantPersonajes][HP] = txtHp.getText();
    personaje[CantPersonajes][ATAQUE] = txtAtaque.getText();
    personaje[CantPersonajes][VELOCIDAD] = txtVelocidad.getText();
    personaje[CantPersonajes][AGILIDAD] = txtAgilidad.getText();
    personaje[CantPersonajes][DEFENSA] = txtDefensa.getText();
    personaje[CantPersonajes][VICIDER] = "0-0"; //Inciamos esto con nada

    CantPersonajes++; //se suma un personaje mas al plantel xddddd
    contadorID++; //otro chups wuuuuu
    JOptionPane.showMessageDialog(this, txtNombre.getText() + " se ha agregado al roster de personajes ");
    txtNombre.setText("");
    txtArma.setText("");
    txtHp.setText("");
    txtAtaque.setText("");
    txtVelocidad.setText("");
    txtAgilidad.setText("");
    txtDefensa.setText("");
    validarAcción.regisAcción("agregar personaje", true, "Salio bien");
}
```

Clase y modulo de modificar personaje

En este módulo se hace exactamente lo mismo que en la clase de personajes sólo que esta vez declaramos las variables para los atributos actuales del personaje y los que serán renovados o cambiados, de igual manera como áreas de texto los que son para cambiar, y sólo texto los atributos actuales del personaje. Siguiendo con el módulo de modificar personaje, se crea una ventana donde encontrarán los textos que mencioné anteriormente, las áreas de texto que mencioné anteriormente también, y un botón con la acción para buscar al personaje que queramos modificar, en la ventana se mostrarán los atributos actuales y luego al cambiar los datos mostrará los datos actuales después de cambiarlos.

```
public class modificarPersonaje extends JFrame {  
  
    private JTextField txtBuscar;  
    private JButton btBuscar;  
  
    //Este para mostrar los datos actuales del personaje en cuestión  
    private JLabel LbNombreAct;  
    private JLabel LbArmaAct;  
    private JLabel LbHpAct;  
    private JLabel LbAtaqueAct;  
    private JLabel LbVelocidadAct;  
    private JLabel LbAgilidadAct;  
    private JLabel LbDefensaAct;  
  
    //Campos de texto para editar los nuevos campos  
    private JTextField txtNArma;  
    private JTextField txtNHp;  
    private JTextField txtNAtaque;  
    private JTextField txtNVelocidad;  
    private JTextField txtNAgilidad;  
    private JTextField txtNDefensa;  
  
    private JButton btGuardar;  
  
    //esto es para almacenar el índice del personaje que se encuentre en la matriz  
    private int pEncontrar = -1;  
  
    public modificarPersonaje() {  
        setTitle("Modificar Personaje"); // título de la ventana  
        setSize(500, 600); //tamaño de la ventana  
        setLocationRelativeTo(null); // centramos la ventana  
        setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE); //ya se la saben para que al cerrar se cierre el proceso  
  
        //Campos que se van a llenar como los nuevos datos  
        panel.add(new JLabel("Escribe el arma de tu preferencia: "));  
        txtNArma = new JTextField();  
        panel.add(txtNArma);  
  
        panel.add(new JLabel("Ingresa los puntos de vida (100 a 500):"));  
        txtNHp = new JTextField();  
        panel.add(txtNHp);  
  
        panel.add(new JLabel("Ingresa los puntos de ataque (10 a 100)"));  
        txtNAtaque = new JTextField();  
        panel.add(txtNAtaque);  
  
        panel.add(new JLabel("Ingresa los puntos de velocidad (1 a 10)"));  
        txtNVelocidad = new JTextField();  
        panel.add(txtNVelocidad);  
  
        panel.add(new JLabel("Ingresa los puntos de agilidad (1 a 10)"));  
        txtNAgilidad = new JTextField();  
        panel.add(txtNAgilidad);  
  
        panel.add(new JLabel("Ingresa los puntos de defensa: "));  
        txtNDefensa = new JTextField();  
        panel.add(txtNDefensa);  
  
        btGuardar = new JButton("Guardar Cambios");  
        panel.add(btGuardar);  
  
        //Agregamos todo esto a la ventana en cuestión  
        add(panel);  
  
        btBuscar.addActionListener(new ActionListener() {  
            @Override  
            public void actionPerformed(ActionEvent e) {  
                buscarPer();  
            }  
        });  
    }  
  
    JPanel panel = new JPanel(new GridLayout(12, 2, 10, 20));  
    //titulo de la ventana  
    JLabel titulo = new JLabel("Modificar personaje", SwingConstants.CENTER);  
    titulo.setFont(new Font("Arial", Font.BOLD, 20));  
    panel.add(titulo);  
    panel.add(new JLabel(""));  
  
    //Apartado para buscar el personaje  
    panel.add(new JLabel("Buscar por ID o Nombre"));  
    txtBuscar = new JTextField();  
    panel.add(txtBuscar);  
  
    btBuscar = new JButton("Buscar");  
    panel.add(btBuscar);  
    panel.add(new JLabel(""));  
  
    //Esto es para que se vean los datos actuales  
    LbNombreAct = new JLabel("Nombre: ");  
    panel.add(LbNombreAct);  
    LbArmaAct = new JLabel("Arma: ");  
    panel.add(LbArmaAct);  
    LbHpAct = new JLabel("Puntos de salud: ");  
    panel.add(LbHpAct);  
    LbAtaqueAct = new JLabel("Ataque: ");  
    panel.add(LbAtaqueAct);  
    LbVelocidadAct = new JLabel("Velocidad: ");  
    panel.add(LbVelocidadAct);  
    LbAgilidadAct = new JLabel("Agilidad: ");  
    panel.add(LbAgilidadAct);  
    LbDefensaAct = new JLabel("Defensa: ");  
    panel.add(LbDefensaAct);  
    panel.add(new JLabel(""));  
    panel.add(new JLabel(""));  
  
    //agregamos la acción al botón de guardar  
    btGuardar.addActionListener(new ActionListener() {  
        @Override  
        public void actionPerformed(ActionEvent e) {  
            guardarDatos(); //Un llamado a un método  
        }  
    });  
  
    private void buscarPer(){  
        String buscar = txtBuscar.getText().trim();  
        pEncontrar = -1; //resetear el índice  
  
        if (buscar.isEmpty()) {  
            JOptionPane.showMessageDialog(this, "Ingresa un ID o Nombre para buscar", "Error 011", JOptionPane.ERROR_MESSAGE);  
            validarAcción.regisAcción("Buscar personaje para modificarlo", false, "Salio MAL APROPOSITO");  
  
            return;  
        }  
  
        //el ciclo for de toda la vida para buscar la cadena de texto en la matriz de personaje  
        for (int i = 0; i < Personajes.Personajes.length; i++) {  
            String Id = Personajes.Personajes[i][Personajes.ID];  
            String nombre = Personajes.Personajes[i][Personajes.NOMBRE];  
  
            if (Id.equals(buscar) || nombre.equalsIgnoreCase(buscar)) {  
                pEncontrar = i;  
                mostrarPer(); //Otro llamado  
                return;  
            }  
        }  
        JOptionPane.showMessageDialog(this, "El personaje no se ha encontrado", "Error 012", JOptionPane.ERROR_MESSAGE);  
        validarAcción.regisAcción("Buscar personaje para modificarlo", false, "Salio MAL");  
    }  
}
```

Modulo mostrar personajes:

Éste módulo se encargará principalmente de imprimir en pantalla los datos actuales del personaje, y luego de haber modificado los atributos mostrar los datos ya modificados, esto como un texto estático, después de haber encontrado al personaje en la matriz de personajes, esta acción está ligada al botón para mostrar al personaje después de buscar

