Informe de Desarrollo

Sistema de gestión de personajes

Introducción

Este informe describe el proceso de desarrollo del **Sistema de Gestión de Personajes**, un programa en Java creado como parte del curso de Introducción a la Programación. El objetivo fue implementar un menú interactivo para administrar personajes, registrar peleas y gestionar datos, aplicando conceptos básicos como estructuras de control, arreglos y validación de entrada.

El Sistema de Gestión de Personajes fue diseñado para:

1. Servir como herramienta práctica de aprendizaje
2. Demostrar la aplicación de conceptos teóricos en un proyecto concreto
3. Resolver problemas comunes en el desarrollo de software (validaciones, manejo de errores, etc.)

Estructura general

Un menú principal (do-while)

hace que el menú al momento de ejecutar alguna acción de las diferentes opciones se vuelve a presentar a menos que se escoja la opción 9 que en ese caso ejecutaría la opción while lo cual hace que el programa presente un mensaje de despedida y cierre el programa para terminar con el ciclo.

Matrices estáticas (String[][])

Para almacenar personajes y el historial de peleas que se realizaron como también la hora y fecha exacta en la que se realizaon.

Validaciones
que sirven para evitar que los datos no sean duplicados o invalidos.

```
boolean existeNombre = false;
for (int i = 0; i < totalPersonajes; i++) {
    if (personajes[i][0].equalsIgnoreCase(nombre)) {
        existeNombre = true;
        break;
    }
}
```

Sirvió para ver si el personaje que se intenta registrar ya existía en el programa y evitar duplicar personajes.

```
try {
    nivel = Integer.parseInt(nivelStr);  // Intenta convertir a número
    if (nivel < 1 || nivel > 100) {
        System.out.println("El nivel debe estar entre 1 y 100.");
        break;
    }
} catch (NumberFormatException e) {  // Si no es un número
    System.out.println("¡Debes ingresar un número válido!");
    break;
}
```

Try que evita que se ingresen letras en ves de números o si el valor que se ingreso es mayor que 100 lanzar un mensaje de que debes de ingresar un numero valido.

En esta caso los problemas que se evitaron son:

- Entradas no numéricas (ej: "abc").
- Números fuera de rango (ej: -5 o 150).

```java
int per1 = -1;
for (int i = 0; i < totalPersonajes; i++) {
    if (personajes[i][0].equalsIgnoreCase(nombre1)) {
        per1 = i;
        break;
    }
}
if (per1 == -1) {  // Si no se encontró
    System.out.println("Personaje 1 no existe.");
    break;
}
```

En este caso la validación de peleas entre personajes y virificar que existan ambos personajes para la peles, y se evito peleas con personajes no registrados.

```java
System.out.print("¿Estás seguro de que deseas eliminar a " + personajes[indexEliminar][0] + "? (s/n): ");
String confirmacion = scanner.nextLine().trim().toLowerCase();
if (!confirmacion.equals("s")) {
    System.out.println("Eliminación cancelada.");
    break;
}
```

La confirmación para evitar que accidentalmente se elimene unpersonaje.ademas en este caso se implemento el conector lógico negativo.

Utilice mucho sout+tab para poder agregar System.*out*.println y poder imprimir mensajes en la consola de manera mas fácil como en el caso del menú

```java
System.out.println("\n==MENU==");
System.out.println("1. Agregar Personaje");
System.out.println("2. modificar personaje");
System.out.println("3. Eliminar personaje");
System.out.println("4. Ver datos de personaje");
System.out.println("5. Ver Listado de personajes");
System.out.println("6. Realizar pelea entre personajes")
System.out.println("7. Ver historial de peleas");
System.out.println("8. Ver datos de estudiantes");
System.out.println("9. salir");
System.out.print("Elige una opción: ");
opcion = scanner.nextInt();
```

For

Ejecuta un bloque de código varias veces y recorre todos los personajes para buscar uno por nombre como en el case 1 y 2 que necesita buscar los nombres de los personajes para poder registrar un personaje no existente y para poder encontrar el personaje que se va eliminar.

```java
boolean existeNombre = false;
for (int i = 0; i < totalPersonajes; i++) {
    if (personajes[i][0].equalsIgnoreCase(nombre)) {
        existeNombre = true;
        break;
    }
}
```

Switch:

Implemente en este caso 9 cases, que me ayudaron en casa uno poder meter los códigos de todas las opciones que necesitaba ejecutar que se presentaban en el menú.

```java
case 5:
    System.out.println("\n***Has seleccionado: Ver listado de personajes");

    if (totalPersonajes == 0) {
        System.out.println("No existen personajes.");
        break;
    }
    System.out.println("Personajes registrados:");
    for (var i = 0; i < totalPersonajes; i++) {
        var nombrePersonaje = personajes[i][0];
        var nivelPoder = personajes[i][7];
        System.out.println("Nombre: " + nombrePersonaje + " | Nivel de poder: " + nivelPoder);
    }
    break;
```

Terminando siempre al final con un break que anula todas las funciones y cancelan las funciones del case que se esta ejecutando asi poder imprimir el menú nuevamente para poder escoger otra opción.

Resultados

Funcionalidades completas: Todas las opciones del menú operan correctamente.
Validaciones robustas: El programa no crashea con entradas incorrectas.
Limitaciones:

- No guarda datos entre ejecuciones (persistencia).
- Ordenamiento básico (muestra registros en orden de inserción).

5. Conclusiones y Aprendizajes

- **Logros**:
    - Dominio de estructuras de control (switch, do-while y aprendí que es una matriz para almacenar datos).
    - Manejo y aprendizaje del código para poder obtener la fecha y hora local con LocalDateTime.
- **Dificultades**: el realizar los códigos y el poco tiempo que llevo aprendiendo a utilizar java que no es mayor a 30 dias,

  El uso de librerías por que al investigar en internet me tiran resultadosa como usar arraylist y como otras librerías relacionadas a objetos y con el poco conocimiento que tengo no tengo la capacidad de diferenciar cuales son relacionadoa a objetos y cuales no.