laberinto

Proyecto fin de curso

IES Clara del Rey

Adrián Caballero Orasio y Carlos del Valle Peláez

**INDICE**

1. [**Introducción.**](#e1)
   1. [**Descripción del proyecto.**](#e11)
   2. [**Justificación del proyecto.**](#e12)
   3. [**Requisitos para el uso de la app Laberinto.**](#e13)
2. [**Metodología.**](#e2)
   1. [**Estructura del proyecto.**](#e21)
3. [**Creación de código base en Java**](#e3)
   1. [**Point.**](#e31)
   2. [**Maze.**](#e32)
   3. [**MazeSolver.**](#e33)
   4. [**FileChars.**](#e34)
   5. [**Movements.**](#e35)
   6. [**Utils.**](#e36)
   7. [**Main.**](#e37)
4. [**Muestra parcial del laberinto en pantalla.**](#e4)
5. **INTRODUCCIÓN**
   1. **DESCRIPCIÓN DEL PROYECTO**

Para realizar nuestro proyecto de fin de curso hemos desarrollado un videojuego en Android, Laberinto.

Durante el desarrollo de este proyecto, hemos procurado simplificar al maximo los procesos y diseñar un juego lo mas optimo posible.

En cuanto a la parte de programación, el proyecto esta desarrollado en Java y Android.

* 1. **JUSTIFICACIÓN DEL PROYECTO**

Hemos optado por este proyecto para ampliar y profundizar el uso de Java y Android, reforzando de este modo los conocimientos que hemos adquirido en las asignaturas de Programacion y Android y ampliando dichos conocimientos mediante información añadida a través de internet.

* 1. **REQUISITOS PARA EL USO DE LA APP LABERINTO**

Tener instalado en el ordenador eclipse (para poder comprobar si se desea el código base en Java) y Android Studio ( para poder visualizar la app final y poder interaccionar con el juego).

Otra opción es descargar en un móvil Android dicho juego para poder visualizarlo.

1. **METODOLOGÍA**
   1. **ESTRUCTURA DEL PROYECTO**

La primera parte del proyecto será crear el código base en Java el cual, una vez realizado, mostrará un laberinto en el que el movimiento del jugador se efectuará indicando el comando por teclado.

Una vez completado, realizaremos el proyecto en Android añadiendo y optimizando el código el cual tendrá algunas mejoras tales como un menú inicial para introducir el nombre, laberinto que se desea ejecutar y se cambiarán los controles por una cruceta.

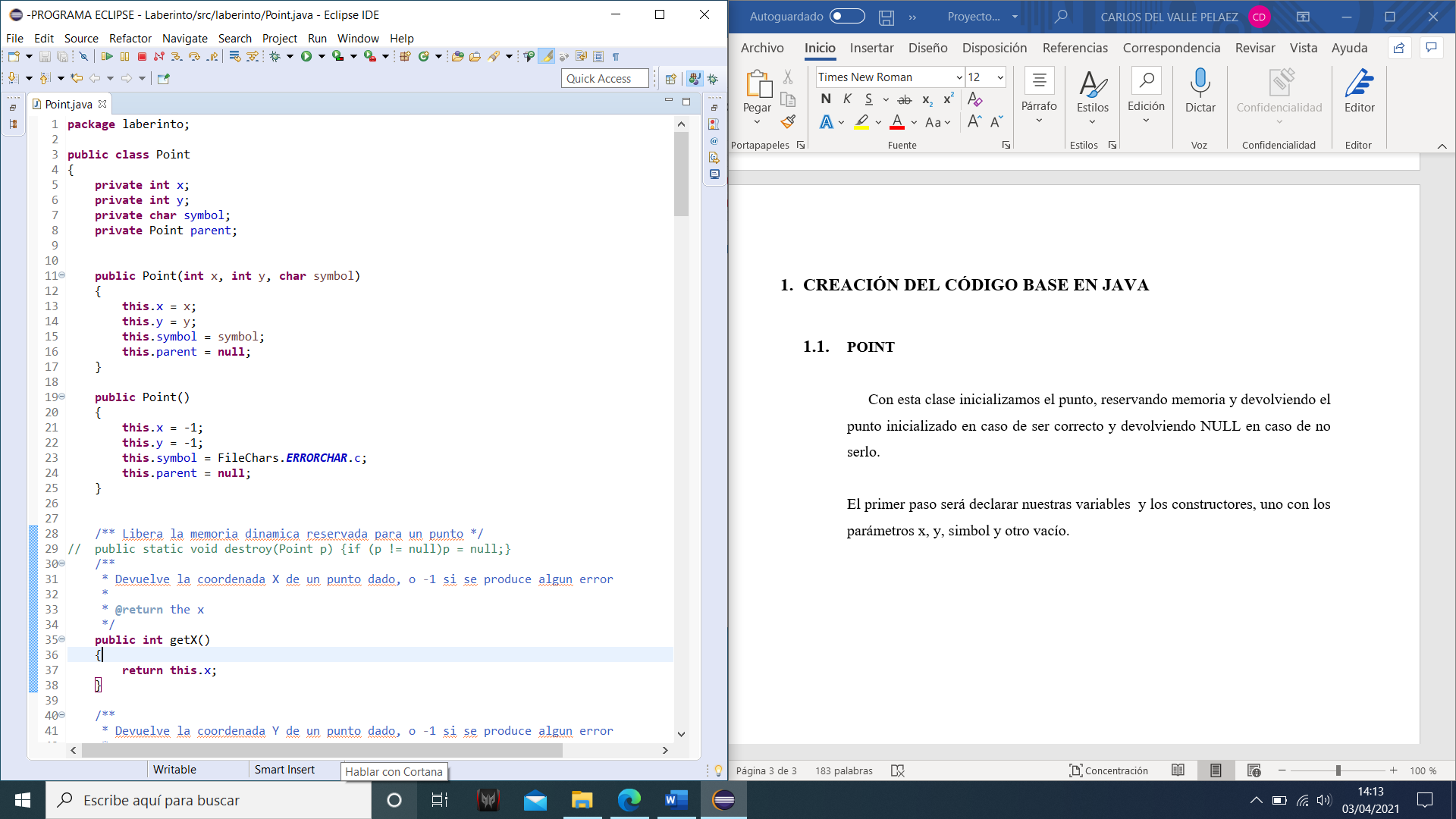
La segunda versión será seguir desarrollando y optimizando el código en Android. Podremos seleccionar el nivel y como queremos visualizar el laberinto, en formato texto o en formato imágenes ( añadiendo obstáculos y el emoticono del jugador, el cual se cambiará más adelante), tendremos unos botones de ayuda e instrucciones.

En la versión final nuestro objetivo es depurar y optimizar pequeños fallos o mejoras que deseemos implantar en el juego.

1. **CREACIÓN DEL CÓDIGO BASE EN JAVA**
   1. **POINT**

Con esta clase inicializamos el punto, reservando memoria y devolviendo el punto inicializado en caso de ser correcto y devolviendo NULL en caso de no serlo.

El primer paso será declarar nuestras variables y los constructores, uno con los parámetros x, y, simbol y otro vacío.



**3.1. Imagen variables y constructor clase Point**

A continuación crearemos los **getters** y **setters**, en donde **getX** y **getY** devolverán la coordenada X o Y de un punto dado o -1 si se produce algún error y **setX** y **setY** modificarán la coordenada X o Y de un punto dado, devolviendo error si se produce algún fallo.

Dentro de esta clase encontraremos también una serie de funciones derivadas que indicarán el tipo de punto que tenemos, estas funciones serán **isInput()**, **isOutput()**, **isBarrier()**, **isSpace()** e **isActual().**

La función **copy()** se encargará de, si todo ha ido bien, copiar los datos de un punto a otro devolviendo el punto copiado( incluyendo la reserva necesaria de memoria), en caso contrario devolverá NULL.