```
//un metodo para mostrar el personaje, solo que aca es con JLabel en vez del System.out.println
private void mostrarPer(int i){
    String[] datos = Personajes.personaje [i];
    LbNombreAct.setText("Nombre: " + datos[Personajes.NOMBRE ]);
    LbArmaAct.setText("Arma: " + datos[Personajes.ARMA ]);
    LbHpAct.setText("Puntos de vida: " + datos[Personajes.HP ]);
    LbAtaqueAct.setText("Ataque: " + datos[Personajes.ATAQUE ]);
    LbVelocidadAct.setText("Velocidad: " + datos[Personajes.VELOCIDAD ]);
    LbAgilidadAct.setText("Agilidad: " + datos[Personajes.AGILIDAD ]);
    LbDefensaAct.setText("Defensa: " + datos[Personajes.DEFENSA ]);

    //CARGAMOS LOS DATOS PARA EDITAR PARA QUE SEA MAS FACIL MODIFICARLOS
    txtNArma.setText(datos[Personajes.ARMA ]);
    txtNHp.setText(datos[Personajes.HP ]);
    txtNAtaque.setText(datos[Personajes.ATAQUE ]);
    txtNVelocidad.setText(datos[Personajes.VELOCIDAD ]);
    txtNAgilidad.setText(datos[Personajes.AGILIDAD ]);
    txtNDefensa.setText(datos[Personajes.DEFENSA ]);
}
```

Módulo de guardar personajes:

De igual manera como mencioné antes este método está ligado a un botón más específico al botón de guardar datos, en este método o módulo tenemos las validaciones de igual manera que en la clase de personajes para evitar que el usuario ingrese valores no deseados, al cumplir las validaciones se cargarán los datos ya modificados asociados al personaje actualizando la matriz de personajes, y como último tenemos un método pequeño que se encarga de limpiar los apartados que ya se llenaron para evitar duplicados o fallas en el programa.

```
private void guardarDatos() {
    if (pEncontrar == -1) {
        JOptionPane.showMessageDialog(this, "Por favor, buscar un personaje primero", "Error 013", JOptionPane.ERROR_MESSAGE);
        validarAcción.regisAcción("Buscar personaje para modificarlo", false, "Salio MAL APROPOSITO");
        return;
    }

    //VALIDAR LOS VALORES:
    try {
        int hp = Integer.parseInt(txtNHp.getText());
        if (hp < 100 || hp > 500) {
            JOptionPane.showMessageDialog(this, "Los puntos de vida deben de estar entre 100 y 500", "Error 014", JOptionPane.ERROR_MESSAGE);
            validarAcción.regisAcción("Modificar vida personaje", false, "Salio MAL APROPOSITO");
            return;
        }
        int ataque = Integer.parseInt(txtNAtaque.getText());
        if (ataque < 10 || ataque > 100) {
            JOptionPane.showMessageDialog(this, "Los puntos de ataque deben de estar entre 10 y 100", "Error 015", JOptionPane.ERROR_MESSAGE);
            validarAcción.regisAcción("Modificar ataque personaje", false, "Salio MAL APROPOSITO");
            return;
        }
        int velocidad = Integer.parseInt(txtNVelocidad.getText());
        if (velocidad < 1 || velocidad > 10) {
            JOptionPane.showMessageDialog(this, "La agilidad debe de estar entre 1 a 10", "Error 016", JOptionPane.ERROR_MESSAGE);
            validarAcción.regisAcción("Modificar velocidad personaje", false, "Salio MAL APROPOSITO");
            return;
        }
        int agilidad = Integer.parseInt(txtNAgilidad.getText());
        if (agilidad < 1 || agilidad > 10) {
            JOptionPane.showMessageDialog(this, "La agilidad debe de estar entre 1 a 10", "Error 017", JOptionPane.ERROR_MESSAGE);
            validarAcción.regisAcción("Modificar agilidad personaje", false, "Salio MAL APROPOSITO");
            return;
        }
    }
```

```

        int defensa = Integer.parseInt (txtNDefensa.getText());
        if (defensa < 1 || defensa > 50) {
            JOptionPane.showMessageDialog (this, "La defensa debe de estar entre 1 a 50", "Error 018", JOptionPane.ERROR_MESSAGE);
            validarAccion.regisAccion ("Modificar defensa personaje", false, "Salio MAL APROPOSITO");
            return;
        }
        //Actualizacion de la matriz
        Personajes.personaje [pEncontrar][Personajes.ARMA] = txtNArma.getText();
        Personajes.personaje [pEncontrar][Personajes.HP] = String.valueOf (hp);
        Personajes.personaje [pEncontrar][Personajes.ATAQUE] = String.valueOf (ataque);
        Personajes.personaje [pEncontrar][Personajes.VELOCIDAD] = String.valueOf (velocidad);
        Personajes.personaje [pEncontrar][Personajes.AGILIDAD] = String.valueOf (agilidad);
        Personajes.personaje [pEncontrar][Personajes.DEFENSA] = String.valueOf (defensa);
    } catch (NumberFormatException e) {
        JOptionPane.showMessageDialog (this, "Los apartados que son numericos deben ser llenados con eso... pos numeros", "Error 019", JOptionPane.ERROR_MESSAGE);
        validarAccion.regisAccion ("Modificar al personaje", false, "Salio MAL APROPOSITO");
    }
}

//esto es bueno para limpiar las entradas
private void Limpiar(){
    txtBuscar.setText("");
    txtNArma.setText("");
    txtNHp.setText("");
    txtNAtaque.setText("");
    txtNVelocidad.setText("");
    txtNAgilidad.setText("");
    LbNombreAct.setText("");
    LbArmaAct.setText("");
    LbHpAct.setText("");
    LbAtaqueAct.setText("");
    LbVelocidadAct.setText("");
    LbAgilidadAct.setText("");
    LbDefensaAct.setText("");
    LbDefensaAct.setText("");
    pEncontrar = -1;
}

```

Clase de eliminar personaje.

En esta clase declaramos una ventana de 700×600 dónde encontraremos un área de texto, donde el usuario podrá escribir el nombre del personaje que querrá buscar para luego eliminarlo, hay un botón con la acción del llamado al método de eliminar personaje dicho método explicaremos a continuación, Y un botón dedicado a buscar al personaje dicho botón también al llamado a un método que se creó en la clase de personajes y como último el botón te desplegable que nos confirmará si queremos eliminar o no al personaje.

