



Universidad de San Carlos de Guatemala  
Facultad de Ingeniería

**PRÁCTICA #1**  
**MANUAL TÉCNICO**  
**JUEGO DE LA SOPA DE LETRAS Y CRUD DE PALABRAS**

**Introducción a la Programación 1**  
Auxiliar Sebastian Alejandro Velasquez Bonilla

Guatemala, febrero de 2025

# 1. Versiones y Requisitos

## 1.1 Versión del Lenguaje

- Lenguaje: Java (versión recomendada: Java 17 o superior).

## 1.2 Entorno de Desarrollo Recomendado

- IDE Recomendado:
  - NetBeans 12.6 o superior.
  - Visual Studio Code con la extensión "Extension Pack for Java".
- Compilador: JDK 17 o superior.

## 1.3 Librerías Utilizadas

- `java.util.Scanner`: Para la entrada de datos del usuario.
- `java.util.Random`: Para la generación aleatoria de caracteres en el tablero.
- `java.util.ArrayList`: Para la gestión de palabras y jugadores.

# 2. Estructura del Proyecto

## 2.1 Archivos y Clases Principales

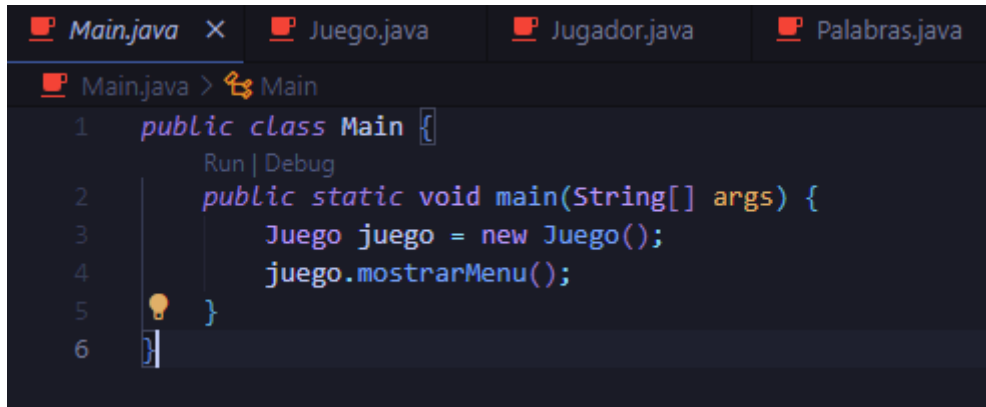
Archivo	Descripción
<i>Main.java</i>	Punto de entrada del programa, inicializa el juego.
<i>Juego.java</i>	Controla el flujo del juego y la interacción con el usuario.
<i>Jugador.java</i>	Representa al jugador y su puntuación.
<i>Palabras.java</i>	Maneja las palabras ingresadas en el juego (CRUD).
<i>Tablero.java</i>	Genera y gestiona el tablero de juego.
<i>Historial.java</i>	Almacena el historial de partidas jugadas.
<i>CargasAnimaciones.java</i>	Gestiona animaciones visuales en consola.



## 3. Explicación del Código

### 3.1 Main.java

Punto de entrada del programa. Inicializa una instancia de Juego y llama al método.

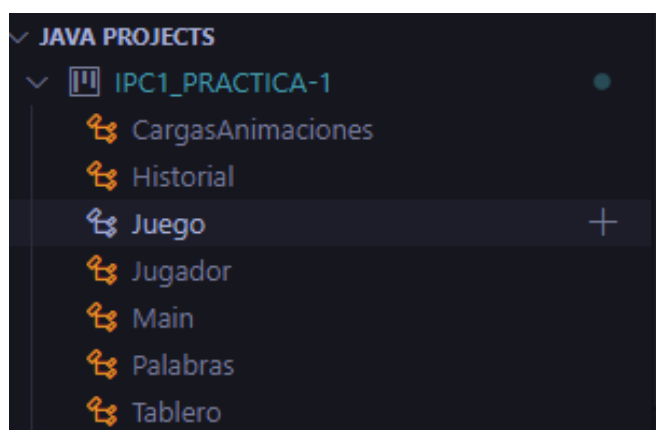
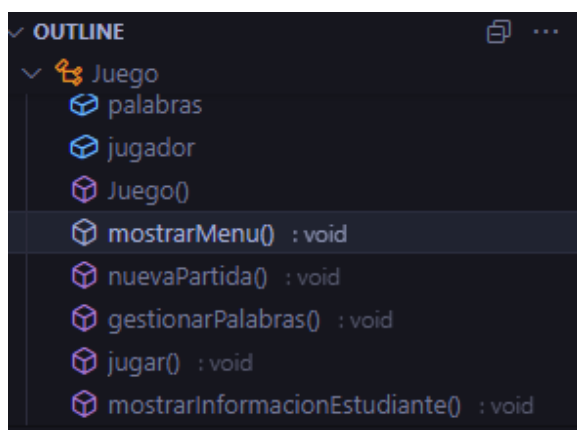


```
1 public class Main {  
2     public static void main(String[] args) {  
3         Juego juego = new Juego();  
4         juego.mostrarMenu();  
5     }  
6 }
```

### 3.2 Juego.java

Esta clase controla la lógica principal del juego.