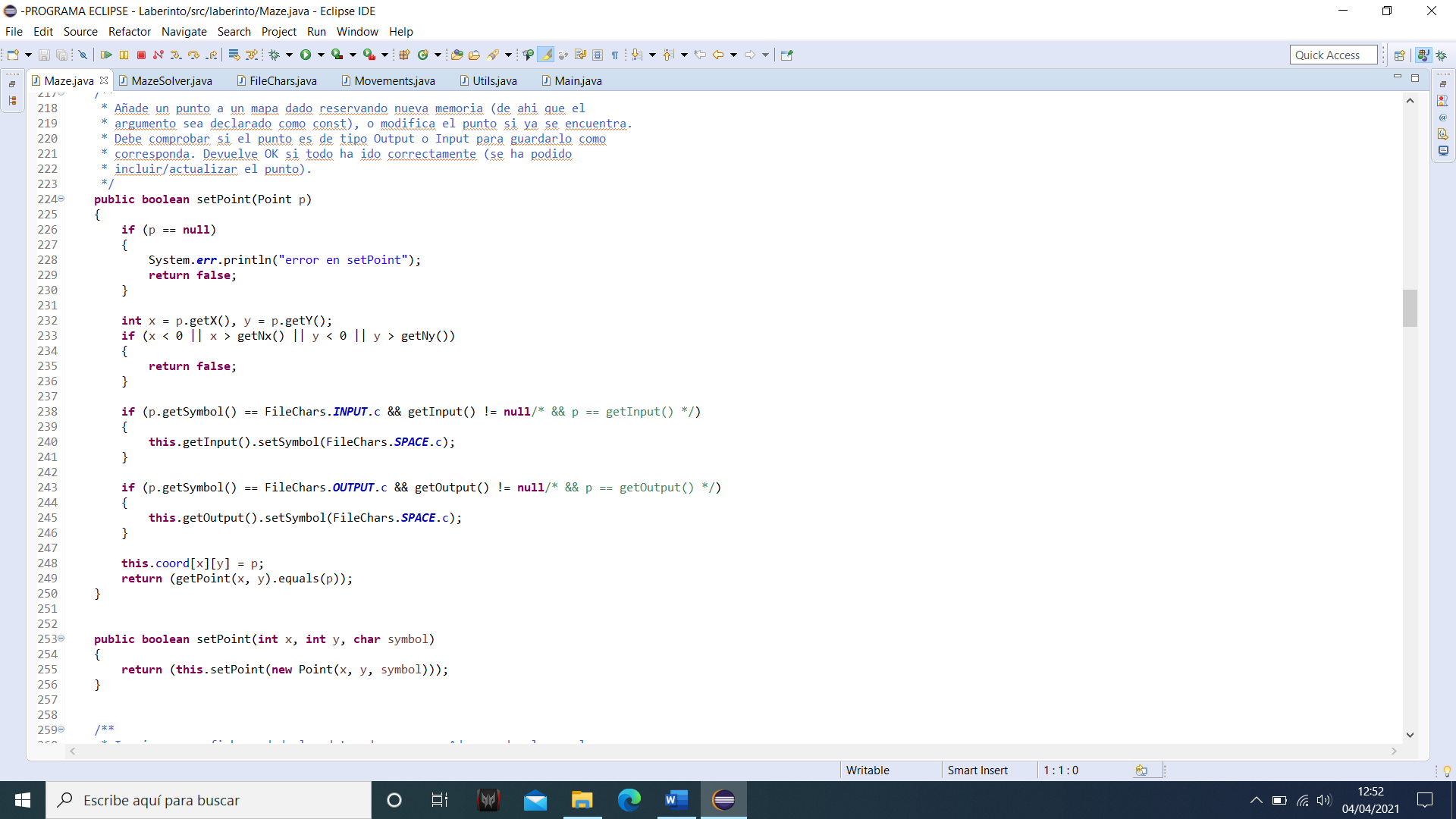
* 1. **MAZE**

La clase **Maze.java** sirve para crear el laberinto. Dentro de ella encontraremos su constructor, el cual inicializa un mapa reservando memoria y devolviendo el mapa inicializado en el caso de que se haya hecho correctamente, siendo NULL en caso contrario.

Dentro de esta clase encontraremos las funciones **getInput()** y **getOutput()**, las cuales se encargarán de devolver el punto de entrada o salida en un mapa dado.

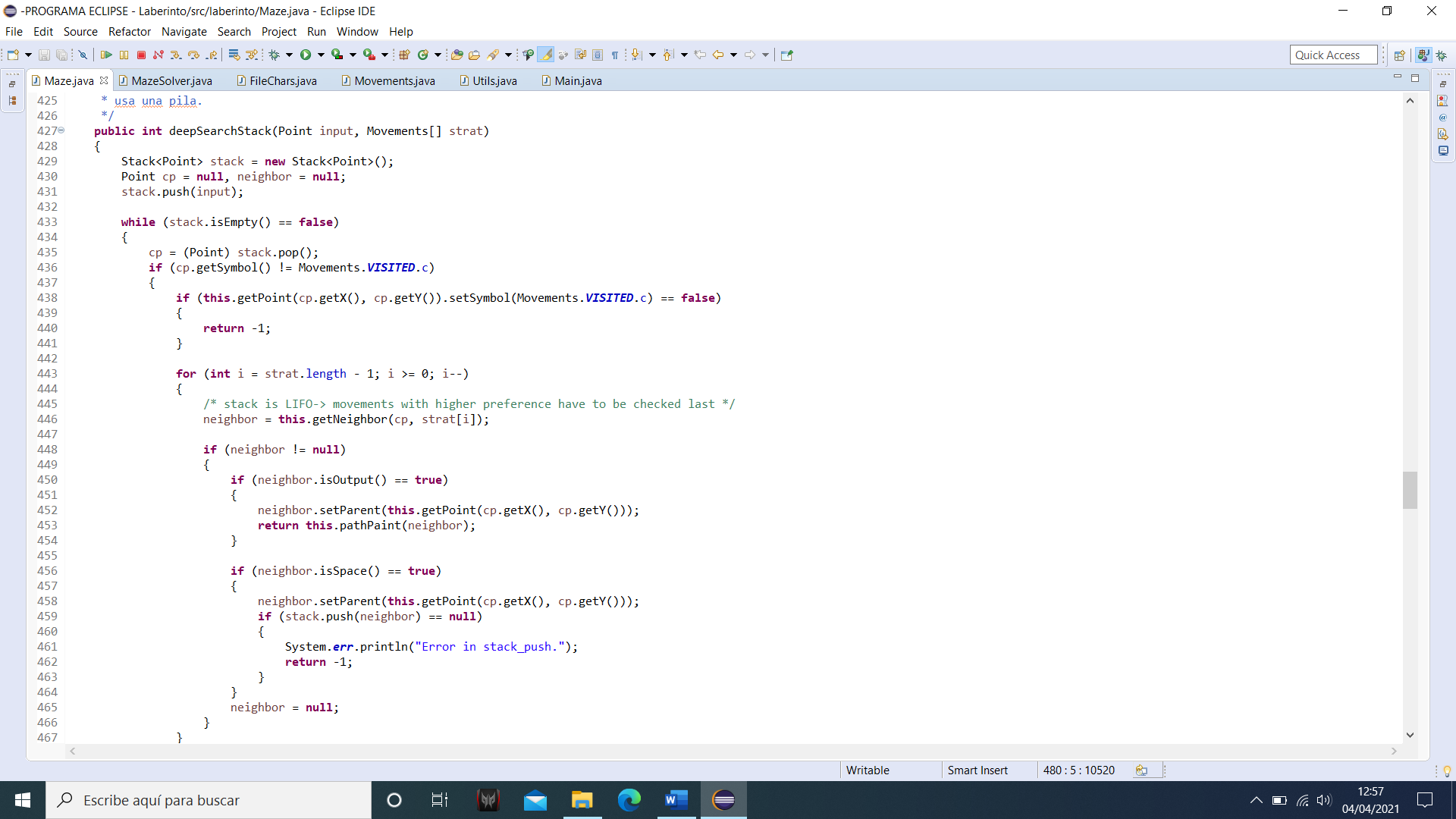
Tendremos también **getNeighbor()**, que devolverá el punto resultante al realizar un movimiento en un mapa a partir de un punto inicial.

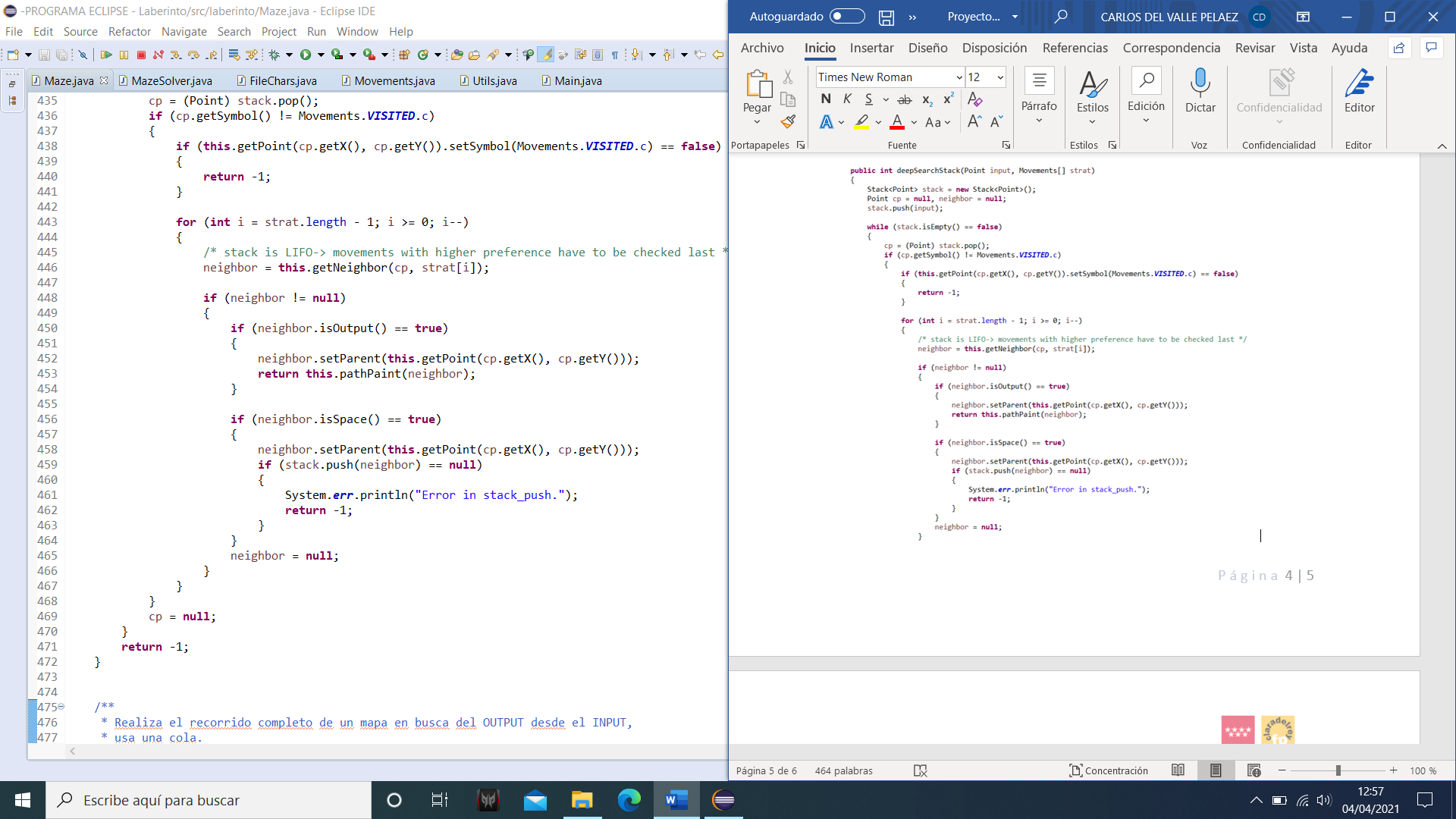
La función **setPoint()** añade un punto a un mapa dado reservando nueva memoria o modifica el punto si ya se encuentra. Debe comprobar de que tipo es el punto, Output o Input, para guardarlo como corresponda.