```
public eliminarPersonaje() {
    setTitle("Eliminar Personaje");
    setSize(700, 600);
    setLocationRelativeTo(null);
    setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);

    JPanel panel = new JPanel(new GridLayout(4, 1, 10, 20));

    JLabel tirulo = new JLabel("Eliminar Personaje", SwingConstants.CENTER);
    tirulo.setFont(new Font("Arial", Font.BOLD, 18));
    panel.add(tirulo);

    panel.add(new JLabel("Escribe ACA el ID o el nombre del personaje que deseas eliminar"));
    txtBuscar = new JTextField();
    panel.add(txtBuscar);

    JPanel pBoton = new JPanel(new FlowLayout());
    btBuscar = new JButton("Buscar");
    pBoton.add(btBuscar);
    panel.add(pBoton);

    LbPencontrado = new JLabel("Tamus buscando", SwingConstants.CENTER);
    LbPencontrado.setVisible(false);
    panel.add(LbPencontrado);

    btEliminar = new JButton("Eliminar Personaje");
    btEliminar.setVisible(false);
    panel.add(btEliminar);
    //añadimos todo eso de arriba al panel o como le dire de ahora en adelante, plantilla
    add(panel);

    btBuscar.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) {
            PersonajeEli(); //llamado al metodo que hace allí abajo
        }
    });

    btEliminar.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) {
            confirmarEliminar(); //otro metodo que hace mas mas abajo
        }
    });
}
```

Módulo personaje eliminado.

En este módulo hacemos llamado un método externo que se creó en una clase de personajes que se encarga de buscar al personaje en la matriz de personajes ya sea por nombre o por ID, tenemos una pequeña validación si en dado caso el área de texto está vacía para que nos indique que tenemos que escribir el nombre del personaje o el ID, si el personaje es encontrado nos mostrará el nombre del personaje que queramos eliminar de esta manera aparecerá el botón de eliminar y el nombre del personaje que agarramos eliminar.

```
//justo como prometi xd

public void PersonajeEli() {
    String Buscar = txtBuscar.getText().trim();
    Pecontrado = -1;
    LbPencontrado.setVisible(false);
    btEliminar.setVisible(false);

    if (Buscar.isEmpty()) {
        JOptionPane.showMessageDialog(this, "INGRESA EL NOMBRE O EL ID CTM", "Error 020", JOptionPane.ERROR_MESSAGE);
        validarAccion.regisAccion("buscar al personaje pa eliminarlo", false, "Salio MAL APROPOSITO");

        return;
    }
    Pecontrado = Personajes.buscarPer(Buscar);

    if (Pecontrado != -1) {
        String Necontrado = Personajes.personaje[Pecontrado][Personajes.NOMBRE];
        LbPencontrado.setText("El personaje a eliminar es: " + Necontrado);
        LbPencontrado.setVisible(true);
        btEliminar.setVisible(true);
        validarAccion.regisAccion("encontrar el personaje para eliminarlo", true, "Salio bien");

        return;
    }

    JOptionPane.showMessageDialog(this, "No se ha encontrado al personaje, f por ti", "Error 021", JOptionPane.ERROR_MESSAGE);
    validarAccion.regisAccion("Buscar al personaje para borrarlo", false, "Salio MAL");
}
```

Módulo confirmar eliminación y eliminar personaje.

En el módulo de confirmar eliminación lo que haremos es mostrar un botón desplegable con opciones de sí y no, esto gracias a “JOptionPane”, lo que nos permitirá eliminar el personaje de la matriz de personajes de igual manera si no queremos eliminarlo no se eliminará, en el módulo de eliminar personajes se creó inicialmente para cuando el personaje haya tenido algún combate pero tenemos que eliminarlo y queremos que aparezca en el historial de combates haciendo que técnicamente no existe el personaje eliminándolo de la matriz pero guardando el dato que si luchó.

```
private void confirmarEliminar() {
    if (Pecontrado != -1) {
        String nEliminar = Personajes.personaje [Pecontrado][Personajes.NOMBRE];
        int Confirmar = JOptionPane.showConfirmDialog (
            this,
            "¿Estas seguro de eliminar a: " + nEliminar + "?",
            "Confirmar Eliminacion",
            JOptionPane.YES_NO_OPTION);
        if (Confirmar == JOptionPane.YES_OPTION) {
            EliminarPersonaje(Pecontrado);
            JOptionPane.showMessageDialog (this, nEliminar + " se ha eliminado del roster de personajes");
            dispose();
            validarAccion.regisAccion ("Borrar el personaje", true, "Salio bien");
        } else {
            validarAccion.regisAccion ("Eliminar el personaje", true, "Salio... y ya");
        }
    }
}

private void EliminarPersonaje(int in) {
    //Obtenemos el historial de peleas... si es que hay
    String historial = Personajes.personaje [in][Personajes.VICDER];
    boolean GP = historial != null && !historial.isEmpty() && !historial.equals("0-0");

    if (GP) {
        //si el personaje no se ha peleado con nadie, vamos a decir que no hay nada, como el meme de avatar
        Personajes.personaje [in][Personajes.NOMBRE] = "Ya no esta";
        Personajes.personaje [in][Personajes.ARMA] = "No hay";
        Personajes.personaje [in][Personajes.HP] = "No hay";
        Personajes.personaje [in][Personajes.ATAQUE] = "No hay";
        Personajes.personaje [in][Personajes.VELOCIDAD] = "No hay";
        Personajes.personaje [in][Personajes.AGILIDAD] = "No hay";
        Personajes.personaje [in][Personajes.DEFENSA] = "No hay";
    } else {
        // si el personaje no se ha sacado la chucha con alguien mas lo eliminamos totalmente y ya
        for (int i = in; i < Personajes.CantPersonajes - 1; i++) {
            Personajes.personaje [i] = Personajes.personaje [i + 1];
        }
        Personajes.personaje [Personajes.CantPersonajes - 1] = new String[9];
        Personajes.CantPersonajes--;
    }
}
```

Clase y módulo de ver personaje y módulo de tabla de personajes.

En este módulo tenemos una ventana de 800×800 con una tabla de desplegable con los datos del personaje que son el ID, nombre, arma, puntos de vida, ataque, velocidad, agilidad, defensa y el historial de victorias y derrotas, seguido tenemos el módulo de tabla de personajes donde se llenará a medida que vayamos registrando personajes de igual manera buscando en matriz de personajes los que ya estén existentes.

