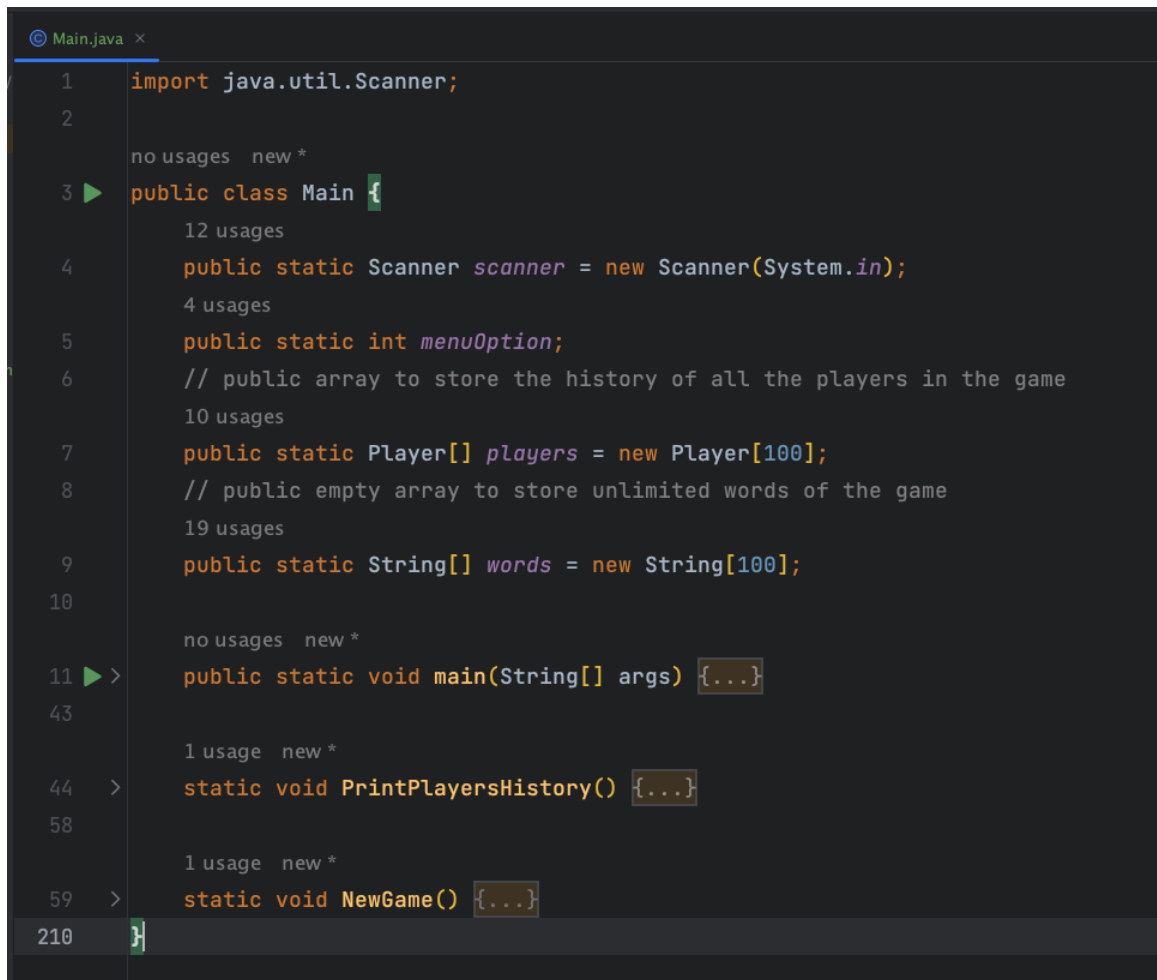


# MANUAL TÉCNICO PRACTICA 1

## 202003654

### 1. CLASE PRINCIPAL



```
1  import java.util.Scanner;
2
3  public class Main {
4      public static Scanner scanner = new Scanner(System.in);
5      public static int menuOption;
6      // public array to store the history of all the players in the game
7      public static Player[] players = new Player[100];
8      // public empty array to store unlimited words of the game
9      public static String[] words = new String[100];
10
11     public static void main(String[] args) {...}
12
13     static void PrintPlayersHistory() {...}
14
15     static void NewGame() {...}
16 }
```

- A. En Main.java se encuentran las variables principales para la función del programa completo, como el historial de jugadores, las palabras del tablero, iniciar una nueva partida y mostrar el historial