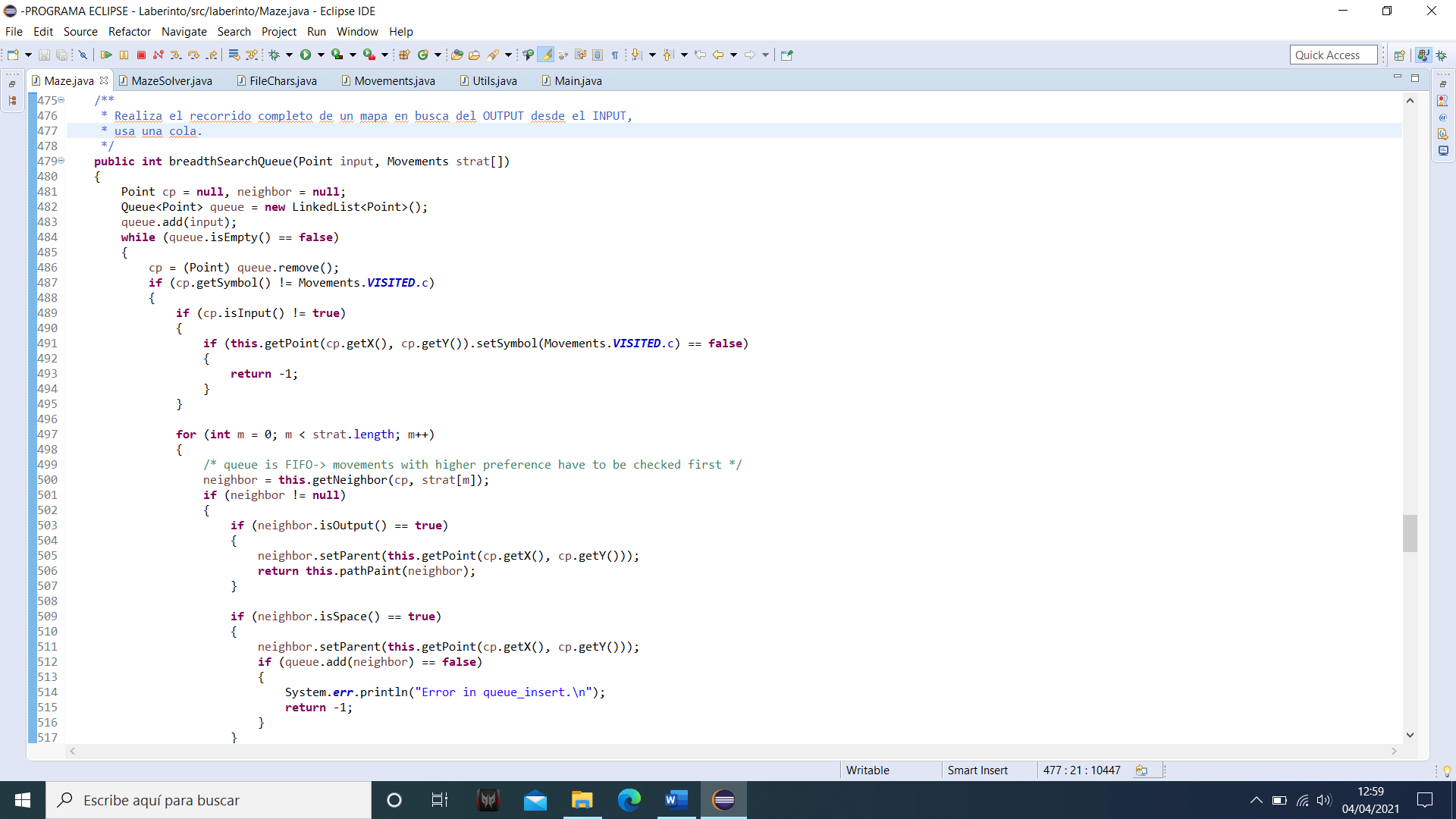
**3.2. Imagen función setPoint clase Maze**

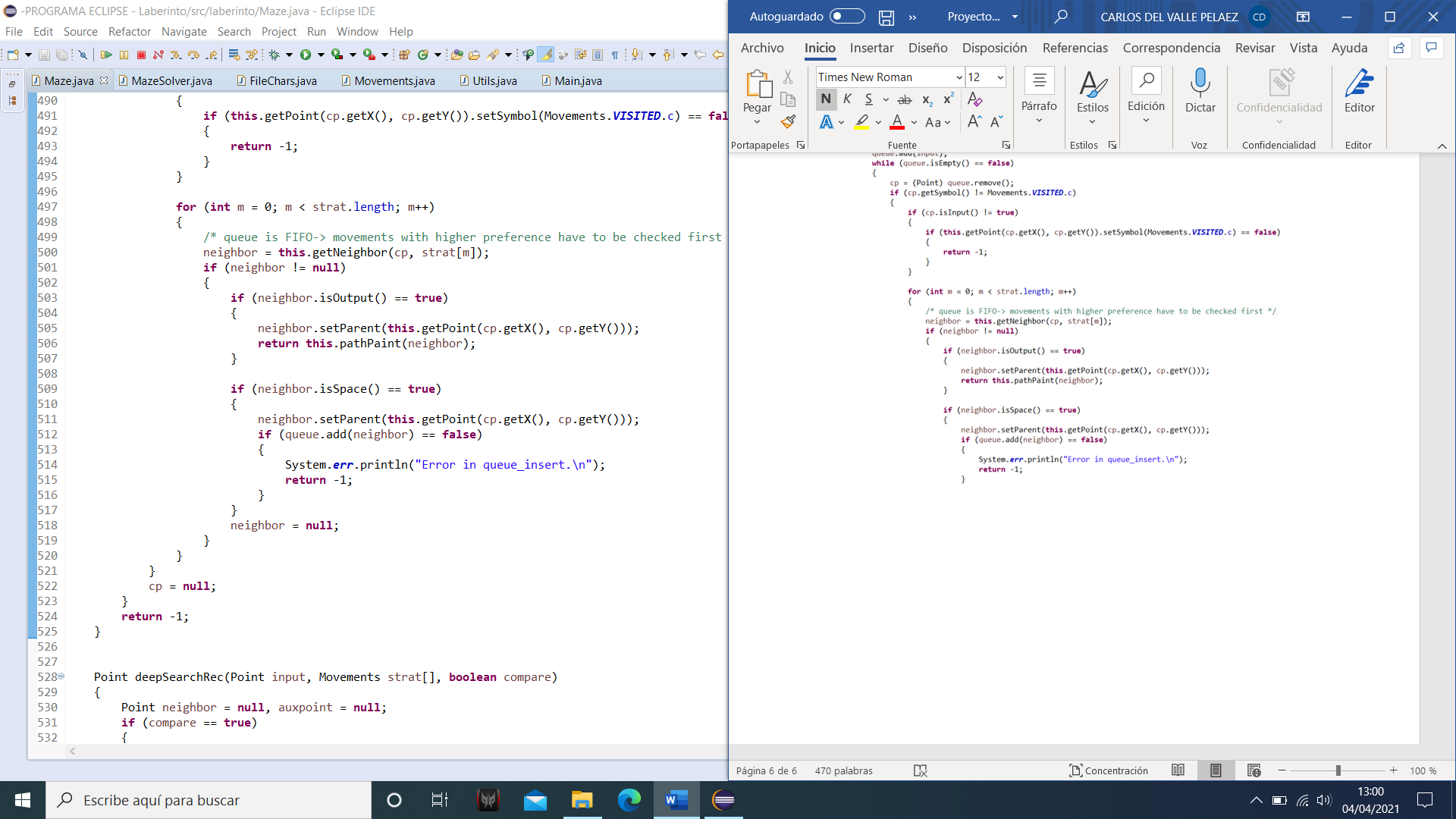
Las funciones mas importantes dentro de esta clase son **deepSearchStack()**, **breadthSearchQueue()** y **deepSearchRec()**, todas ellas realizan el recorrido completo de un mapa en busca del Output desde el Input, usando una pila.



