Manual Técnico

1. Estructura del Proyecto

- Archivo Principal: Este únicamente contiene la ventana que se ejecuta cuando el programa inicia en donde se muestran los de mas botones.

Practica2.java

```java
package practica2;

public class Practica2 {
    public static void main(String[] args) {
        new Ventana();
    }
}
```

- Clase Agregar Personaje: En esta clase se agrega al personaje con las validaciones correspondientes y con los parámetros establecidos en las instrucciones como por ejemplo que el HP debe de ser de 100 a 500.

```java
public AgregarPersonaje(JFrame parent) {
    super(parent, "Agregar Personaje", true);
    setSize(400, 400);
    setLayout(new GridLayout(8, 2));
    setLocationRelativeTo(parent);

    txtNombre = new JTextField();
    txtArma = new JTextField();
    txtHP = new JTextField();
    txtAtaque = new JTextField();
    txtVelocidad = new JTextField();
    txtAgilidad = new JTextField();
    txtDefensa = new JTextField();

    add(new JLabel("Nombre:"));
    add(txtNombre);
    add(new JLabel("Arma:"));
    add(txtArma);
    add(new JLabel("HP (100-500):"));
    add(txtHP);
    add(new JLabel("Ataque (10-100):"));
    add(txtAtaque);
    add(new JLabel("Velocidad (1-10):"));
    add(txtVelocidad);
    add(new JLabel("Agilidad (1-10):"));
    add(txtAgilidad);
    add(new JLabel("Defensa (1-50):"));
    add(txtDefensa);
```

- Guardar Personaje: En esta clase se guarda también el personaje y verifica si lo que el usuario agrega es válido o verifica también si se ingresaron valores en todos los campos.

```
private void guardarPersonaje() {
    try {
        String nombre = txtNombre.getText().trim();
        String arma = txtArma.getText().trim();
        String strHP = txtHP.getText().trim();
        String strAtaque = txtAtaque.getText().trim();
        String strVelocidad = txtVelocidad.getText().trim();
        String strAgilidad = txtAgilidad.getText().trim();
        String strDefensa = txtDefensa.getText().trim();

        // Validar campos vacíos
        if (nombre.isEmpty() || arma.isEmpty() || strHP.isEmpty() || strAtaque.isEmpty()
                || strVelocidad.isEmpty() || strAgilidad.isEmpty() || strDefensa.isEmpty()) {
            JOptionPane.showMessageDialog(this, "Todos los campos deben estar llenos.");
            return;
        }
```

- Validaciones de Rango: Así mismo verifica si en los campos de números lo que ingrese el usuario es válido y cumple.

```
// Validaciones de rango
if (hp < 100 || hp > 500) throw new Exception("HP fuera de rango (100-500).");
if (ataque < 10 || ataque > 100) throw new Exception("Ataque fuera de rango (10-100).");
if (velocidad < 1 || velocidad > 10) throw new Exception("Velocidad fuera de rango (1-10).");
if (agilidad < 1 || agilidad > 10) throw new Exception("Agilidad fuera de rango (1-10).");
if (defensa < 1 || defensa > 50) throw new Exception("Defensa fuera de rango (1-50).");
```

- Clase Personaje: Esta clase se encargar de almacenar y de ir guardando a los personajes de 1 en 1, cada uno va adquiriendo un ID distinto.

```
package practica2;
public class Personaje {
    private static int contadorID = 1;
    private int id;
    private String nombre;
    private String arma;
    private int hp;
    private int ataque;
    private int velocidad;
    private int agilidad;
    private int defensa;
```

- Contador: Asegura que el contador no se repita y vaya sumando ++1.

```
        // Asegurarse de que el contador no repita ID
            if (id >= contadorID) {
                contadorID = id + 1;
        }
    }
```

