Universidad de San Carlos de Guatemala
Facultad de Ingeniería
Escuela de Ciencias y Sistemas
Laboratorio Introducción a la Programación y Computación 1
Sección: D
Primer Semestre 2025
Ing. Herman Veliz
Aux. Lesther López

PRÁCTICA #1
SOPA DE LETRAS

MANUAL TÉCNICO

Armando José Vásquez Castillo
Carnet: 202401451

INTRODUCCIÓN:
En el siguiente manual se estarán presentando los aspectos técnicos del programa, es decir, todo lo relacionado a la parte programable con el fin de brindar el conocimiento de como las tareas y ejecuciones se están llevando a cabo.

FUNCIONAMIENTO:
El programa está diseñado en Java y consta de dos clases principales, cada una desempeñando un papel fundamental en la ejecución. Estas clases se denominan **“MenuPrincipal”** y **“EjecucionSopa”**.

La clase **“MenuPrincipal”** actúa como la interfaz inicial del proyecto. Su funcionamiento se basa en tres estructuras de control switch-case utilizadas de manera consecutiva, lo que permite presentar un menú en consola de forma organizada y eficiente. Además, esta clase se encarga de invocar los distintos métodos definidos en **“EjecucionSopa”**, los cuales analizaremos más adelante en detalle.

Por otro lado, la clase **“EjecucionSopa”** constituye el núcleo del desarrollo del juego, ya que en ella se implementan los métodos esenciales para la creación, ejecución y gestión de usuarios, así como para la administración de nuevas palabras y sus respectivas modificaciones. Además, integra sistemas de puntuación, restricciones y otras funciones clave para garantizar un correcto desempeño del juego.
Asimismo, esta clase también incorpora una estructura top-down, utilizada para organizar el flujo del programa de manera eficiente, facilitando la claridad y comprensión del código.

REQUERIMIENTOS:

1. Sistema Operativo: El programa debería poder ejecutarse en la mayoría de los sistemas operativos, incluidos:
   1.2. Windows (7, 8, 10, 11)
   1.3. macOS (10.x o versiones posteriores)
   1.4. Linux (cualquier distribución moderna)
2. Requisitos de Software: Necesitarás el JDK instalado en la máquina. Este es el entorno de desarrollo necesario para compilar y ejecutar programas Java.
   2.1. Versión recomendada: JDK 8 o superior. El programa debería funcionar en versiones más recientes de Java (como JDK 11 o 17) sin problemas, pero en general, JDK 8 es una buena opción de compatibilidad.
3. Un IDE o editor de texto (como IntelliJ IDEA, Eclipse, o incluso VS Code con las extensiones de Java).

CLASE 1: MenuPrincipal

```
int partida;
saltoMenu: //Variable para retornar a este punto
do {
    // Mostramos el menú en la consola con sus 5 opciones
    System.out.println("--MENÚ PRINCIPAL--");
    System.out.println("1. Nueva Partida");
    System.out.println("2. Historial de partidas");
    System.out.println("3. Puntuaciones mas altas");
    System.out.println("4. Información del estudiante");
    System.out.println("5. Salir");
    System.out.println("ESCOGE UNA VALOR:");

    // Leemos la opción que ingresa el usuario
    partida = scanner.nextInt();
    scanner.nextLine();
```

*Se inicia un **do-while**, donde primero se mostrarán en consola las 5 opciones disponibles para que el usuario pueda seleccionar y ejecutar. Seguidamente la ejecución del primer switch del menú principal.*

```
    switch (partida) {
        case 1:
            //partida 1.1: ejecuta el metodo NuevoUsuario
            EjecucionSopa.NuevoUsuario(scanner);
```

```
        case 2:
            // partida 2: mostramos nombre, puntos, fallos, palabras encontradas
            EjecucionSopa.mostrarHistorialPartidas();
            break;

        case 3:
            // partida 3: Mostramos las 3 puntuaciones más altas descendentemente.
            System.out.println("Apartado no disponible en estos momentos");
            break;

        case 4:
            // partida 4: Mostramos los datos del estudiante (nombre, carnet, sección)
            EjecucionSopa.InformacionEstudiante(scanner);
            break;

        case 5:
            // partida 5: Finaliza la ejecución
            System.out.println("Gracias por jugar, vuelve pronto");
            break;

        default:
            // Mensaje si el usuario ingresa una opción inválida
            System.out.println("Opción inválida. Inténtelo de nuevo.");
    }

} while (partida != 5); // El bucle se repite hasta que el usuario elija la opción 5 que significa salir.
```