### 2. CLASE JUEGO

- A. En la clase juego se encuentran todos los algoritmos necesarios para imprimir el tablero, almacenar los puntos, encontrar las palabras en el tablero y con base a las vidas saber si la partida ya terminó

```
3 usages new *
5 public class Game {
    13 usages
6     private static int MATRIX_SIZE;
    12 usages
7     private static char[][] letterSoup; // Global variable
    1 usage
8     private static Scanner scanner = new Scanner(System.in);
    5 usages
9     private static int guessedWords = 0;
    5 usages
10    private static int points = 20;
    6 usages
11    private static int lives = 3;
    2 usages
12    private static long startTime;
    1 usage new *
13    @ > public static GameResult startGame(String[] wordsArray, int size, String playerName) {...}
    1 usage new *
76    > private static boolean isWordInMatrix(String word) {...}
    1 usage new *
118    @ > private static char[][] generateEmptyMatrix() {...}
    1 usage new *
127    @ > private static void placeWordsInMatrix(String[] words) {...}
    1 usage new *
145    > private static boolean canPlaceWord(int row, int col, int direction, int wordLength) {...}
    1 usage new *
169    @ > private static void placeWord(int row, int col, int direction, String word) {...}
    1 usage new *
179    > private static void fillMatrixWithRandomLetters() {...}
    1 usage new *
189    > private static void printMatrix() {...}
    1 usage new *
200    > private static String formatElapsedTime(long elapsedTime) {...}
207 }
208
```

- i. startGame: setea los valores necesarios como el nombre del jugador, el tamaño del tablero y las palabras a repartir, así como solicitar una palabra para buscarla dentro del tablero y al final la partida retornará un resultado de juego con los puntos, tiempo, palabras adivinadas, saber si ganó o no y nombre del jugador.
- ii. isWordInMatrix: buscará la palabra ingresada recibida como parámetro horizontal y verticalmente por separado y si la encuentra la reemplazará con el signo "\$" cada una de las letras.
- iii. generateEmptyMatrix: creará una matriz vacía con los parámetros establecidos en MATRIX\_SIZE.
- iv. placeWordsInMatrix: en la matriz ya creada buscará por medio de posiciones aleatorias hasta encontrar una posición en la que encaje la palabra completamente dentro de la matriz.

- v. canPlaceWord: analizará la posición, dirección y el largo de la palabra para saber si es posible colocarla en el tablero.
- vi. fillMatrixWithRandomLetters: por cada espacio vacío en la matriz colocará una letra al azar.
- vii. printMatrix: Imprimirá la matriz actual junto con las vidas y puntos del jugador.
- viii. formatElapsedTime: convertirá el tiempo transcurrido en horas:minutos:segundos para agregarlo al resultado del juego

### 3. CLASE RESULTADO DEL JUEGO

```

4 usages new *
public class GameResult {
    2 usages
    private int points;
    2 usages
    private int guessedWords;
    2 usages
    private int lives;
    2 usages
    private int timesFailed;
    2 usages
    private boolean won;
    2 usages
    private String time;
    1 usage new *
    > public GameResult(int points, int guessedWords, int lives, int timesFailed, boolean won, String time) {...}
    1 usage new *
    > public int getPoints() { return points; }
    1 usage new *
    > public int getGuessedWords() { return guessedWords; }
    no usages new *
    > public int getLives() { return lives; }
    2 usages new *
    > public int getTimesFailed() { return timesFailed; }
    1 usage new *
    > public boolean getWon() { return won; }
    1 usage new *
    > public String getTime() { return time; }
}

```

- A. Este objeto guarda la información de una partida, que es lo que retorna al terminar la misma
  - i. El resultado de juego como parámetros recibidos cuenta con los puntos, las palabras encontradas, las vidas restantes, las veces falladas, si el jugador completó la sopa de letras o perdió y el tiempo transcurrido de su partida

## 4. CLASE JUGADOR

- A. Este objeto se construye con base a la información requerida para mostrar en el historial de partidas guardando el nombre, la puntuación, las veces falladas, las palabras adivinadas y el tiempo que duró la partida

```
4 usages
public class Player {
    2 usages
    private String name;
    2 usages
    private int score;
    2 usages
    private int timesFailed;
    2 usages
    private int wordsGuessedCount;
    2 usages
    private String time;
    // constructor
    1 usage
    > public Player(String name, int score, int timesFailed, int wordsGuessedCount, String time) {...}
    1 usage
    > public String getName() { return name; }
    1 usage
    > public int getScore() { return score; }
    1 usage
    > public int getTimesFailed() { return timesFailed; }
    1 usage
    > public int getWordsGuessedCount() { return wordsGuessedCount; }
    1 usage
    > public String getTime() { return time; }
}
```

## 5. CLASE MENÚS

- A. Aquí se encuentran los textos de cada uno de los menús disponibles a lo largo del programa

```
7 usages
public class Menu {
    2 usages
    > public static void mainMenu() {...}
    2 usages
    > public static void newGameMenu() {...}
    2 usages
    > public static void modifyWordsMenu() {...}
    1 usage
    > public static void studentInfo() {...}
}
```