- Clase Ventana: Esta ventana es la que se encarga de los funcionamientos de los botones mayormente de cada opción y de verificar si se cumplen sus funciones.

```
public class Ventana extends JFrame {

    public static final int MAX_PERSONAJES = 100;
    public static Personaje[] personajes = new Personaje[MAX_PERSONAJES];
    public static int cantidadPersonajes = 0;
    public static final int MAX_HISTORIAL = 100;
    public static String[] historialNombres = new String[MAX_HISTORIAL];
    public static int cantidadHistorial = 0;
    public static final int MAX_BATALLAS = 100;
    public static HistorialBatalla[] historialBatallas = new HistorialBatalla[MAX_BA
    public static int cantidadBatallas = 0;

    public static void agregarBatallaHistorial(HistorialBatalla batalla) {
        if (cantidadBatallas < MAX_BATALLAS) {
            historialBatallas[cantidadBatallas++] = batalla;
        }
    }
```

  - Creación de Botones: Crea botones de todas las opciones.

```
//crear botones
JButton btnAgregar = new JButton("Agregar Personaje");
JButton btnModificar = new JButton("Modificar Personaje");
JButton btnEliminar = new JButton("Eliminar Personaje");
JButton btnVisualizar = new JButton("Visualizar Personajes");
JButton btnBatalla = new JButton("Simular Batalla");
JButton btnHistorial = new JButton("Historial de batallas");
JButton btnBuscar = new JButton("Buscar Personaje");
JButton btnGuardar = new JButton("Guardar Estado");
JButton btnCargar = new JButton("Cargar Estado");
JButton btnDatosEstudiante = new JButton("Mostrar datos del estudiante");
```

  - Acción de modificar:

```
//accion boton modificar
btnModificar.addActionListener(e -> {
    String entrada = JOptionPane.showInputDialog(this, "Ingrese ID o Nombre del personaje:");
    if (entrada != null && !entrada.trim().isEmpty()) {
        Personaje encontrado = buscarPorNombreOID(entrada.trim());
        if (encontrado != null) {
            new ModificarPersonaje(this, encontrado).setVisible(true);
    }else {
            JOptionPane.showMessageDialog(this, "Personaje no encontrado.");
        }
    }
        });
```

- Acción de botón eliminar:

```
//accion boton eliminar
btnEliminar.addActionListener(e -> {
    String entrada = JOptionPane.showInputDialog(this, "Ingrese ID o Nombre del personaje a eliminar:");
    if (entrada != null && !entrada.trim().isEmpty()) {
        Personaje encontrado = buscarPorNombreOID(entrada.trim());
        if (encontrado != null) {
            int confirm = JOptionPane.showConfirmDialog(this,
        "Seguro que desea eliminar al personaje:\n" + encontrado + "?",
        "Confirmar eliminacion", JOptionPane.YES_NO_OPTION);
        if (confirm == JOptionPane.YES_OPTION) {
            boolean eliminado = eliminarPersonajePorNombreOID(entrada.trim());
            if (eliminado){
                JOptionPane.showMessageDialog(this, "Personaje eliminado correctamente.");
            }else{
                JOptionPane.showMessageDialog(this, "Error al eliminar el personaje.");
            }
        }
    }else {
            JOptionPane.showMessageDialog(this, "Personaje no encontrado.");
        }
    }
});
```

- Acción de botón visualizar:

```
//accion boton visualizar
btnVisualizar.addActionListener(e -> {
    String lista = obtenerListaPersonajes();
        JTextArea textArea = new JTextArea(lista);
            textArea.setEditable(false);
                JScrollPane scrollPane = new JScrollPane(textArea);
            scrollPane.setPreferredSize(new Dimension(500, 300));
        JOptionPane.showMessageDialog(this, scrollPane, "Personajes Registrados", JOptionPane.INFORMATION_MESSAGE);
});
```

- Acción simular batalla:

```
//simular batalla
btnBatalla.addActionListener(e -> {
    if (cantidadPersonajes ==0){
        JOptionPane.showMessageDialog(null, "No hay personajes disponibles para realizar la batalla", "ERROR", JOptionPane.ERROR_MESSAGE);
    }else{
        new SimularBatalla(this).setVisible(true);
    }
});
```