- `mostrarMenu()`: Despliega el menú principal.
- `nuevaPartida()`: Inicia una nueva partida.
- `gestionarPalabras()`: Permite agregar, modificar y eliminar palabras.
- `jugar()`: Controla el desarrollo del juego, manejo de intentos, validación de palabras y puntuación.
- `mostrarInformacionEstudiante()`: Muestra los datos del estudiante.



### 3.2.1 Palabras.java - CRUD de Palabras

```
class Palabras {
    private List<String> palabras = new ArrayList<>();

    // CANTIDAD INICIAL Y CREAR PALABRAS
    public void gestionarPalabras(Scanner scanner) {
        System.out.print(s:"Ingrese la cantidad de palabras: ");
        int cantidad = scanner.nextInt();
        scanner.nextLine();

        for (int i = 0; i < cantidad; i++) {
            while (true) {
                System.out.print("Ingrese la palabra " + (i + 1) + ": ");
                String palabra = scanner.nextLine().toUpperCase();

                if (!validarLongitud(palabra)) {
                    System.out.println(x:"Error: La palabra debe tener entre 3 y 8 caracteres.");
                } else {
                    palabras.add(palabra);
                    break;
                }
            }
        }
    }
}
```

```
// MODIFICAR PALABRA
public void modificarPalabra(Scanner scanner) {
    if (palabras.isEmpty()) {
        System.out.println(x:"No hay palabras ingresadas.");
        return;
    }

    mostrarPalabras();

    System.out.print(s:"Ingrese la palabra a modificar: ");
    String palabraAntigua = scanner.nextLine().toUpperCase();

    if (!palabras.contains(palabraAntigua)) {
        System.out.println(x:"Error: La palabra no se encuentra en la lista.");
        return;
    }

    while (true) {
        System.out.print(s:"Ingrese la nueva palabra: ");
        String palabraNueva = scanner.nextLine().toUpperCase();

        if (!validarLongitud(palabraNueva)) {
            System.out.println(x:"Error: La palabra debe tener entre 3 y 8 caracteres.");
        } else {
            palabras.set(palabras.indexOf(palabraAntigua), palabraNueva);
            System.out.println(x:"Palabra modificada correctamente.");
            break;
        }
    }
}
```

```
// ELIMINAR PALABRA
public void eliminarPalabra(String palabra) {
    boolean eliminada = palabras.removeIf(p -> p.equalsIgnoreCase(palabra));

    if (eliminada) {
        System.out.println(x:"Palabra eliminada correctamente.");
    } else {
        System.out.println(x:"Error: La palabra no se encuentra en la lista.");
    }
}

private boolean validarLongitud(String palabra) {
    return palabra.length() >= 3 && palabra.length() <= 8;
}

public void mostrarPalabras() {
    if (palabras.isEmpty()) {
        System.out.println(x:"No hay palabras ingresadas.");
    } else {
        System.out.println("Palabras actuales: " + palabras);
    }
}

public List<String> getPalabras() {
    return palabras;
}
}
```

### 3.3 Tablero.java

Clase encargada de la creación y administración del tablero de juego

```
public void generarTablero(List<String> palabras) {

    CargasAnimaciones.mostrarCarga(mensaje:"Creando el tablero...");

    // SIZE SEGUN TABLA 1
    this.tamaño = 15;
    this.tablero = new char[tamaño][tamaño];

    for (int i = 0; i < tamaño; i++) {
        Arrays.fill(tablero[i], val:' ');
    }

    for (String palabra : palabras) {
        colocarPalabra(palabra);
    }

    for (int i = 0; i < tamaño; i++) {
        for (int j = 0; j < tamaño; j++) {
            if (tablero[i][j] == ' ') {
                tablero[i][j] = (char) ('A' + random.nextInt(bound:26));
            }
        }
    }
}
```