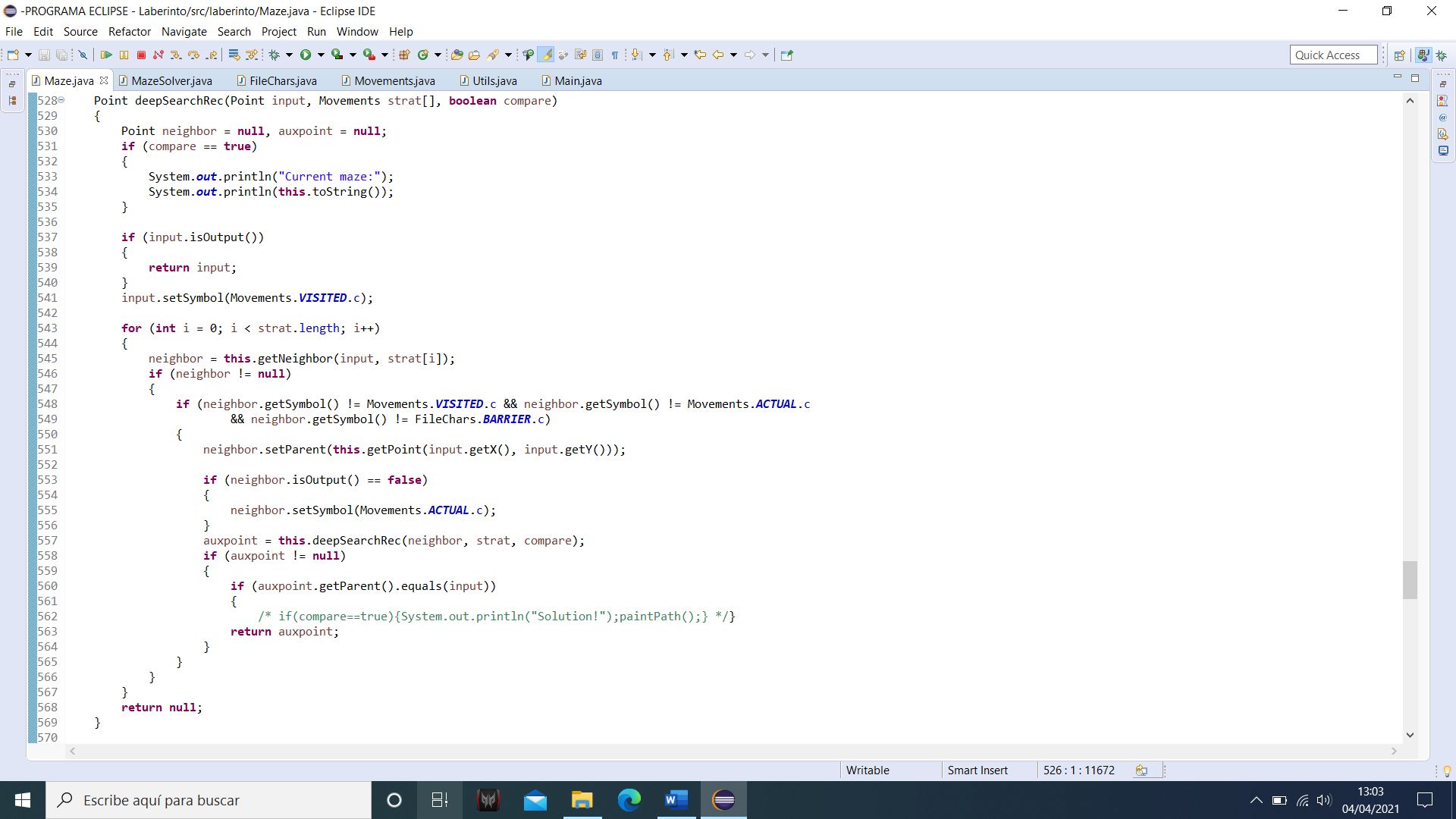


**3.2. Imagen función deepSearchStack clase Maze**





**3.2. Imagen función breadthSearchQueue clase Maze**

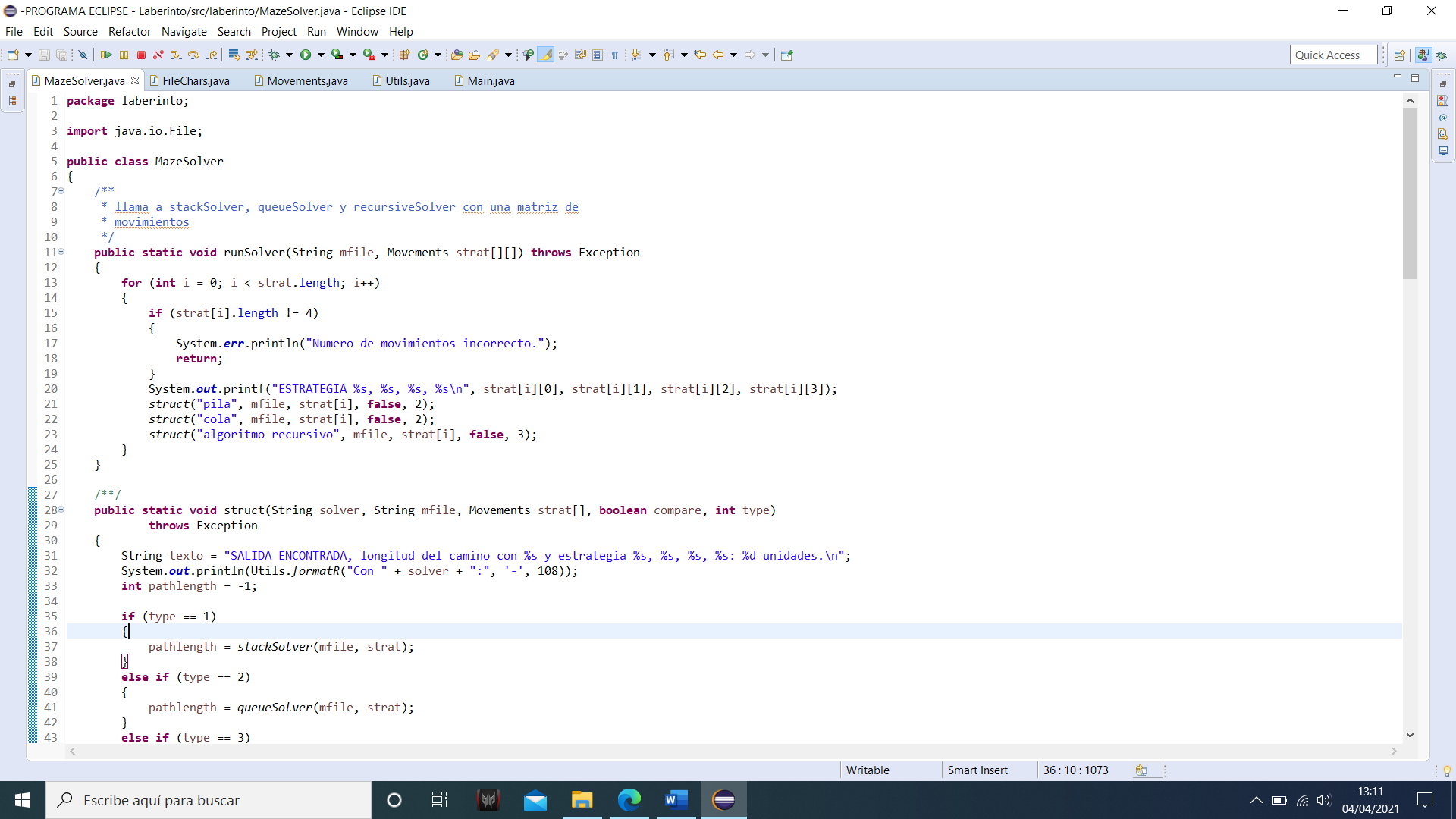


**3.2. Imagen función Point deepSearchRec clase Maze**

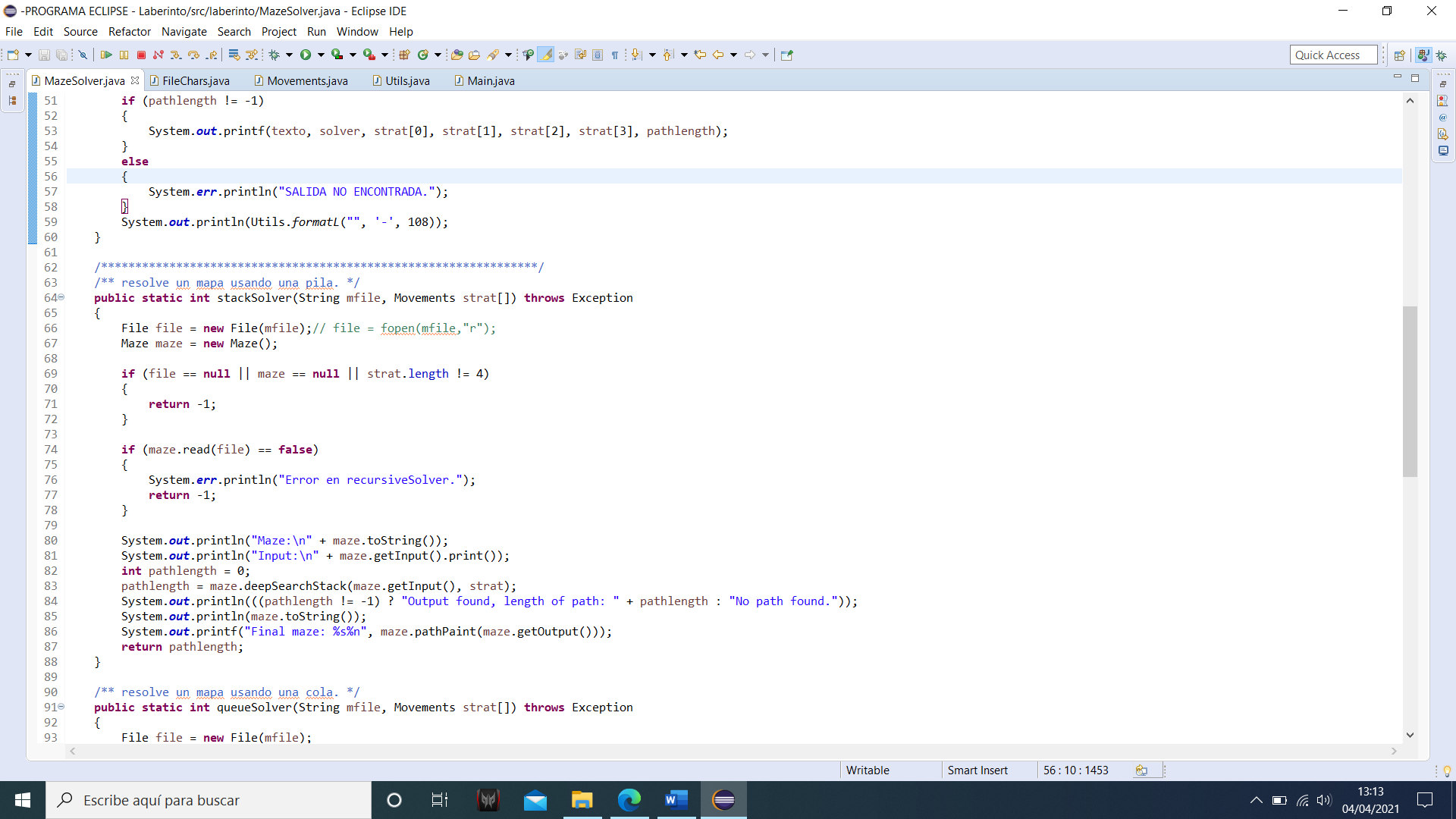
* 1. **MAZESOLVER**

La clase **Mazesolver.java** se encargará de comprobar que se puede llegar a la salida.