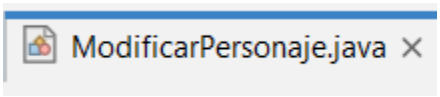
- Historial de batallas y mostrar personajes:

```
//Historial de batallas
btnHistorial.addActionListener(e -> verHistorialBatallas(this));

//Buscar personajes
btnBuscar.addActionListener(e -> {
    String nombre = JOptionPane.showInputDialog(this, "Ingrese el nombre del personaje:");
        if (nombre != null && !nombre.trim().isEmpty()) {
            buscarPersonajePorNombre(nombre.trim());
        } else {
            JOptionPane.showMessageDialog(this, "Entrada inválida.");
        }
});
```

- Creación de paneles funcionales de los botones:

```
//crear panel y agregar botones
JPanel panel = new JPanel();
panel.add(btnAgregar);
panel.add(btnModificar);
panel.add(btnEliminar);
panel.add(btnVisualizar);
panel.add(btnBatalla);
panel.add(btnHistorial);
panel.add(btnBuscar);
panel.add(btnGuardar);
panel.add(btnCargar);
panel.add(btnDatosEstudiante);

add(panel);
setVisible(true);
}
```

- Clase Modificar Personaje: Esta clase se encarga de modificar al personaje y de igual manera verifica que se cumplan los rangos que ingrese el usuario sean acorde a los que muestra.



- Guardar Cambios: Guardar oficialmente los cambios realizados en los personajes.

```
private void guardarCambios() {
    try {
        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());

        if (arma.isEmpty()) throw new Exception("El campo Arma no puede estar vacío.");
        if (hp < 100 || hp > 500) throw new Exception("HP fuera de rango.");
        if (ataque < 10 || ataque > 100) throw new Exception("Ataque fuera de rango.");
        if (velocidad < 1 || velocidad > 10) throw new Exception("Velocidad fuera de rango.");
        if (agilidad < 1 || agilidad > 10) throw new Exception("Agilidad fuera de rango.");
        if (defensa < 1 || defensa > 50) throw new Exception("Defensa fuera de rango.");
```

- Clase Simular Batalla: Esta clase se encarga de realizar la batalla.

```
public SimularBatalla(JFrame parent) {
    super(parent, "Seleccionar personajes para batalla", true);
    setSize(400, 200);
    setLayout(new GridLayout(3, 2));
    setLocationRelativeTo(parent);

    combo1 = new JComboBox<>();
    combo2 = new JComboBox<>();

    // Agregar personajes vivos al combo
    for (int i = 0; i < Ventana.cantidadPersonajes; i++) {
        Personaje p = Ventana.personajes[i];
        if (p.getHp() > 0) {
            combo1.addItem(p.getNombre() + " (ID: " + p.getId() + ")");
            combo2.addItem(p.getNombre() + " (ID: " + p.getId() + ")");
        }
```

  - Iniciar Batalla: Inicia la batalla mediante su botón y valida que los personajes sean diferentes ya que los personajes no pueden ser los mismos.

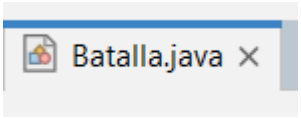
```
private void iniciarBatalla() {
    String seleccion1 = (String) combo1.getSelectedItem();
    String seleccion2 = (String) combo2.getSelectedItem();

    if (seleccion1.equals(seleccion2)) {
        JOptionPane.showMessageDialog(this, "Debe seleccionar dos personajes distintos.");
        return;
    }
```

- Clase Ventalla de Batalla: Esta clase se encarga de generar una nueva ventana en donde se realizara la batalla donde se muestran los hilos de los ataque de cada uno de los personajes.

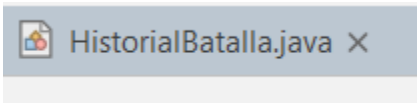
VentanaBatalla.java

```
public VentanaBatalla(Personaje p1, Personaje p2) {
    setTitle("Batalla: " + p1.getNombre() + " vs " + p2.getNombre());
    setSize(600, 400);
    setLayout(new BorderLayout());
    setLocationRelativeTo(null);

    areaBitacora = new JTextArea();
    areaBitacora.setEditable(false);
    JScrollPane scroll = new JScrollPane(areaBitacora);
    add(scroll, BorderLayout.CENTER);
```

- Clase Batalla: En esta clase es en donde se ejecutan las peleas.