Podemos identificar cinco case, cada uno correspondiente a las opciones mencionadas en el inciso anterior. Los primeros cuatro contienen los métodos necesarios para ejecutar las instrucciones seleccionadas por el usuario, mientras que el último case actúa como mecanismo para finalizar el programa.

```java
// partida 1.2: se presentan 3 opciones (Menu de palabras, jugar, terminar partida)
// Variable para almacenar la opción que el usuario seleccionará en el menú
int menu;
menuLoop: //Variable para retornar a este punto
do {
    System.out.println("--MENÚ SECUNDARIO--");
    System.out.println("1. Menú de palabras");
    System.out.println("2. Jugar");
    System.out.println("3. Terminar partida");
    System.out.println("ESCOGE UN VALOR:");

    // Leemos la opción que ingresa el usuario
    menu = scanner.nextInt();
    scanner.nextLine();
```

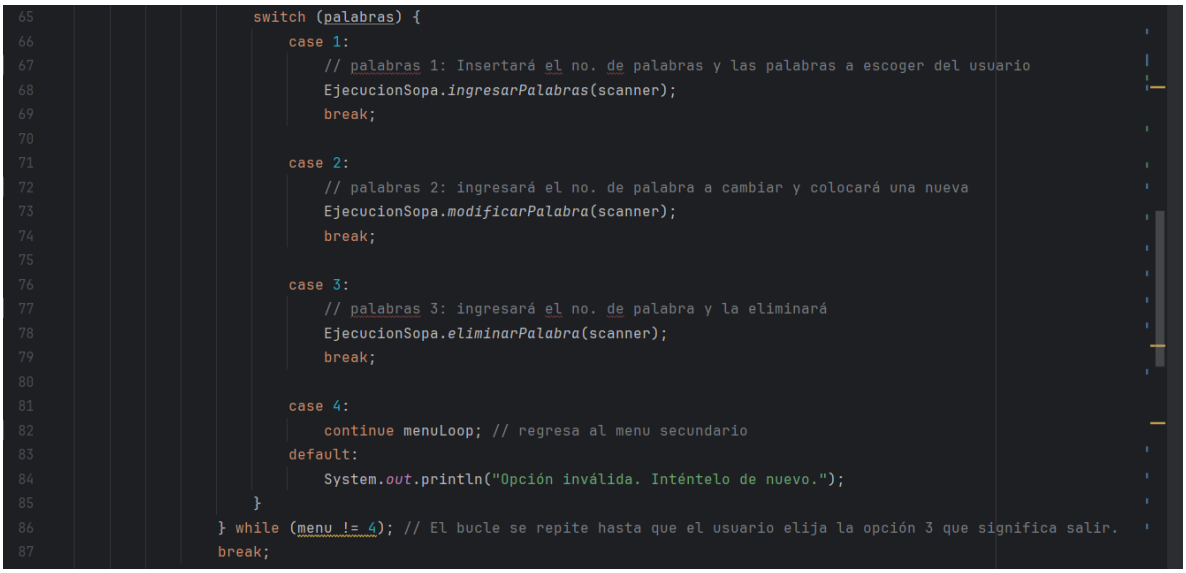
*Dentro del anterior **"case 1"**, se inicia un nuevo **do-while** con el propósito de ejecutar un submenú secundario. En primer lugar, se mostrarán en consola las tres opciones disponibles para que el usuario pueda seleccionar y ejecutar. Posteriormente, se llevará a cabo la ejecución del segundo switch correspondiente a este menú secundario.*

```java
    switch (menu) {
        case 1:
            //menu 1: se presentan 4 opciones (insertar, modificar, eliminar, salir)
        case 2:
            // menu 2: creará la tabla de juego y ejecutará el juego
            EjecucionSopa.inicioJuego(scanner);
            break;

        case 3:
            //menu 3: regresara al menu principal
            continue saltoMenu;
        default:
            System.out.println("Opción inválida. Inténtelo de nuevo.");
    }

} while (menu != 3); // El bucle se repite hasta que el usuario elija la opción 3 que significa salir.
break;
```