```
/*
public class verPersonaje extends JFrame {
    private JTable tablaPerson;
    private JScrollPane panelS;

    public verPersonaje(){
        //titulo de la ventana
        setTitle("Personajes Registrados");
        setSize(800, 800);
        setLocationRelativeTo(null);
        setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);

        //nombres de las columnas
        String[] ncolumns = {"ID", "Nombre", "Arma", "Hp", "Ataque", "Velocidad", "Agilidad", "Defensa", "Historial"};

        //para arreglar las filas usamos defaultable Model, esto lo encontre en satackoverflow :3
        DefaultTableModel modTabla = new DefaultTableModel(ncolumns, 0);
        tablaPerson = new JTable(modTabla);

        //llenamos la tabla con los personajes del metodo que esta alli abajo
        TablaPersonajes(modTabla);

        panelS = new JScrollPane(tablaPerson);
        add(panelS, BorderLayout.CENTER );
    }

    private void TablaPersonajes(DefaultTableModel mod){
        //vamos a limpiar las filas que ya esten, cuadro se agreguen o borrar personajes
        mod.setRowCount(0);

        //un hermoso ciclo for para recorrer la matriz de personajes y agregar cada fila a la tabla de personajes
        for(int i=0; i < Personajes.CantPersonajes ; i++){
            // Creamos un arreglo con los datos de cada personaje en la posicion de i
            Object[] filas = new Object[Personajes.personaje [i].length];
            for(int j = 0; j < Personajes.personaje [i].length; j++){
                filas[j] = Personajes.personaje [i][j];
            }
            mod.addRow(filas);
        }
    }
}
```

Clase simular combates

En esta clase tenemos declarada lo que son dos botones plegables con la lista de personajes ya registrados dado que en el combate sólo pueden luchar dos personajes tenemos el botón de iniciar lucha y el Scrollpane para el texto del combate.

```
/*
public class SimPelea extends JFrame {
    //esto es para buscar los personajes de la matriz

    private JComboBox<String> cblLuchador1;
    private JComboBox<String> cblLuchador2;

    private JButton btInilucha;
    private JTextArea bitacoraArea;
    private JScrollPane sp;

    private Random random = new Random();
}
```

Módulo simular combates

Acá de igual manera tenemos una ventana desplegable de 700×600 con los botones desplegables de los personajes el botón de buscar y el área de texto donde se mostrará las acciones en combate, el botón de iniciar combate hace llamado al método del mismo nombre.

```
public SimPelea() {
    //esto ya se lo saben
    setTitle("Simulador de Combates");
    setSize(700, 600);
    setLocationRelativeTo(null);
    setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
    //otro chapus para que el texto si se vea...
    JPanel PanelSuperior = new JPanel(new FlowLayout(FlowLayout.CENTER, 10, 10));
    JPanel panelPrincipal = new JPanel(new BorderLayout(10, 10));

    JLabel titulo = new JLabel("Elige dos personajes para LUCHAR", SwingConstants.CENTER);
    titulo.setFont(new Font("Arial", Font.BOLD, 18));

    cblLuchador1 = new JComboBox<>();
    cblLuchador2 = new JComboBox<>();
    CargarLuchador(); //un metodo que alli abajo esta xd

    PanelSuperior.add(titulo);
    PanelSuperior.add(new JLabel("Elige al primer luchador"));
    PanelSuperior.add(cblLuchador1);
    PanelSuperior.add(new JLabel("Ahora Elige al segundo luchador"));
    PanelSuperior.add(cblLuchador2);

    btInilucha = new JButton("MADRAZOS");
    PanelSuperior.add(btInilucha);

    //otro chapus para evitar que el boton crashee el programa
    bitacoraArea = new JTextArea();
    bitacoraArea.setEditable(false);
    JScrollPane sp = new JScrollPane(bitacoraArea);

    //agregar los componentes a la region del panel principal
    panelPrincipal.add(PanelSuperior, BorderLayout.NORTH); //el panel con el titulo, el botón y el seleccionador
    panelPrincipal.add(sp, BorderLayout.CENTER);
    add(panelPrincipal);

    btInilucha.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) {
            IniciarCombate(); //el metodo que alli ta MAS abajo
        }
    });
}
```

Módulo cargar personajes

Este método queda implícito a los botones desplegables de los personajes este método se encargará de hacer llamado de ambos personajes de la matriz y tratarlos como objetos para el combate.

```
private void CargarLuchador() {
    for (int i = 0; i < Personajes.CantPersonajes; i++) {
        String nombre = Personajes.personaje[i][Personajes.NOMBRE];
        cbLuchador1.addItem(nombre);
        cbLuchador2.addItem(nombre);
    }
}
```

Módulo iniciar combate

Este módulo se encargará de seleccionar los personajes que el usuario haya elegido hay varias validaciones si los personajes son iguales si no hay ningún personaje si ambos no existen o si en dado caso ninguno de los dos tenga ningún punto de vida seguido aparecerá en la pantalla de texto la bienvenida al combate e iniciará el hilo del personaje 1 y del personaje 2.

```
private void IniciarCombate(){
    String Luchador1 = (String) cbLuchador1.getSelectedItem();
    String Luchador2 = (String) cbLuchador2.getSelectedItem();

    //otro chapas...
    if (Personajes.CantPersonajes < 2){
        JOptionPane.showMessageDialog(this, "NECESITAS ALMENOS DOS Personajes para el combate", "Error 022", JOptionPane.ERROR_MESSAGE);
        validarAccion.regisAccion("Madrazos entre personajes", false, "Salio MAL APROPOSITO");
        return;
    }

    if (Luchador1 == null || Luchador2 == null){
        JOptionPane.showMessageDialog(this, "Selecciona DOS Personajes para el combate", "Error 023", JOptionPane.ERROR_MESSAGE);
        validarAccion.regisAccion("Madrazos entre personajes", false, "Salio MAL APROPOSITO");
    }

    if (Luchador1.equals(Luchador2)){
        JOptionPane.showMessageDialog(this, "Un personaje no puede pelear solito, seria muy raro eso", "Error 024", JOptionPane.ERROR_MESSAGE);
        validarAccion.regisAccion("Madrazos entre personajes", false, "Salio MAL APROPOSITO");
    }

    int indice1 = Personajes.buscarPer(Luchador1);
    int indice2 = Personajes.buscarPer(Luchador2);

    if (indice1 == -1 || indice2 == -1){
        JOptionPane.showMessageDialog(this, "Algun personaje, no existe, ¿Qual? no se la verdad", "Error 025", JOptionPane.ERROR_MESSAGE);
        validarAccion.regisAccion("Madrazos entre personajes", false, "Salio MAL APROPOSITO");
        return;
    }

    //los luchadores
    String[] L1 = Personajes.personaje [indice1];
    String[] L2 = Personajes.personaje [indice2];

    //los puntos de vida de cada uno
    int Hp1 = Integer.parseInt (L1[Personajes.HP]);
    int Hp2 = Integer.parseInt (L2[Personajes.HP]);

    if (Hp1 <= 0 || Hp2 <= 0){
        JOptionPane.showMessageDialog(this, "Alguno de los personajes ya se murio xD", "Error 026", JOptionPane.ERROR_MESSAGE);
        validarAccion.regisAccion("Madrazos entre personajes", false, "Salio MAL");
        return;
    }
```