Podemos encontrar dentro de ella el método **runSolver(),** el cual llama a **stackSolver**, **queueSolver** y **recursiveSolver** con una matriz de movimientos.

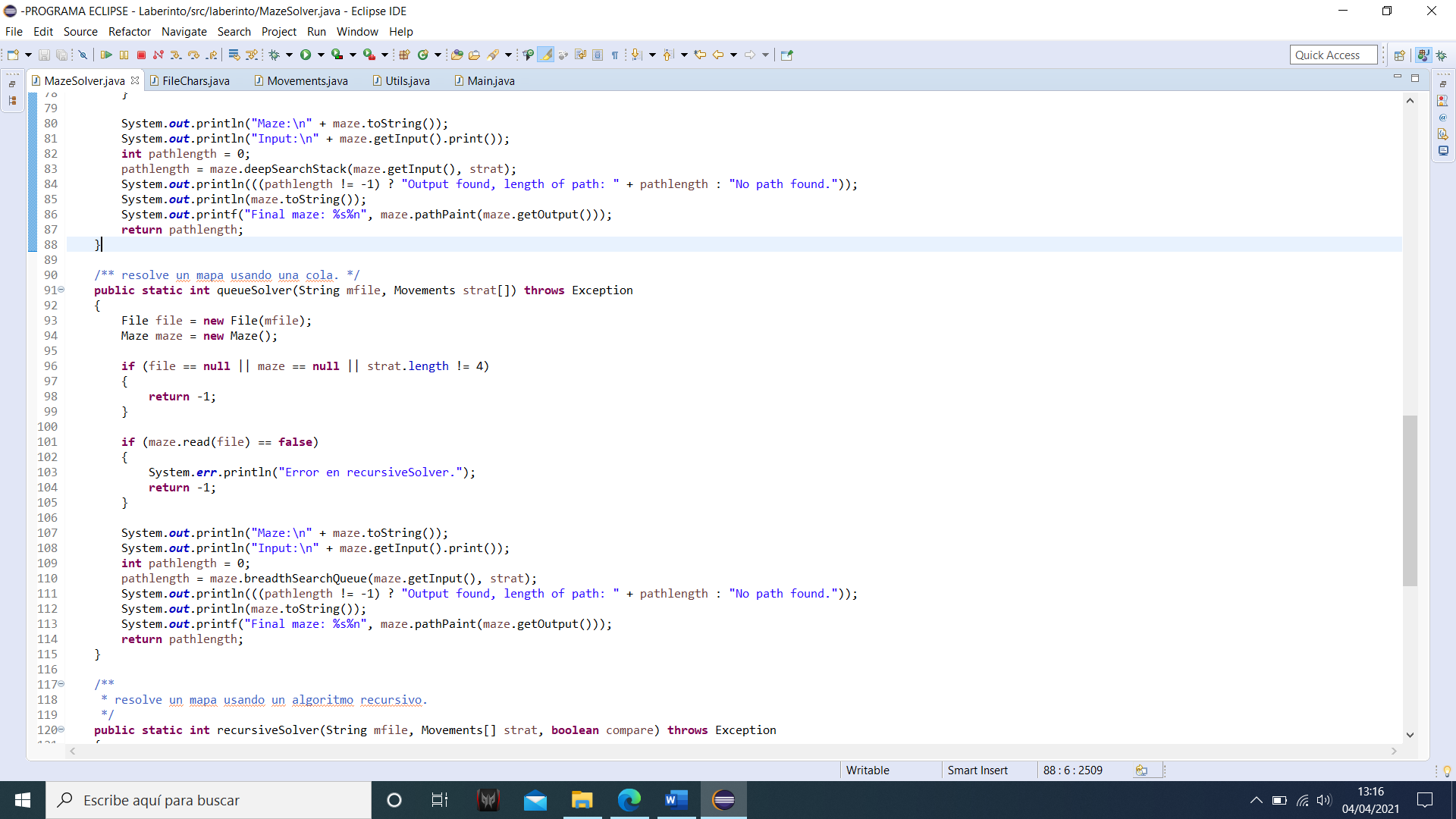


**3.3. Imagen método runSolver clase MazeSolver**

La función **stackSolver()** se encarga de resolver un mapa usando una pila. 