Podemos identificar tres case, cada uno correspondiente a las opciones mencionadas en el inciso anterior. El primero contiene el tercer switch, el segundo el método necesario para ejecutar la instrucción seleccionada por el usuario, mientras que el último case actúa como mecanismo para retornar al menú principal.

```java
        // Variable para almacenar la opción que el usuario seleccionará en el menú
        int palabras;
        do {
            System.out.println("--MENÚ DE PALABRAS--");
            System.out.println("1. Insertar palabras");
            System.out.println("2. Modificar palabra");
            System.out.println("3. Eliminar palabra");
            System.out.println("4. Salir");
            System.out.println("ESCOGE UN VALOR:");

            // Leemos la opción que ingresa el usuario
            palabras = scanner.nextInt();
            scanner.nextLine();
```

*Nuevamente, dentro del anterior **"case 1"**, se inicia un nuevo **do-while** con el propósito de ejecutar un submenú terciario. En primer lugar, se mostrarán en consola las cuatro opciones disponibles para que el usuario pueda seleccionar y ejecutar. Posteriormente, se llevará a cabo la ejecución del tercer switch correspondiente a este menú terciario.*



```
                switch (palabras) {
                    case 1:
                        // palabras 1: Insertará el no. de palabras y las palabras a escoger del usuario
                        EjecucionSopa.ingresarPalabras(scanner);
                        break;

                    case 2:
                        // palabras 2: ingresará el no. de palabra a cambiar y colocará una nueva
                        EjecucionSopa.modificarPalabra(scanner);
                        break;

                    case 3:
                        // palabras 3: ingresará el no. de palabra y la eliminará
                        EjecucionSopa.eliminarPalabra(scanner);
                        break;

                    case 4:
                        continue menuLoop; // regresa al menu secundario
                    default:
                        System.out.println("Opción inválida. Inténtelo de nuevo.");
                }
            } while (menu != 4); // El bucle se repite hasta que el usuario elija la opción 3 que significa salir.
            break;
```

Podemos identificar cuatro case, cada uno correspondiente a las opciones mencionadas en el inciso anterior. Los primeros tres contienen los métodos necesarios para ejecutar las instrucciones seleccionadas por el usuario, mientras que el último case actúa como mecanismo para retornar al menú secundario.

CLASE 2: EjecuciónSopa

```java
public class EjecucionSopa {  Vasquez005 *
    // Matriz para almacenar usuarios (máximo de 10 usuarios)
    static String[][] usuarios = new String[10][3]; // [número de usuarios][nombre, carnet, sección]  7 usages
    static int contadorUsuarios = 0;  7 usages
    //----------
    // Matriz para almacenar Historial de partidas (máximo 20 registros)
    static String[][] historialPartidas = new String[20][4]; // [numero de partidas][nombre, puntaje, fallos, palabras enc
    static int contadorPartidas = 0;  8 usages
    //----------
    //Declaración de variables utilizadas para el proyecto
    private static final int TAMANO = 25; // Tamaño de la matriz  12 usages
    private static final char[][] sopa = new char[TAMANO][TAMANO];  8 usages
    private static final Random random = new Random();  4 usages
    private static final char RELLENO = '#'; //Relleno de las palabras encontradas  2 usages
    private static final int INTENTOS_MAXIMOS = 4; //Definiendo el tamaño máximo de los intentos por usuario  2 usages
    private static String[] bancoPalabras = new String[100]; // Cramos un Banco de palabras con máximo de 100 palabras  9 us
    private static int totalPalabras = 0; // Contador de palabras en el banco de palabras  16 usages
```

En este primer segmento, se definen las variables globales que se utilizarán dentro de la clase. En términos generales, se establecen variables como el tamaño de la matriz para la sopa de letras, espacios para almacenar a los usuarios y el historial de partidas, así como un banco de palabras y otros elementos esenciales para el desarrollo del juego.

```java
    /* Proceso que ejecuta la creación de un nuevo usuario al seleccionar "Nueva Partida",
    Pedimos datos del usuario (nombre, carnet, sección)*/
    public static void NuevoUsuario(Scanner sc) {  1 usage  new *
        if (contadorUsuarios < 10) { // Verificamos que haya espacio en la matriz
            System.out.println("Ingrese su nombre: ");
            String nombre = sc.nextLine(); // Leemos el nombre
            System.out.println("Ingrese su carnet");
            String carnet = sc.nextLine(); // Leemos el carnet
            System.out.println("Ingrese su sección");
            String seccion = sc.nextLine(); // Leemos la sección
            System.out.println("Te damos la bienvenida a la sopa de letras " + nombre);

            // Guardamos en la matriz
            usuarios[contadorUsuarios][0] = nombre;
            usuarios[contadorUsuarios][1] = carnet;
            usuarios[contadorUsuarios][2] = seccion;
            contadorUsuarios++; // Incrementamos el contador de usuarios

        } else {
            System.out.println("Límite de usuarios alcanzado. No se pueden registrar más usuarios :)");
        }
    }
```