```

bitacoraArea.setText("BIENVENIDOS AL COMBATE, HOY SE ENFRENTARÁ: " + Luchador1 + " CONTRA " + Luchador2 + " SIGAN VIENDO");
btIniLucha.setEnabled(false);

String[] P1 = L1.clone();
String[] P2 = L2.clone();

//la parte de los hilos... me wa a morir
Thread hilo1 = new Thread(() -> Combate(P1, P2, bitacoraArea));
Thread hilo2 = new Thread(() -> Combate(P1, P2, bitacoraArea));
hilo1.start();
hilo2.start();
}

```

Modulo combate

En este módulo con un ciclo While, hacemos conteo de turnos por cada ataque realizado hacemos el cálculo del daño que es el ataque menos la defensa tenemos una condición si en dado caso el personaje logra evitar el ataque gracias a la librería random, se mostrará en pantalla las acciones como los ataques y cuánta vida le queda al defensor, para cuando termine el combate se mostrará en pantalla el ganador el número de turnos que fueron requeridos para que el combate acabe cuando esto pase el botón de luchar se habilitará.

```

private void Combate(String[] atacante, String[] defensor, JTextArea bitacora) {
    String Natacante = atacante[Personajes.NOMBRE];
    String Ndefensor = defensor[Personajes.NOMBRE];
    int turnos = 0;
    while (Integer.parseInt(atacante[Personajes.HP]) > 0 && Integer.parseInt(defensor[Personajes.HP]) > 0) {
        try {
            turnos++; // cada ataque cuenta como turno
            int dañoBase = Integer.parseInt(atacante[Personajes.ATAQUE]);
            int agiliDef = Integer.parseInt(defensor[Personajes.AGILIDAD]);
            int defDef = Integer.parseInt(defensor[Personajes.DEFENSA]);

            //la probabilidad de esquivar
            boolean esquivar = random.nextInt(10) < agiliDef;

            if (esquivar) {
                int t = turnos;
                SwingUtilities.invokeLater () -> bitacora.append("\nTurno" + t + ":" + Natacante + " HA ATACADO, PERO \n" + Ndefensor + " HA ESQUIVADO EL ATAQUE ");
            } else {
                int dañoFinal = dañoBase - defDef;
                if (dañoFinal < 0) {
                    dañoFinal = 0;
                }
                int hpActualDef = Integer.parseInt(defensor[Personajes.HP]);
                hpActualDef -= dañoFinal;
                defensor[Personajes.HP] = String.valueOf(hpActualDef);

                //Chapus para evitar un error
                int t = turnos;
                final int dañoF = dañoFinal;
                final int HpActDef = hpActualDef;
                // Es la linea mas larga que he echo... dios... me habre ganado un recor guinen' +
                SwingUtilities.invokeLater () -> bitacora.append("\nTurno" + t + ":" + Natacante + " HA ECHO UN ATAQUE " + Ndefensor + " HA SUFRIDO \n" + dañoF + " PUNTOS DE DAÑO, LOS DE VIDA RESTANTES DE " + Ndefensor + " ES: " + HpActDef);
            }
            int veloAtacante = Integer.parseInt(atacante[Personajes.VELOCIDAD]);
            Thread.sleep(1000 / veloAtacante);
        } catch (InterruptedException e) {
            Thread.currentThread().interrupt();
        }
    }
}

```

```

        int turnosF = turnos;
        //vamos arreglar esto..
        SwingUtilities.invokeLater (() -> {
            String ganador = "";
            String perdedor = "";

            if (Integer.parseInt (atacante[Personajes.HP]) > 0) {
                ganador = Natacantante;
                perdedor = Ndefensor;
            } else if (Integer.parseInt (defensor[Personajes.HP]) > 0) {
                ganador = Ndefensor;
                perdedor = Natacantante;
            }
            if (ganador.isEmpty()) {
                bitacora.append("\n" + ganador + " Ha ganado el combate en \n" + turnosF + " turnos \n");
                actualHistorial(ganador, perdedor, turnosF);
                validarAcción.registraAccion ("madrazos entre personajes", true, "Salio bien");
            } else {
                bitacora.append("TENEMOS UN EMPATE A LOS: " + turnosF + " TURNO \n");
            }
            btIniLucha.setEnabled(true);
        });
    }
}

```

Módulo actualizar historial

Este módulo se encarga de actualizar el historial de combate de los personajes después de un combate empezamos declarando los formatos de fecha y hora siguiendo el contador de Victoria Sierra rotas empezando en cero de allí con el bloque try catch, hacemos llamadas de una matriz, de datos que está en el siguiente módulo a explicar, sea la victoria derrota mostrará quién ganó quién perdió y la cantidad de turnos que se llevó a cabo junto con la hora y fecha en que se llevó a cabo el combate.

```

private void actualHistorial(String ganador, String perdedor, int turnos) {
    //la hora y fecha
    SimpleDateFormat fechafoma = new SimpleDateFormat("yyyy-MM-dd");
    SimpleDateFormat Horafoma = new SimpleDateFormat("HH:mm:ss");

    Date ahora = new Date();
    String fecha = fechafoma.format(ahora);
    String hora = Horafoma.format(ahora);

    //otro chapus
    int vicAct = 0;
    int derrAct = 0;

    int inGanar = Personajes.buscarPer (ganador);
    int inPerder = Personajes.buscarPer (perdedor);

    String HistGanar = Personajes.personaje [inGanar][Personajes.VICDERR];
    String HistPer = Personajes.personaje [inPerder][Personajes.VICDERR];

    //sin esto el programa falla... y tuve que rehacer esta basura
    try {
        String[] datos = HistGanar.split("\\|");
        vicAct = Integer.parseInt (datos[0].trim());
        derrAct = Integer.parseInt (datos[1].trim());
    } catch (Exception e) {
        try {
            String[] datos = HistGanar.split("-");
            vicAct = Integer.parseInt (datos[0].trim());
            derrAct = Integer.parseInt (datos[1].trim());
        } catch (Exception e2) {
            vicAct = 0;
            derrAct = 0;
        }
    }
    vicAct++;
    Personajes.personaje [inGanar][Personajes.VICDERR] = vicAct + " | " + derrAct + " | " + turnos + " | " + fecha + " | " + hora;
    //reseteamos el contenido de perdidas y ganancias
    vicAct = 0;
    derrAct = 0;

    try {
        String[] datos = HistPer.split("\\|");
        vicAct = Integer.parseInt (datos[0].trim());
        derrAct = Integer.parseInt (datos[1].toLowerCase());
    } catch (Exception e) {
        try {
            String[] datos = HistPer.split("-");
            vicAct = Integer.parseInt (datos[0].trim());
            derrAct = Integer.parseInt (datos[1].trim());
        } catch (Exception e2) {
            vicAct = 0;
            derrAct = 0;
        }
    }
    derrAct++;
    Personajes.personaje [inPerder][Personajes.VICDERR] = vicAct + " | " + derrAct + " | " + turnos + " | " + fecha + " | " + hora;
    //reseteamos el contenido de perdidas y ganancias
    vicAct = 0;
    derrAct = 0;
}

```

Clase de historial combate

En esta clase declaramos el área de texto junto con la fecha y hora, el área de textos se encargará de mostrar los datos de los combates junto con los tenedores y la fecha.