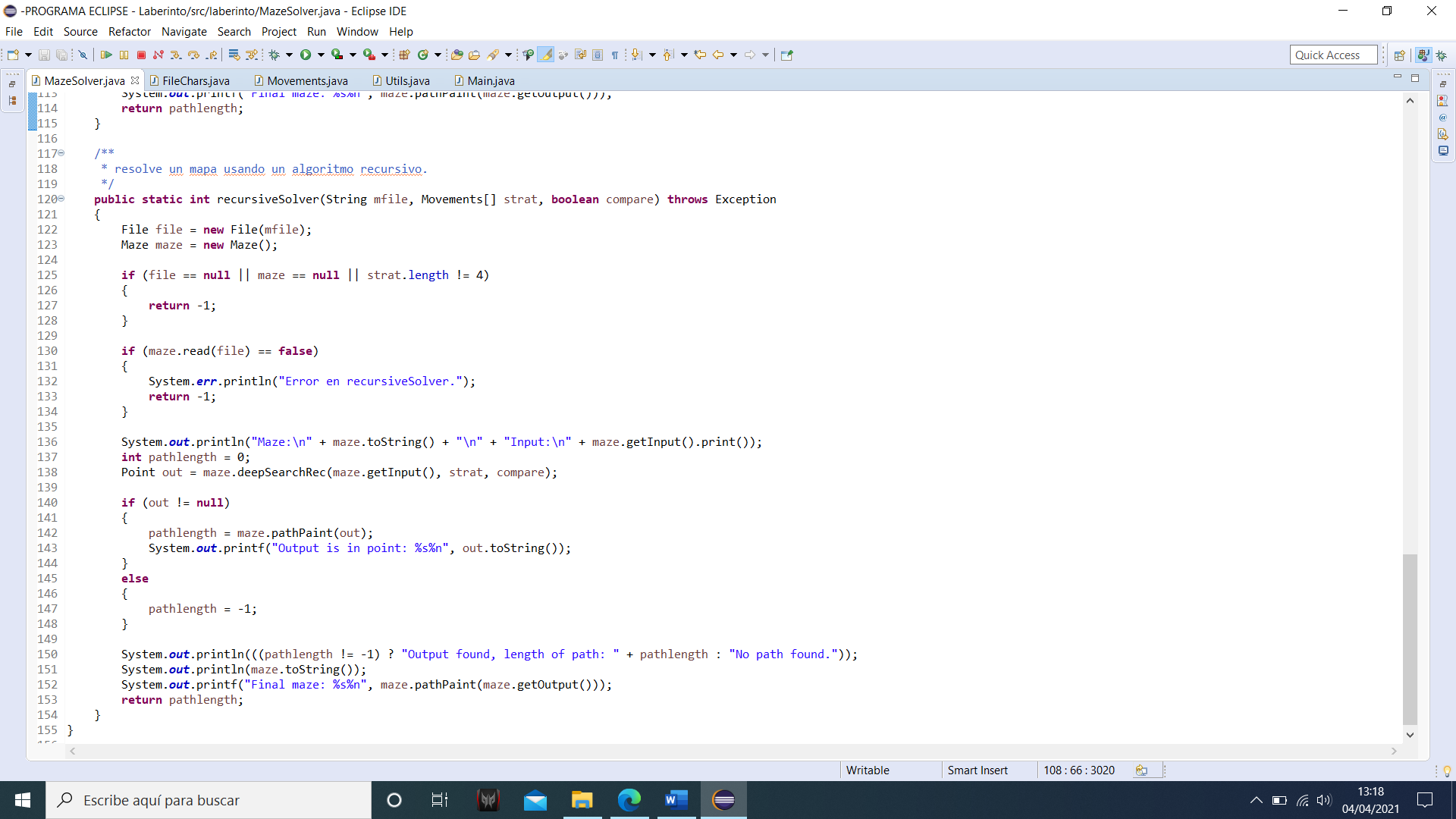
**3.3. Imagen función stackSolver() clase MazeSolver**

La función **queueSolver()**  se encarga de resolver un mapa usando una cola.



**3.3. Imagen función queueSolver clase MazeSolver**

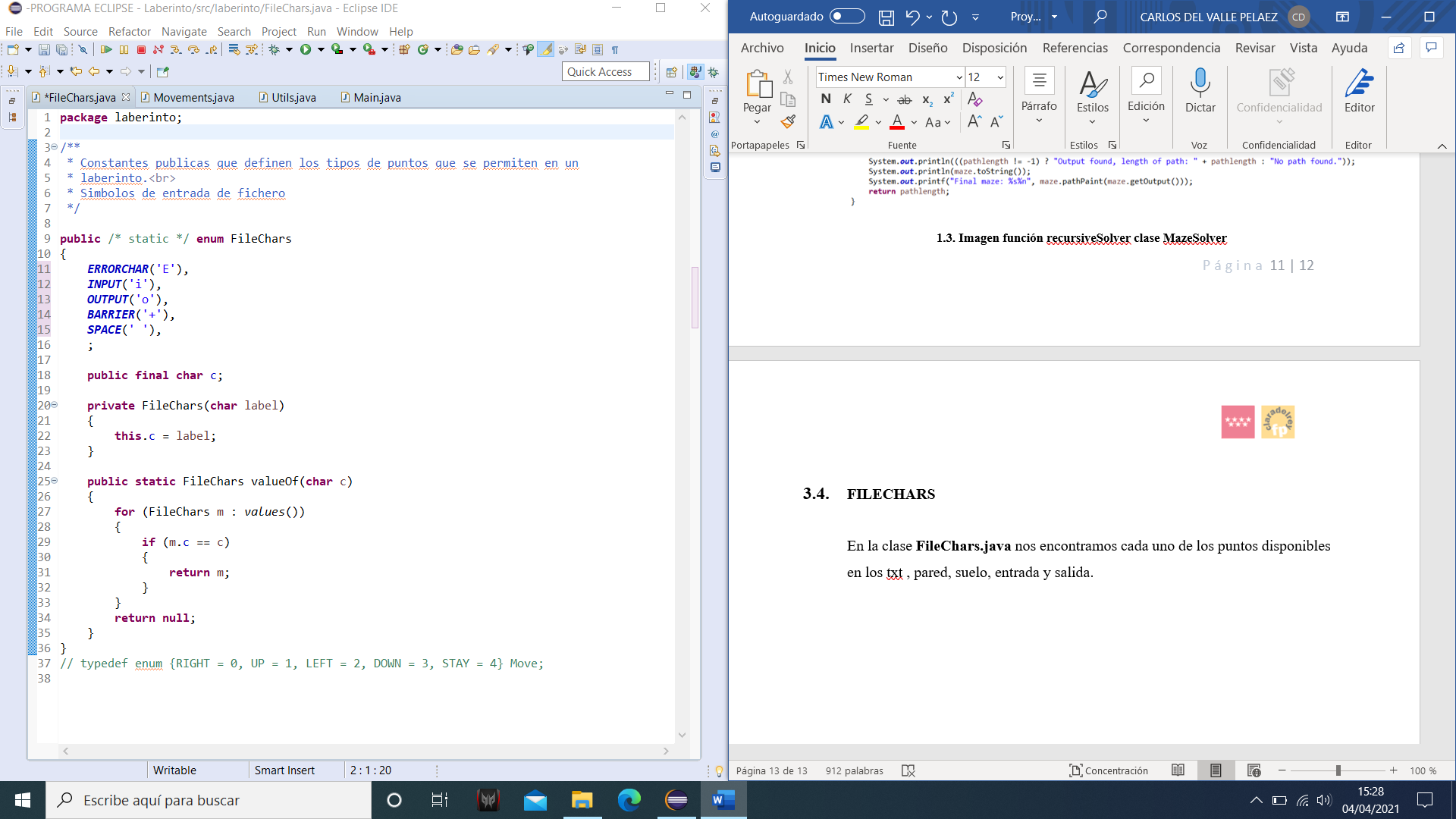
La función **recursiveSolver()** se encarga de resolver un mapa usando un algoritmo recursivo.



**3.3. Imagen función recursiveSolver clase MazeSolver**

* 1. **FILECHARS**

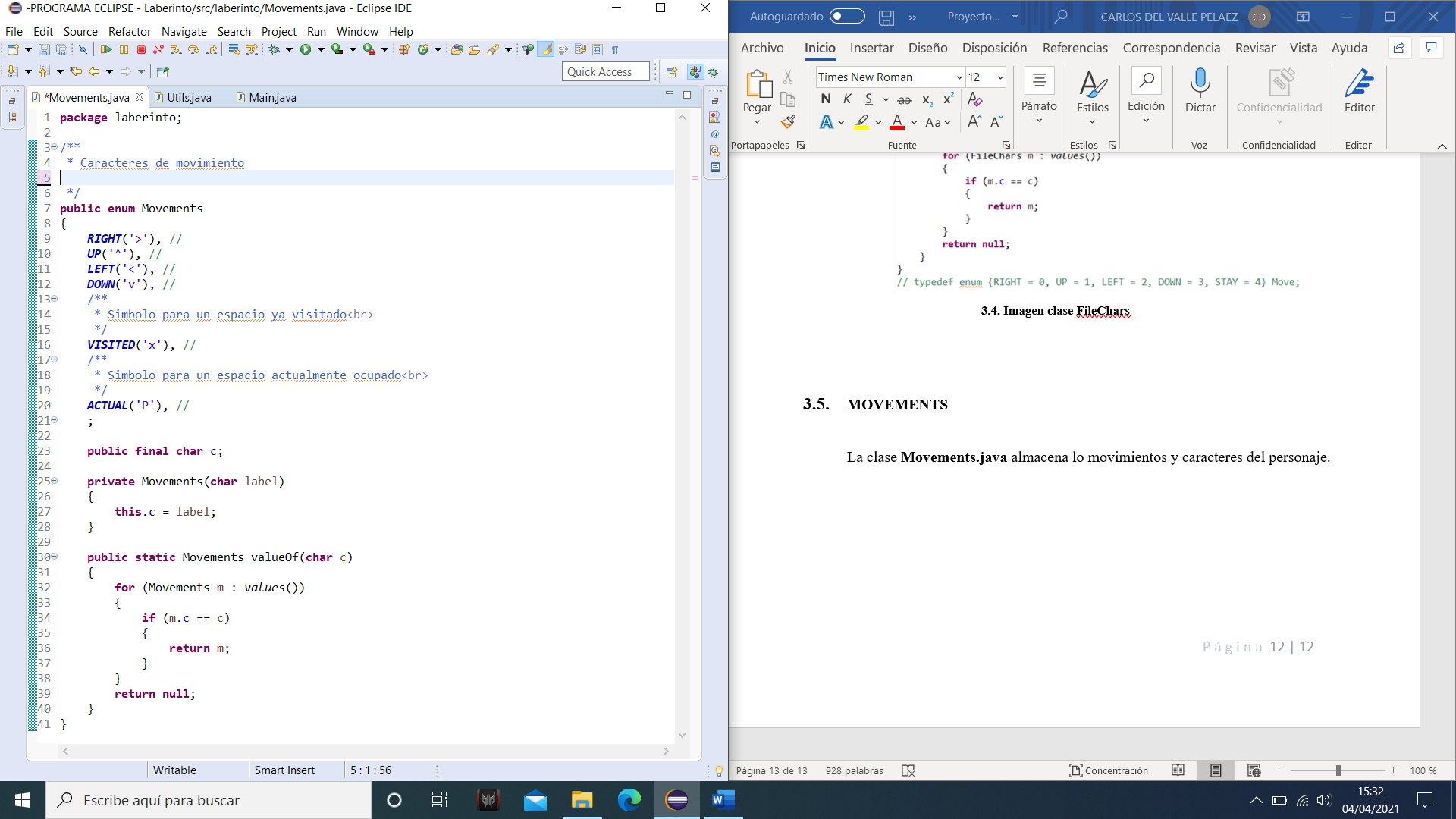
En la clase **FileChars.java** nos encontramos cada uno de los puntos disponibles en los txt , pared, suelo, entrada y salida.