*El método **NuevoUsuario** se encarga de crear un nuevo usuario cuando se selecciona la opción "Nueva Partida" en el menú principal. Para ello, solicita al usuario información como nombre, carné y sección, la cual se almacena en una matriz para su uso posterior dentro del juego.*

```java
// Proceso que ejecuta la opción "Informacion del estudiante"
public static void InformacionEstudiante(Scanner sc) {  1 usage  Vasquez005 *
    if (contadorUsuarios == 0) {
        System.out.println("No hay usuarios registrados.");
        return;
    }

    System.out.println("Ingrese el nombre del usuario que desea consultar:");
    String nombreConsulta = sc.nextLine();
    boolean encontrado = false;

    for (int i = 0; i < contadorUsuarios; i++) {
        if (usuarios[i][0].equalsIgnoreCase(nombreConsulta)) {
            System.out.println("Información del estudiante:");
            System.out.println("Nombre: " + usuarios[i][0]);
            System.out.println("Carnet: " + usuarios[i][1]);
            System.out.println("Sección: " + usuarios[i][2]);
            encontrado = true;
            break;
        }
    }

    if (!encontrado) {
        System.out.println("Usuario no encontrado.");
    }
}
```

*El método **InformaciónEstudiante** permite buscar y mostrar la información de un estudiante previamente registrado en la matriz usuarios. Se encarga de recibir un nombre de usuario ingresado por el usuario y verificar si existe en la base de datos, para luego poder imprimir la información correspondiente del estudiante, incluyendo su nombre, carnet y sección en la opción "Información del estudiante". Si el usuario no existe, el método devuelve un mensaje indicando error.*

```java
public static void ingresarPalabras(Scanner sc) {  1 usage  Vasquez005 *
    System.out.print("Ingrese la cantidad de palabras a agregar: ");
    int numPalabras = sc.nextInt();
    sc.nextLine();

    for (int i = 0; i < numPalabras; i++) {
        if (totalPalabras >= bancoPalabras.length) {
            System.out.println("El banco de palabras está lleno.");
            break;
        }

        String palabra;
        do {
            System.out.print("Ingrese la palabra " + (totalPalabras + 1) + " (6-15 caracteres permitidos): ");
            palabra = sc.nextLine().toUpperCase();
            if (palabra.length() < 6 || palabra.length() > 15) {
                System.out.println("Error: La palabra debe tener entre 6 y 15 caracteres.");
            }
        } while (palabra.length() < 6 || palabra.length() > 15);

        bancoPalabras[totalPalabras++] = palabra;
    }
}
```