```
^/
public class historialCombate extends JFrame {

    private JTextArea Harea;
    private JLabel LbFecha;
```

Módulo historial de combates.

En este módulo tenemos la pantalla de desplegable que contendrá el título el área de texto donde se mostraran la información de los combates y un botón dedicado para actualizar el historial de combates esto para cuando no cerremos la ventana y estemos realizando combate entre personajes, esto con su debida acción a la llamada al método de cargar historial

```
public historialCombate() {
    //ya se la saben... el titulo de la ventana y el tamaño de la misma... quiero dormir...
    setTitle("Historial de Combates");
    setSize(600, 700);
    setLocationRelativeTo(null);
    setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);

    JPanel panelPrincipal = new JPanel(new BorderLayout(10, 10));

    JLabel titulo = new JLabel("Este es el historial de combates realizados");
    titulo.setFont(new Font("Arial", Font.BOLD, 18));
    panelPrincipal.add(titulo, BorderLayout.NORTH); //esto va a estar ARRIBA

    //agregamos el panel a la fecha y hora
    JPanel panelCentro = new JPanel(new BorderLayout());

    //Inciamos la etiqueta de la fecha
    LbFecha = new JLabel("", SwingConstants.CENTER);
    panelCentro.add(LbFecha, BorderLayout.NORTH);

    Harea = new JTextArea();
    Harea.setEditable(false);
    JScrollPane panelAbajo = new JScrollPane(Harea);
    panelCentro.add(panelAbajo, BorderLayout.CENTER);
    panelPrincipal.add(panelCentro, BorderLayout.CENTER); //aca la kgue xd

    //el boton para que el historial se actualice
    JButton btActualizar = new JButton("Actualizar historial");
    panelPrincipal.add(btActualizar, BorderLayout.SOUTH);
    add(panelPrincipal);

    Cargarhist(); //un llamado de un metodo que esta alli abajo

    btActualizar.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) {
            Cargarhist(); //la misma cosa que dije arriba.
        }
    });
}
```

Módulo cargar historial.

En este método o módulo declaramos el formato de hora y fecha seguido declaramos la la matriz secundaria donde van a mostrarse los datos de los combates, de apareciendo en pantalla el personaje sí las victorias las derrotas el número de turnos que tuvo su último combate todo con su última validación al método de validar acciones.

```
private void Cargarhist() {
    SimpleDateFormat fechahora = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
    Date Factual = new Date();
    String HFformat = fechahora.format(Factual);
    LFecha.setText("Ultima actualizacion: " + HFformat);
    StringBuilder Hist = new StringBuilder();
    Hist.append("Historial de Victorias y Derrotas \n");
    Hist.append("----- \n"); //es innecesario, pero se ve mas bonito asi
    if (Personajes.CantPersonajes == 0) {
        JOptionPane.showMessageDialog(this, "No hay personajes registrados, como para que se madreen", "Error 027", JOptionPane.ERROR_MESSAGE);
        validarAcción.regisAcción("ver historial de madrazos entre personajes", false, "Salio MAL APROPOSITO");
    } else {
        for (int i = 0; i < Personajes.CantPersonajes; i++) {
            String nombre = Personajes.personaje[i][Personajes.NOMBRE];
            String vicDerr = Personajes.personaje[i][Personajes.VICDEERR];

            String[] datos = new String[5];

            if (vicDerr.contains(":")) {
                datos = vicDerr.split("\\:\\");
            } else {
                datos[0] = "0";
                datos[1] = "0";
                datos[2] = "N/A";
                datos[3] = "...";
                datos[4] = "...";
            }
            //para que se vea mejor el hisotial
            String victorias = datos.length > 0 ? datos[0] : "0";
            String derrotas = datos.length > 1 ? datos[1] : "0";
            String turnos = datos.length > 2 ? datos[2] : "N/A";
            String fecha = datos.length > 3 ? datos[3] : "...";
            String hora = datos.length > 4 ? datos[4] : "...";

            Hist.append("Personaje: ").append(nombre).append("\n"); //salto de linea
            Hist.append("Victorias: ").append(victorias).append("Derrotas: ").append(derrotas).append("\n");
            Hist.append("Ultimo Combate: ").append(turnos).append(" turnos \n");
            Hist.append("fecha: ").append(fecha).append(" | Hora: ").append(hora).append("\n");
            Hist.append("----- \n");
            validarAcción.regisAcción("ver historial de madrazos entre personajes", true, "Salio bien");
        }
    }
    Harea.setText(Hist.toString());
}
```

Clase buscar personajes por nombre

En esta clase declaramos una ventana de 500× 600, junto al área de texto donde vamos a escribir el nombre de personaje que queremos buscar junto a eso el botón para buscar el personaje junto al método asociado al botón.

```
public class BuscarPer extends JFrame {  
  
    private JTextField txtBuscar;  
    private JLabel LbEncontrar;  
    private JTextArea Area;  
  
    public BuscarPer() {  
        //ya no quiero explicar esta madre  
        setTitle("Buscar el personaje por su nombre");  
        setSize(500, 600);  
        setLocationRelativeTo(null);  
        setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);  
  
        JPanel PanelPrincipal = new JPanel(new BorderLayout(10, 10));  
  
        JLabel titulo = new JLabel("Buscar Personaje", SwingConstants.CENTER);  
        titulo.setFont(new Font("Arial", Font.BOLD, 19));  
        PanelPrincipal.add(titulo, BorderLayout.NORTH);  
  
        //el caso para buscar  
        JPanel panelBuscar = new JPanel(new GridLayout(1, 2, 5, 5));  
        txtBuscar = new JTextField();  
        JButton btBuscar = new JButton("Buscar Personajes");  
        panelBuscar.add(new JLabel("Escribe del Personaje: "));  
        panelBuscar.add(txtBuscar);  
  
        JPanel PanelCentro = new JPanel(new BorderLayout());  
        LbEncontrar = new JLabel("Informacion del Personaje es: ", SwingUtilities.CENTER);  
        LbEncontrar.setFont(new Font("Arial", Font.BOLD, 14));  
        PanelCentro.add(LbEncontrar, BorderLayout.NORTH);  
  
        Area = new JTextArea();  
        Area.setEditable(false);  
        PanelCentro.add(Area, BorderLayout.CENTER);  
  
        PanelPrincipal.add(panelBuscar, BorderLayout.CENTER);  
        PanelPrincipal.add(btBuscar, BorderLayout.SOUTH);  
        add(PanelPrincipal, BorderLayout.NORTH);  
        add(PanelCentro, BorderLayout.CENTER);  
  
        btBuscar.addActionListener(new ActionListener() {  
            @Override  
            public void actionPerformed(ActionEvent e) {  
                buscarPersonajes(); //ya te ls sabes no... un metodo alli abajo  
            }  
        });  
    }  
}
```

Modulo buscar personajes

En este módulo hacemos un llamado al método que hicimos una clase de personajes para buscar al personaje por nombre o por ID, si se encuentra el personaje con sus atributos, en el área de texto designada se mostrará los datos del personaje, su ataque, su velocidad, su arma, sus puntos de defensa, su agilidad y sus victorias y derrotas, seguido eso se tiene una pequeña matriz creada con datos que contendrán las victoria la derrota los turnos y la abrí la fecha para luego mostrar en pantalla también dichos datos, junto al llamado de las clases de validar acciones.