**3.4. Imagen clase FileChars**

* 1. **MOVEMENTS**

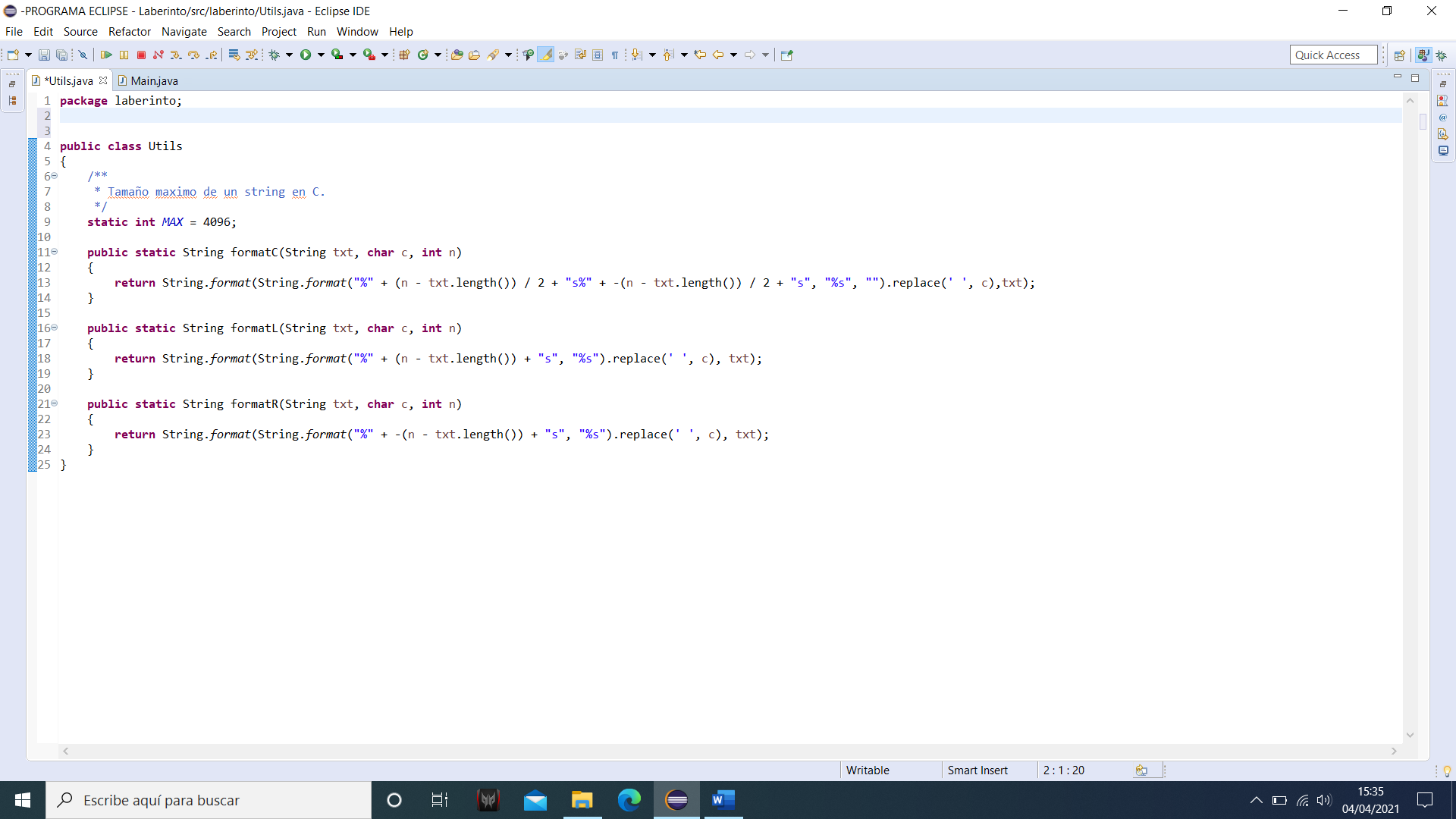
La clase **Movements.java** almacena lo movimientos y caracteres del personaje.



**3.5. Imagen clase Movements**

* 1. **UTILS**

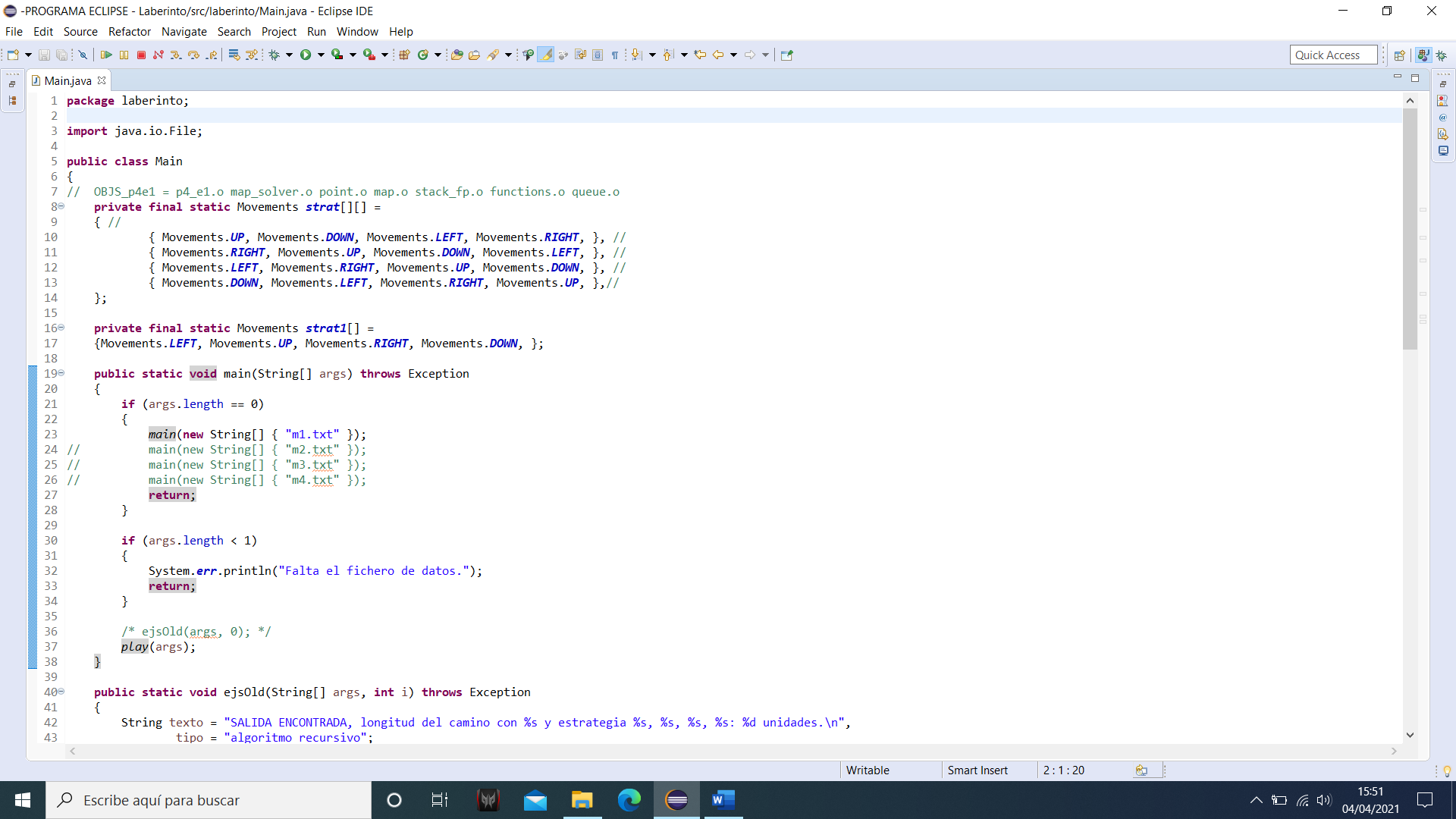
La clase **Utils.java** contiene el limite que creamos para la lectura de los txt, consiguiendo de este modo que tengan un maximo a la hora de crearlo.