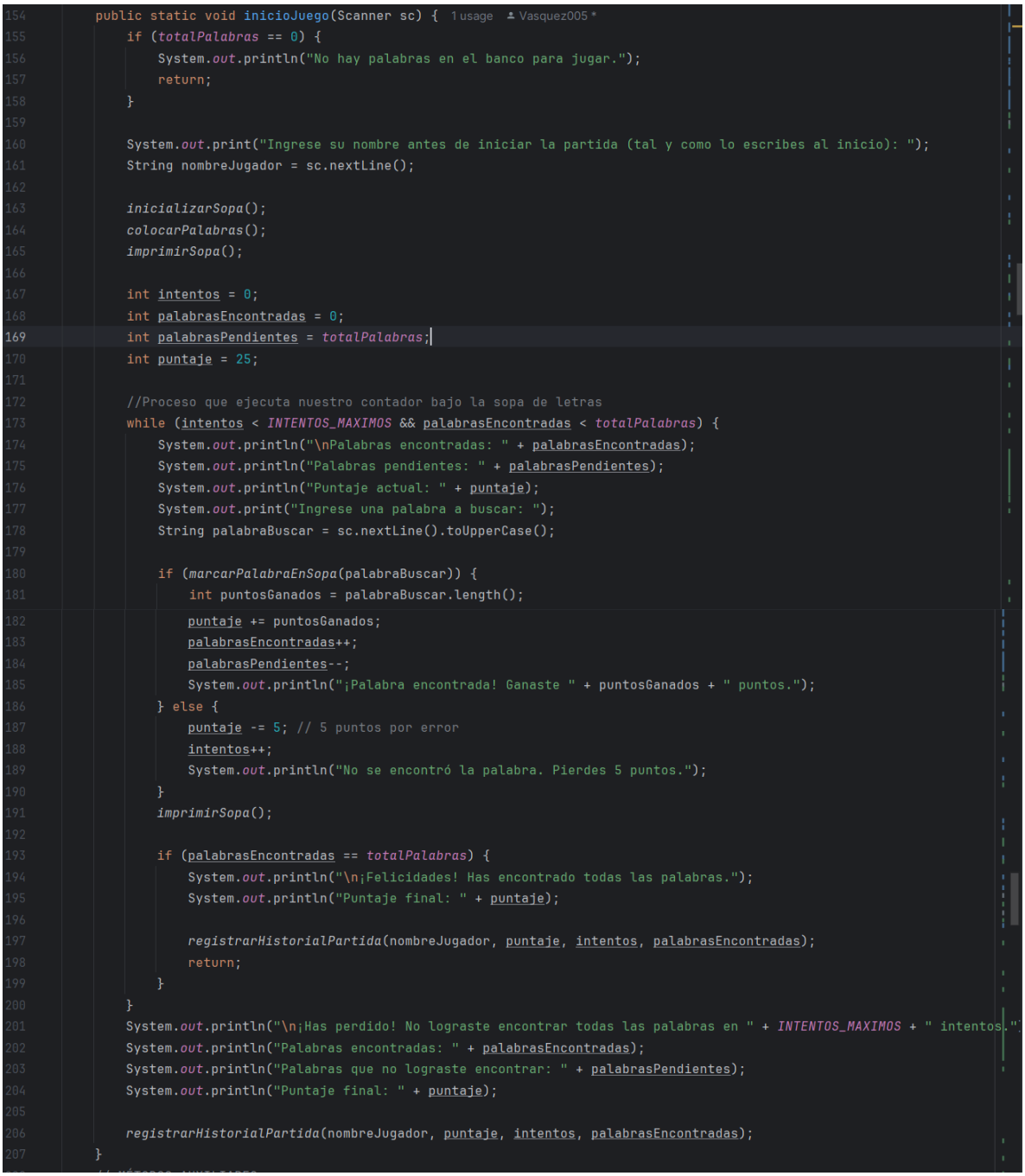
*El método **ingresarPalabra** permite al usuario agregar palabras al banco de palabras. Se encarga de solicitar la cantidad de palabras que se desean ingresar, verificar que cada palabra cumpla con los requisitos de longitud, y almacenarlas en la matriz.*

```java
public static void modificarPalabra(Scanner sc) {  1 usage   Vasquez005 *
    if (totalPalabras == 0) {
        System.out.println("No hay palabras en el banco para modificar.");
        return;
    }

    mostrarBancoPalabras();
    System.out.print("Ingrese el número de la palabra que desea modificar: ");
    int indice = sc.nextInt() - 1;
    sc.nextLine();

    if (indice >= 0 && indice < totalPalabras) {
        String nuevaPalabra;
        do {
            System.out.print("Ingrese la nueva palabra (6-15 caracteres permitidos): ");
            nuevaPalabra = sc.nextLine().toUpperCase();
            if (nuevaPalabra.length() < 6 || nuevaPalabra.length() > 15) {
                System.out.println("Error: La palabra debe tener entre 6 y 15 caracteres.");
            }
        } while (nuevaPalabra.length() < 6 || nuevaPalabra.length() > 15);

        bancoPalabras[indice] = nuevaPalabra;
        System.out.println("Palabra modificada correctamente.");
    } else {
        System.out.println("Número inválido, escoge otro.");
    }
}
```

*El método **modificarPalabra** permite modificar una palabra existente en el banco de palabras, garantizando que se cumplan ciertas restricciones. Si la palabra a modificar no existe, el método devuelve un mensaje indicando error.*