```
private void buscarPersonajes() {
    String Nbuscar = txtBuscar.getText().trim();
    if (Nbuscar.isEmpty()) {
        JOptionPane.showMessageDialog(this, "Mira... pon un nombre si?", "Error 028", JOptionPane.ERROR_MESSAGE);
        validarAccion.regisAccion("buscar personajes", false, "Salio MAL APROPOSITO");
    }
    int indicePer = Personajes.buscarPer(Nbuscar); //sabia que seria util hacer el llamado
    if (indicePer != -1) {
        String[] personaje = Personajes.personaje [indicePer];
        String nombre = personaje[Personajes.NOMBRE];
        String arma = personaje[Personajes.ARMA];
        String ataque = personaje[Personajes.ATAQUE];
        String Hp = personaje[Personajes.HP];
        String velocidad = personaje[Personajes.VELOCIDAD];
        String vicDerr = personaje[Personajes.VICDERR];
        String defensa = personaje[Personajes.DEFENSA];

        StringBuilder resultado = new StringBuilder();
        resultado.append("Hemos encontrado a este Buey: \n");
        resultado.append("El nombre es: ").append(nombre).append("\n");
        resultado.append("El Arma del personaje es: ").append(arma).append("\n");
        resultado.append("Los puntos de vida son: ").append(Hp).append("\n");
        resultado.append("Los puntos de ataque son: ").append(ataque).append("\n");
        resultado.append("La velocidad es: ").append(velocidad).append("\n");
        resultado.append("Los puntos de defensa son: ").append(defensa).append("\n");

        //un chapus de ayuda
        String[] datos = new String[5];
        if (vicDerr != null && vicDerr.contains("|")) {
            datos = vicDerr.split("\\|");
        } else {
            datos[0] = "0";
            datos[1] = "0";
            datos[2] = "N/A";
            datos[3] = "...";
            datos[4] = "...";
        }
        String victorias = datos.length > 0 ? datos[0].trim() : "0";
        String derrotas = datos.length > 1 ? datos[1].trim() : "0";
        String turnos = datos.length > 2 ? datos[2].trim() : "N/A";
        String fecha = datos.length > 3 ? datos[3].trim() : "...";
        String hora = datos.length > 4 ? datos[4].trim() : "...";

        resultado.append("Historial: ").append("Victorias: ").append(victorias).append(", Derrotas: ").append(derrotas).append("\n");
        resultado.append("Ultimo Combate: ").append(turnos).append(", ").append("Fecha: ").append(fecha).append(", Hora: ").append(hora).append("\n");

        Area.setText(resultado.toString());
        validarAccion.regisAccion("Buscar el personaje", true, "Salio bien :p");
    } else {
        JOptionPane.showMessageDialog(this, "No hemos encontrado al personaje... no pusiste nada verdad?", "Error 029", JOptionPane.ERROR_MESSAGE);
        validarAccion.regisAccion("Buscar personajes", false, "Salio MAL");
    }
}
```

Clase Gestor de archivos

En esta clase declaramos una variable de tipo string que corresponde al nombre del archivo

```
/*
public class gestorArchivo {

    private static final String NombreArchi = "Historial_personajes.txt";

    //ya me canseeeeeee, el jodido cafe ya no me hace efecto
```

Módulo de guardar historial de personajes.

En este método enfocamos el nombre del archivo en el cual buscamos también en la matriz de personajes todos los personajes que hayamos ingresado, de esta manera imprimir en el archivo todos los personajes que se hayan registrado de esta manera siendo una forma de memoria en archivo de texto

```
//ya me canseeeeeee, el jodido cafe ya no me hace efecto
public static void GuardarPersonaje () {
    try (PrintWriter escribir = new PrintWriter(new FileWriter(NombreArchi))){
        escribir.println(Personajes.CantPersonajes);
        for (int i = 0; i < Personajes.CantPersonajes; i++){
            escribir.println(String.join ("|", Personajes.personaje [i]));
        }
        JOptionPane.showMessageDialog (null, "Se ha hecho guardado los Datos :D");
        validarAccion.regisAccion ("guardar datos del personaje", true, "Salio bien");
    } catch (IOException e){
        JOptionPane.showMessageDialog (null, "No se pudo guardar, porque yo que se man D:" + e.getMessage(), "Error 030", JOptionPane.ERROR_MESSAGE);
        validarAccion.regisAccion ("guardar datos del personaje", false, " datos de " + Personajes.CantPersonajes + "Salio MAL");
    }
}
```

Modulo de cargar historial de personajes

Este módulo se encarga principalmente de buscar al archivo de texto, para luego leer sobre el archivo y luego cargar esos datos al programa diciendo que el archivo no existe lo creara.

```
public static void CargarPersonajes () {
    //para buscar el archivo y buscar cada dato xd
    File archivo = new File(NombreArchi);
    if (!archivo.exists()){
        JOptionPane.showMessageDialog (null, "No hemos encontrado nada... Estamos Jodidos", "Error 031", JOptionPane.WARNING_MESSAGE);
        validarAccion.regisAccion ("Encontrar el archivo", false, "Salio MAL");
        return;
    }
    try (BufferedReader lector = new BufferedReader(new FileReader(archivo))){
        String Lineas;
        if ((Lineas = lector.readLine()) != null){
            Personajes.CantPersonajes = Integer.parseInt (Lineas.trim());
        }
        for (int i = 0; i < Personajes.CantPersonajes; i++){
            if ((Lineas = lector.readLine()) != null){
                String[] datos = Lineas.split("\\|");
                if (datos.length > Personajes.NUMCAMPSPERSONAJE){
                    for (int j = 0; j < Personajes.NUMCAMPSPERSONAJE; j++){
                        Personajes.personaje [i][j] = datos[j].trim();
                    }
                }
            }
        }
        JOptionPane.showMessageDialog (null, "Se ha Cargado el sistema :D");
        validarAccion.regisAccion ("guardar datos en archivo", true, "Salio bien");
    } catch (IOException | NumberFormatException e){
        JOptionPane.showMessageDialog (null, "No se pudo cargar el archivo.... SIP NOS JODIMOS" + e.getMessage(), "Error 032", JOptionPane.ERROR_MESSAGE);
        validarAccion.regisAccion ("guardar datos en archivo", false, "Salio MAL");
    }
}
```

Módulo de borrar datos.

El módulo de borrar datos se encarga de buscar de igual manera el archivo para luego mandar un mensaje en pantalla si se quiere eliminar o no el archivo si en dado caso se escoge la opción de si el archivo se borrará lo contrario no se va a borrar.