### 3.4 Juego.java - Método *jugar()*

```
private void jugar() {  
    if (palabras.getPalabras().isEmpty()) {  
        System.out.println(x:"No hay palabras para jugar.");  
        return;  
    }  
    int errores = 0;  
    jugador.aumentarPuntuacion(puntos:25);  
    // NUMERO DE PALABRAS INICIALES  
    int totalPalabras = palabras.getPalabras().size();
```

```
    while (errores < 4) {  
        System.out.print(s:"Ingrese una palabra: ");  
        String palabra = scanner.nextLine().toUpperCase();  
  
        if (palabras.getPalabras().contains(palabra)) {  
            System.out.println(x:"¡Palabra encontrada!");  
            jugador.aumentarPuntuacion(palabra.length());  
            palabras.eliminarPalabra(palabra);  
            tablero.reemplazarPalabra(palabra);  
            tablero.imprimirTablero();  
        } else {  
            errores++;  
            System.out.println("Palabra incorrecta. Llevas " + errores + "/4 errores.");  
            jugador.aumentarPuntuacion(-5);  
        }  
    }
```

```
    // PROGRESO DEL JUEGO  
    System.out.println("Palabras encontradas: " + (totalPalabras - palabras.getPalabras().size()));  
    System.out.println("Palabras pendientes: " + palabras.getPalabras().size());  
  
    if (palabras.getPalabras().isEmpty()) {  
        System.out.println("¡Ganaste! Puntuación final: " + jugador.getPuntuacion());  
        Historial.agregarJugador(jugador, errores, totalPalabras);  
        return;  
    }  
}  
  
System.out.println("¡Perdiste! Puntuación final: " + jugador.getPuntuacion());  
Historial.agregarJugador(jugador, errores, totalPalabras - palabras.getPalabras().size());  
}
```

### 3.5 Historial.java

Clase que almacena el historial de partidas jugadas.

- *agregarJugador(Jugador jugador, int fallos, int palabrasEncontradas)*: Registra el resultado de una partida.
- *mostrarHistorial()*: Muestra todas las partidas jugadas.
- *mostrarPuntuacionesAltas()*: Muestra el top 3 de jugadores con más puntos.

```

public class Historial {
    private static List<Jugador> jugadores = new ArrayList<>();

    public static void agregarJugador(Jugador jugador, int fallos, int palabrasEncontradas) {
        jugador.setFallos(fallos);
        jugador.setPalabrasEncontradas(palabrasEncontradas);
        jugadores.add(jugador);
    }
}

```

```

// ARMANDO LA TABLA DE RESULTADOS REUTILIZABLE
private static void mostrarTabla(List<Jugador> lista, String titulo) {
    if (lista.isEmpty()) {
        System.out.println(x:"No hay datos para mostrar.");
        return;
    }
    System.out.println("\n--- " + titulo + " ---");
    System.out.printf(format:"%-15s %-10s %-10s %-10s%n", ...args:"Nombre", "Puntos", "Fallos", "Encontradas");
    System.out.println(x:"-----");

    for (Jugador j : lista) {
        System.out.printf(format:"%-15s %-10d %-10d %-10d%n",
            j.getNombre(), j.getPuntuacion(), j.getFallos(), j.getPalabrasEncontradas());
    }
}

```

Usando la tabla de resultados reutilizable mostrar el Array de Jugador (el cual contiene su información de puntos, fallos y número de palabras encontrada) o bien, solo mostrar el top 3 de jugadores con mejor puntaje, este se ordena dentro del Arreglo usando `.sort` y `.compare`

```

public static void mostrarHistorial() {
    mostrarTabla(jugadores, titulo:"HISTORIAL DE PARTIDAS");
}

public static void mostrarPuntuacionesAltas() {
    List<Jugador> topJugadores = new ArrayList<>(jugadores);
    topJugadores.sort((a, b) -> Integer.compare(b.getPuntuacion(), a.getPuntuacion()));
    mostrarTabla(topJugadores.subList(fromIndex:0, Math.min(a:3, topJugadores.size())), titulo:"TOP 3 JUGADORES");
}
}

```

## 4. Flujo de Ejecución

1. Se ejecuta *Main.java*, lo que inicializa Juego y muestra el menú.
2. El usuario selecciona Nueva Partida, ingresa su nombre y gestiona las palabras.
3. Se genera un tablero con las palabras ingresadas.
4. Se inicia el juego con un límite de 4 errores.
5. El usuario intenta encontrar palabras.
6. El juego termina cuando se encuentran todas las palabras o se superan 4 errores.
7. Se actualiza el historial y se muestra el puntaje final.

## 5. Conclusión

Este documento detalla la implementación del juego de sopa de letras en Java, proporcionando una guía técnica clara para su comprensión y mantenimiento.