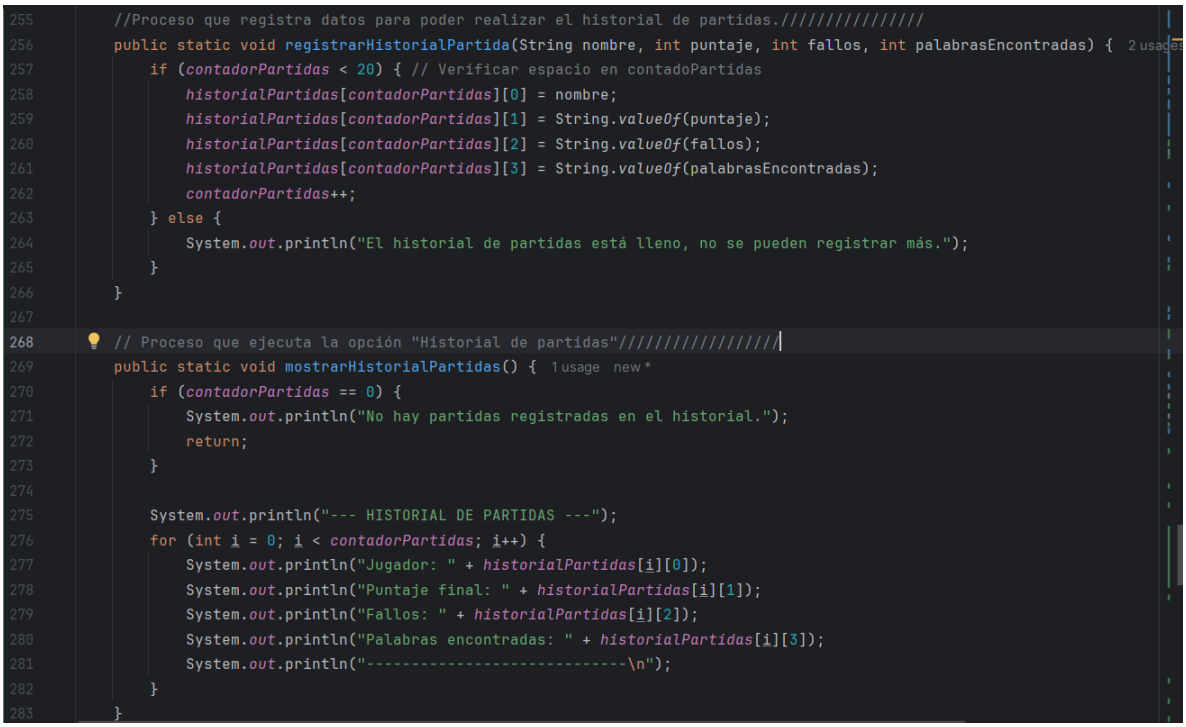
```java
public static void eliminarPalabra(Scanner sc) {  1 usage   Vasquez005 *
    if (totalPalabras == 0) {
        System.out.println("No hay palabras en el banco para eliminar.");
        return;
    }

    mostrarBancoPalabras();
    System.out.print("Ingrese el número de la palabra que desea eliminar: ");
    int indice = sc.nextInt() - 1;
    sc.nextLine();

    if (indice >= 0 && indice < totalPalabras) {
        for (int i = indice; i < totalPalabras - 1; i++) {
            bancoPalabras[i] = bancoPalabras[i + 1];
        }
        bancoPalabras[totalPalabras - 1] = null;
        totalPalabras--;
        System.out.println("Palabra eliminada correctamente.");
    } else {
        System.out.println("Número inválido, escoge otro.");
    }
}
```