- Clase Historial Batalla: En esta clase se almacena toda la información de las batallas que se realizan.



2. Software Necesarios:
   - ✓ Sistema Operativo:
     - Windows 10/11
   - ✓ Java Development Kit (JDK):
     - Versión 8 o superior
   - ✓ IDE:
     - NetBeans 12 en adelante
   - ✓ Librerías externas:
     - ITextPDF para poder generar reportes en formato PDF.

3. Hardware Necesarios:
   - ✓ Procesador: Intel 5 en adelante.
   - ✓ Memoria RAM: 4GB.
   - ✓ Almacenamiento 500 MB en adelante.
   - ✓ Una computadora.
   - ✓ Un mouse si en caso se está utilizando computadora de escritorio.
   - ✓ Un teclado si en caso se está utilizando computadora de escritorio.

4. Métodos más Importantes:

- Simular Batalla: Verifica que los dos personajes seleccionados sean diferentes y que estén vivos.

```
private void iniciarBatalla() {
    String seleccion1 = (String) combo1.getSelectedItem();
    String seleccion2 = (String) combo2.getSelectedItem();

    if (seleccion1.equals(seleccion2)) {
        JOptionPane.showMessageDialog(this, "Debe seleccionar dos personajes distintos.");
        return;
    }
```

- Simular Batalla: Lanza dos hilos paralelos que ejecutan ataques entre los personajes. Thread.

```
public static void simularBatalla(Personaje p1, Personaje p2, VentanaBatalla ventana) {
    batallaFinalizada = false;

    Thread t1 = new Thread(() -> ejecutarTurnos(p1, p2, ventana));
    Thread t2 = new Thread(() -> ejecutarTurnos(p2, p1, ventana));

    t1.start();
    t2.start();
}
```

- Ejecutar Turnos: Realiza los ataques de un personaje a otro, considerando esquivas, daño mínimo y las actualizaciones del HP.

```
private static void ejecutarTurnos(Personaje atacante, Personaje defensor, VentanaBatalla ventana) {
while (true) {
    try {
        Thread.sleep(1000 / atacante.getVelocidad());

        boolean fin;
        synchronized (Batalla.class) {
            if (batallaFinalizada || defensor.getHp() <= 0 || atacante.getHp() <= 0) {
                break;
            }
```

- Buscar por Nombre o ID: Permite buscar un personaje ya sea por nombre o por ID, lo cual facilita el acceso a personajes específicos.

```
public static void buscarPersonajePorNombre(String nombreBuscado) {
    boolean encontrado = false;
        int totalBatallas = 0;
        int ganadas = 0;
        int perdidas = 0;

    for (int i = 0; i < cantidadPersonajes; i++) {
        Personaje p = personajes[i];
    if (p.getNombre().equalsIgnoreCase(nombreBuscado)) {
        encontrado = true;
```

- Agregar Personaje: Crea un nuevo personaje con los datos proporcionados y lo agrega al arreglo personajes.

```
//Agregar personaje
    public static boolean agregarPersonaje(Personaje p) {
        if (cantidadPersonajes >= MAX_PERSONAJES) {
    return false;
    }
        personajes[cantidadPersonajes] = p;
        cantidadPersonajes++;
            return true;
    }
```

- Eliminar Personaje: Elimina a los personajes.

```
public static boolean eliminarPersonajePorNombreOID(String entrada) {
    for (int i = 0; i < cantidadPersonajes; i++) {
        Personaje p = personajes[i];
            if (p.getNombre().equalsIgnoreCase(entrada) || String.valueOf(p.getId()).equals(entrada)) {
                agregarNombreAlHistorial(p.getNombre());
        for (int j = i; j < cantidadPersonajes - 1; j++) {
            personajes[j] = personajes[j + 1];
        }
        personajes[cantidadPersonajes - 1] = null;
        cantidadPersonajes--;
        return true;
            }
    }
    return false;
}
```