```
public static void BorrarDatos () {
    File archivo = new File(NombreArchi);
    if (archivo.exists()){
        int Confirm = JOptionPane.showConfirmDialog(null, "Se va a eliminar todos los datos...¿ REALMENTE ESTAS SEGURO?", "Confirmar Limpieza", JOptionPane.YES_NO_OPTION);
        if (Confirm == JOptionPane.YES_OPTION){
            if (archivo.delete()){
                Personajes.CantPersonajes = 0; //reseteamos el contador de personajes
                JOptionPane.showMessageDialog(null, "TODO fue Borrado... yay..");
                validarAccion.regisAccion("borrar datos del archivo", true, "Salio bien");

            } else {
                JOptionPane.showMessageDialog(null, "Algo salio mal.. aun..o bueno ya sabes ", "Error 033", JOptionPane.ERROR_MESSAGE);
                validarAccion.regisAccion("borrar datos del historial", false, "Salio MAL");
            }
        } else {
            JOptionPane.showMessageDialog(null, "No hay nada que borrar aun...", "Error 034", JOptionPane.WARNING_MESSAGE);
            validarAccion.regisAccion("Borrar datos del hisotial", false, "Salio MAL");
        }
    }
}
```

Clase validar acción

En esta clase declaramos el nombre del archivo junto el formato de hora y fecha.

```
public class validarAccion {
    //una mausque herramienta que nos ayudara mas tarde

    private static final String NombreBit = "Bitacora_Acciones.txt"; //un archivo de datos
    private static final SimpleDateFormat FECHA = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
```

Modulo Registrar acción.

En este módulo declaramos una variable que se encargará del estado de la tarea, una que represente al formato de fecha, junto con impresión en pantalla con el comando de “tiempo acción y mensaje”, este módulo se hace llamado en todas las acciones que el usuario haga, tanto salgan bien, o que salgan mal estas acciones, estas se imprimirán en la consolada del IDE de NetBeans.

```
public static void regisAccion (String accion, boolean estado, String mensaje) {
    //ibtenemos la fecha
    String tiempo = FECHA .format(new Date());

    // ahora veremos male sal o sale bien
    String Estado = estado ? "Salio bien" : "Salio mal";

    //creamos un registro
    String resgitrar = String.format ("%s [%s] - %s: %s", tiempo, Estado, accion, mensaje);

    //lo podemos en consola
    System.out.println(resgitrar);

    //lo mandamos a un txt xddd por si las moscas
    try (PrintWriter escribir = new PrintWriter(new FileWriter(NombreBit , true))) {
        escribir.println(resgitrar);
    } catch (IOException e) {
        JOptionPane.showMessageDialog(null, "Algo Malo sal... no se que chucha fue", "Error 030", JOptionPane.ERROR_MESSAGE);
        validarAccion.regisAccion ("guardar datos en la bitacora", false, "Salio mal");
    }
}
```

Módulo mostrar borrado de bitácora.

Este módulo se encarga directamente de buscar el nombre del archivo de la bitácora de acciones dicho archivo que es de texto se encontrara para luego mandar un mensaje de confirmación si se quiere eliminar o no el archivo si se elimina mandar un mensaje de qué ha sido borrar de lo contrario no se borrara.

```
//ahora veremos la bitacora en consola
public static void MostrarBit () {
    File archivo = new File(NombreBit );
    //ojala que no se vea feo despues del archivo de musica o si no me mato, se va a ver TODAS LAS LLAMADAS QUE HE HECHO DE ESTE METODO, no este pero ya sabes... enseria estas leyendo esto...
    if (archivo.exists()){
        JOptionPane.showMessageDialog(null, "La bicatara no exixte xd, ojala no la hayas borrado meno", "Error 031", JOptionPane.ERROR_MESSAGE);
        validarAccion.regisAccion ("Buscar bitacora", false, "Salio mal APROPOSITO");
        return;
    }
    //para que se borre eso
    int confirm = JOptionPane.showConfirmDialog(null, "¿Se va a borrar la bicatara... estas seguro?", "Confirmar boorador de la botacora", JOptionPane.YES_NO_OPTION);
    if (confirm == JOptionPane.YES_OPTION){
        if (archivo.delete()){
            JOptionPane.showMessageDialog(null, "Todas las acciones se borraron :D", "salio bien", JOptionPane.INFORMATION_MESSAGE);
            validarAccion.regisAccion ("Borrar la bitacora", true, "Salio bien");
        } else {
            JOptionPane.showMessageDialog(null, "No borramos la bitacora, porque? no me pregantes wn tu lo decidiste", "Error 032", JOptionPane.ERROR_MESSAGE);
            validarAccion.regisAccion ("borrar la bitacora", true, "Salio y ya");
        }
    }
}
```

Clase y módulo de datos del estudiante

Éste es el último módulo por explicar, en este módulo se imprimirá en una ventana se mostrarán los datos del estudiante o sea yo, junto a un botón dedicado a salir de la ventana junto a eso se mostrará el llamado del método de validar acciones.

```
public class YOOOOO extends JFrame {  
  
    public YOOOOO(){  
        //ya no quiero explicar esto  
        setTitle("SOY YOOOOOOOOOO");  
        setSize(500, 300);  
        setLocationRelativeTo(null);  
        setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);  
  
        JPanel panelPrincipal = new JPanel(new GridLayout(6, 1, 10, 10));  
        //el titulo  
        JLabel titulo = new JLabel("MIS DATOS :3", SwingConstants.CENTER);  
        titulo.setFont(new Font("Papyrus", Font.BOLD, 25));  
        panelPrincipal.add(titulo);  
  
        panelPrincipal.add(new JLabel("Mi nombre es: Guillermo Enrique Marroquin Morán", SwingConstants.CENTER));  
        panelPrincipal.add(new JLabel("Mi Carnet es 202103527", SwingConstants.CENTER));  
        panelPrincipal.add(new JLabel("Este programa tiene kilos y kilos de copy :3, mas la musica de undertale xdddddd"));  
  
        //boton para que se cierre esta madre  
        JButton btcerrar = new JButton("salir");  
        btcerrar.addActionListener(e -> dispose());  
        panelPrincipal.add(btcerrar);  
  
        add(panelPrincipal);  
        validarAccion.regisAccion("Ver mis datos", true, "Salio bien, como demonios debe de salir mal esto?");  
    }  
}
```