*El método **eliminarPalabra** permite eliminar una palabra del banco de palabras, asegurando que la lista se mantenga organizada sin espacios vacíos. La eliminación se realiza desplazando las palabras restantes en el arreglo.*



```java
public static void inicioJuego(Scanner sc) {  1 usage   Vasquez005 *
    if (totalPalabras == 0) {
        System.out.println("No hay palabras en el banco para jugar.");
        return;
    }

    System.out.print("Ingrese su nombre antes de iniciar la partida (tal y como lo escribes al inicio): ");
    String nombreJugador = sc.nextLine();

    inicializarSopa();
    colocarPalabras();
    imprimirSopa();

    int intentos = 0;
    int palabrasEncontradas = 0;
    int palabrasPendientes = totalPalabras;
    int puntaje = 25;

    //Proceso que ejecuta nuestro contador bajo la sopa de letras
    while (intentos < INTENTOS_MAXIMOS && palabrasEncontradas < totalPalabras) {
        System.out.println("\nPalabras encontradas: " + palabrasEncontradas);
        System.out.println("Palabras pendientes: " + palabrasPendientes);
        System.out.println("Puntaje actual: " + puntaje);
        System.out.print("Ingrese una palabra a buscar: ");
        String palabraBuscar = sc.nextLine().toUpperCase();

        if (marcarPalabraEnSopa(palabraBuscar)) {
            int puntosGanados = palabraBuscar.length();
            puntaje += puntosGanados;
            palabrasEncontradas++;
            palabrasPendientes--;
            System.out.println("¡Palabra encontrada! Ganaste " + puntosGanados + " puntos.");
        } else {
            puntaje -= 5; // 5 puntos por error
            intentos++;
            System.out.println("No se encontró la palabra. Pierdes 5 puntos.");
        }
        imprimirSopa();

        if (palabrasEncontradas == totalPalabras) {
            System.out.println("\n¡Felicidades! Has encontrado todas las palabras.");
            System.out.println("Puntaje final: " + puntaje);

            registrarHistorialPartida(nombreJugador, puntaje, intentos, palabrasEncontradas);
            return;
        }
    }
    System.out.println("\n¡Has perdido! No lograste encontrar todas las palabras en " + INTENTOS_MAXIMOS + " intentos.");
    System.out.println("Palabras encontradas: " + palabrasEncontradas);
    System.out.println("Palabras que no lograste encontrar: " + palabrasPendientes);
    System.out.println("Puntaje final: " + puntaje);

    registrarHistorialPartida(nombreJugador, puntaje, intentos, palabrasEncontradas);
}
```

El método inicioJuego(Scanner sc) es el encargado de gestionar el flujo principal del juego de sopa de letras. Permite al usuario jugar buscando palabras dentro de una matriz, llevando un registro de su puntaje, intentos y palabras encontradas.



```java
//Proceso que registra datos para poder realizar el historial de partidas.////////////////
public static void registrarHistorialPartida(String nombre, int puntaje, int fallos, int palabrasEncontradas) {  2 usages
    if (contadorPartidas < 20) { // Verificar espacio en contadoPartidas
        historialPartidas[contadorPartidas][0] = nombre;
        historialPartidas[contadorPartidas][1] = String.valueOf(puntaje);
        historialPartidas[contadorPartidas][2] = String.valueOf(fallos);
        historialPartidas[contadorPartidas][3] = String.valueOf(palabrasEncontradas);
        contadorPartidas++;
    } else {
        System.out.println("El historial de partidas está lleno, no se pueden registrar más.");
    }
}

// Proceso que ejecuta la opción "Historial de partidas"//////////////////
public static void mostrarHistorialPartidas() {  1 usage  new *
    if (contadorPartidas == 0) {
        System.out.println("No hay partidas registradas en el historial.");
        return;
    }

    System.out.println("--- HISTORIAL DE PARTIDAS ---");
    for (int i = 0; i < contadorPartidas; i++) {
        System.out.println("Jugador: " + historialPartidas[i][0]);
        System.out.println("Puntaje final: " + historialPartidas[i][1]);
        System.out.println("Fallos: " + historialPartidas[i][2]);
        System.out.println("Palabras encontradas: " + historialPartidas[i][3]);
        System.out.println("----------------------------\n");
    }
}
```

*Aquí podremos encontrar 2 métodos; El método **registrarHistorialPartida** tiene como propósito registrar una nueva partida en el historial del juego. Cuando un jugador finaliza una partida, sus resultados (nombre, puntaje, fallos y palabras encontradas) se almacenan en una matriz llamada historialPartidas.*

*Por consecuente, el método **mostrarHistorialPartidas** se encarga de mostrar en consola el historial de partidas almacenadas. Al ejecutarse, recorre los datos guardados en la matriz historialPartidas y muestra la información de cada partida, como el nombre del jugador, su puntaje, el número de fallos y las palabras encontradas.*

-------Métodos Auxiliares-------
*Dentro de la clase **EjecucionSopa**, encontraremos varios métodos auxiliares que complementan los principales. Estos métodos realizan tareas como llenar aleatoriamente la sopa de letras, ejecutar las palabras seleccionadas por el usuario dentro de la matriz, reemplazar las palabras correctas por el carácter #, y otras funciones esenciales que garantizan el correcto funcionamiento del juego.*