**3.6. Imagen clase Utils**

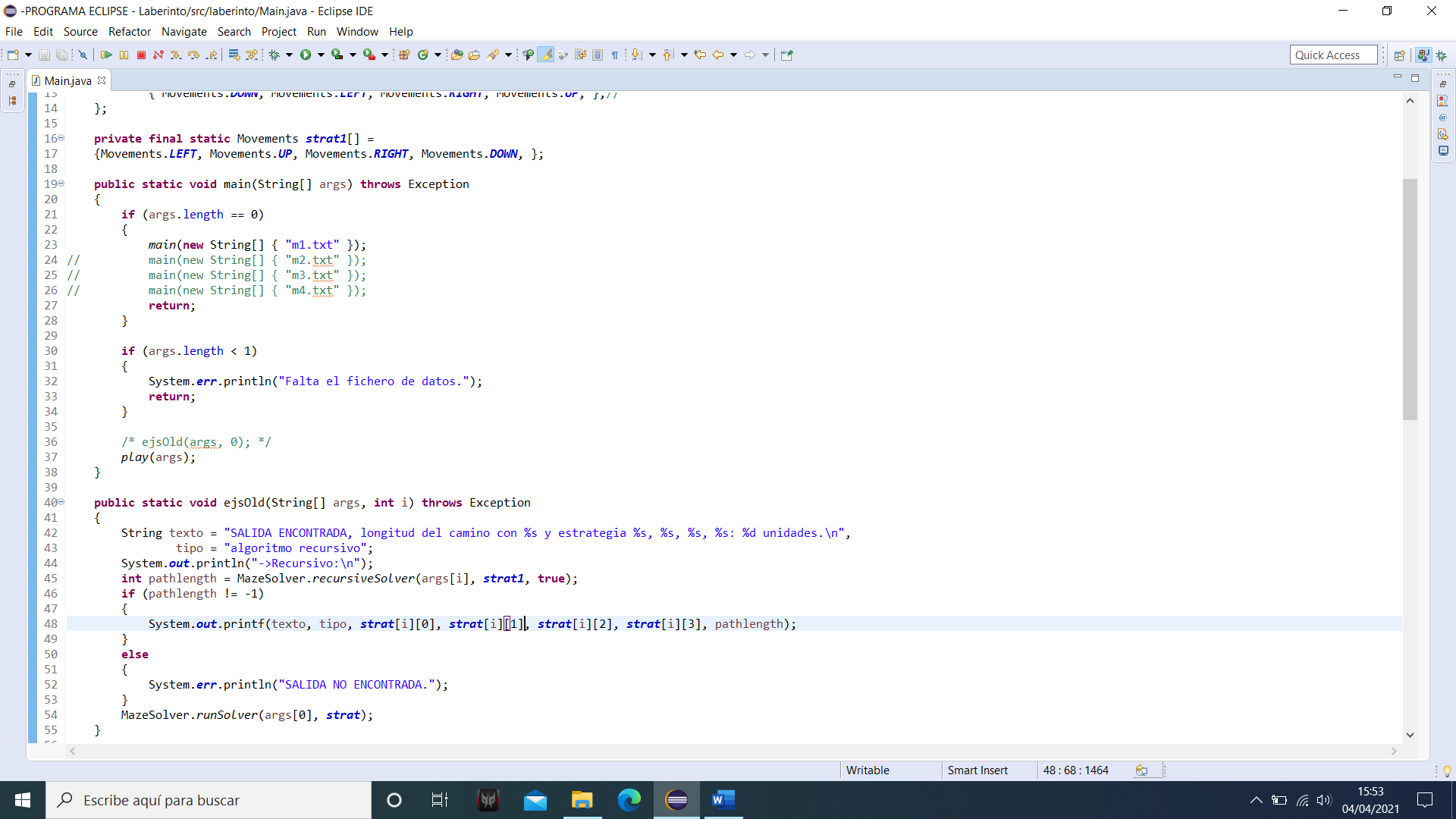
* 1. **MAIN**

La clase Main.java nos permitirá ejecutar nuestro programa. En ella encontraremos en primer lugar una llamada a la clase Movements.java, la cual nos permitirá ejecutar los movimientos de nuestro jugador.

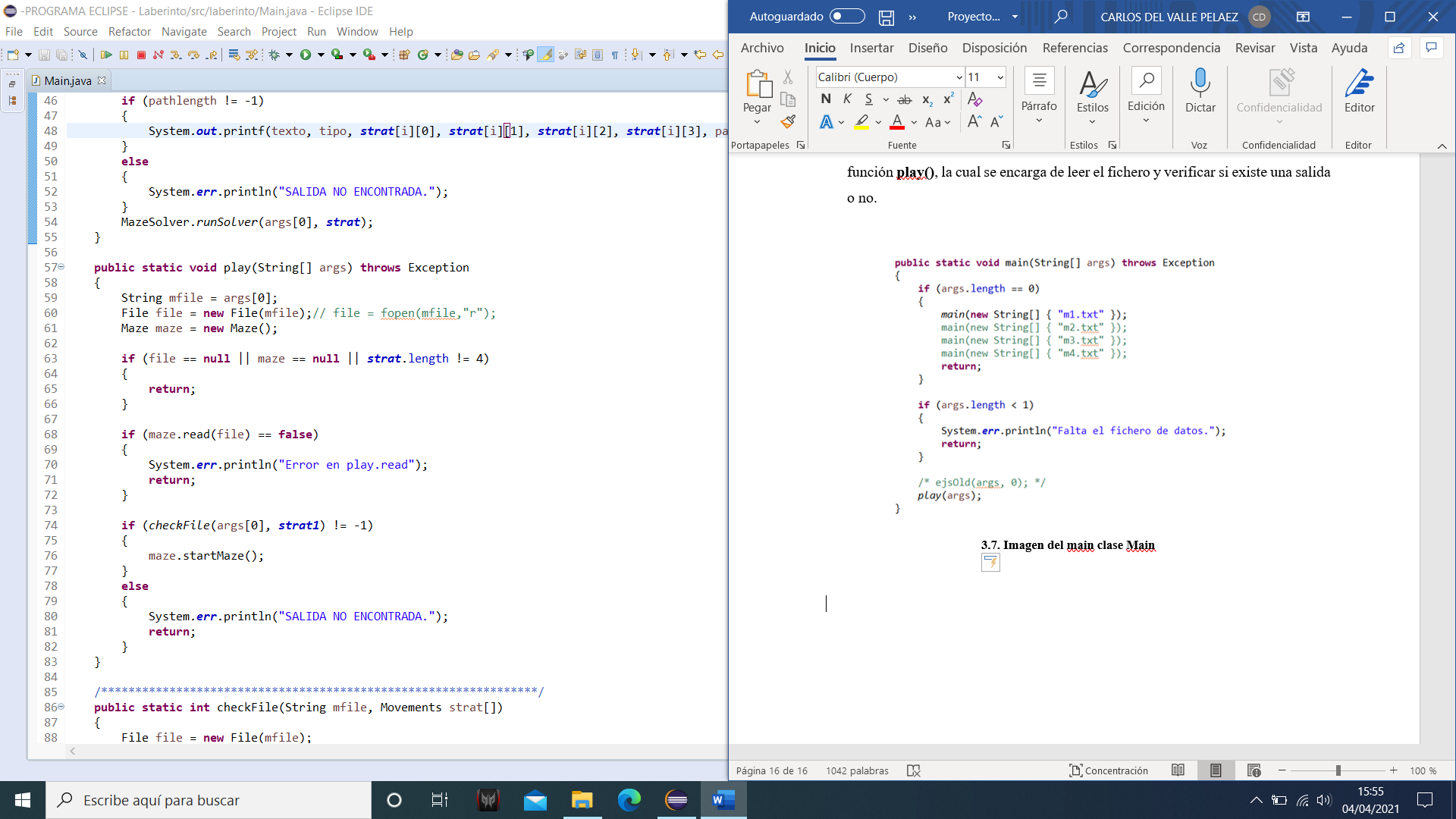


**3.7. Imagen del método Movements clase Main**

Dentro del main tendremos la ejecución de nuestro fichero .txt y llamamos a la función **play()**, la cual se encarga de leer el fichero y nos mostrará los controles posibles a introducir y, de ser correcto el control introducido, ejecutará el movimiento del personaje, esto lo realiza con **maze.startMaze()**, que pertenece a la clase **Maze.java**.



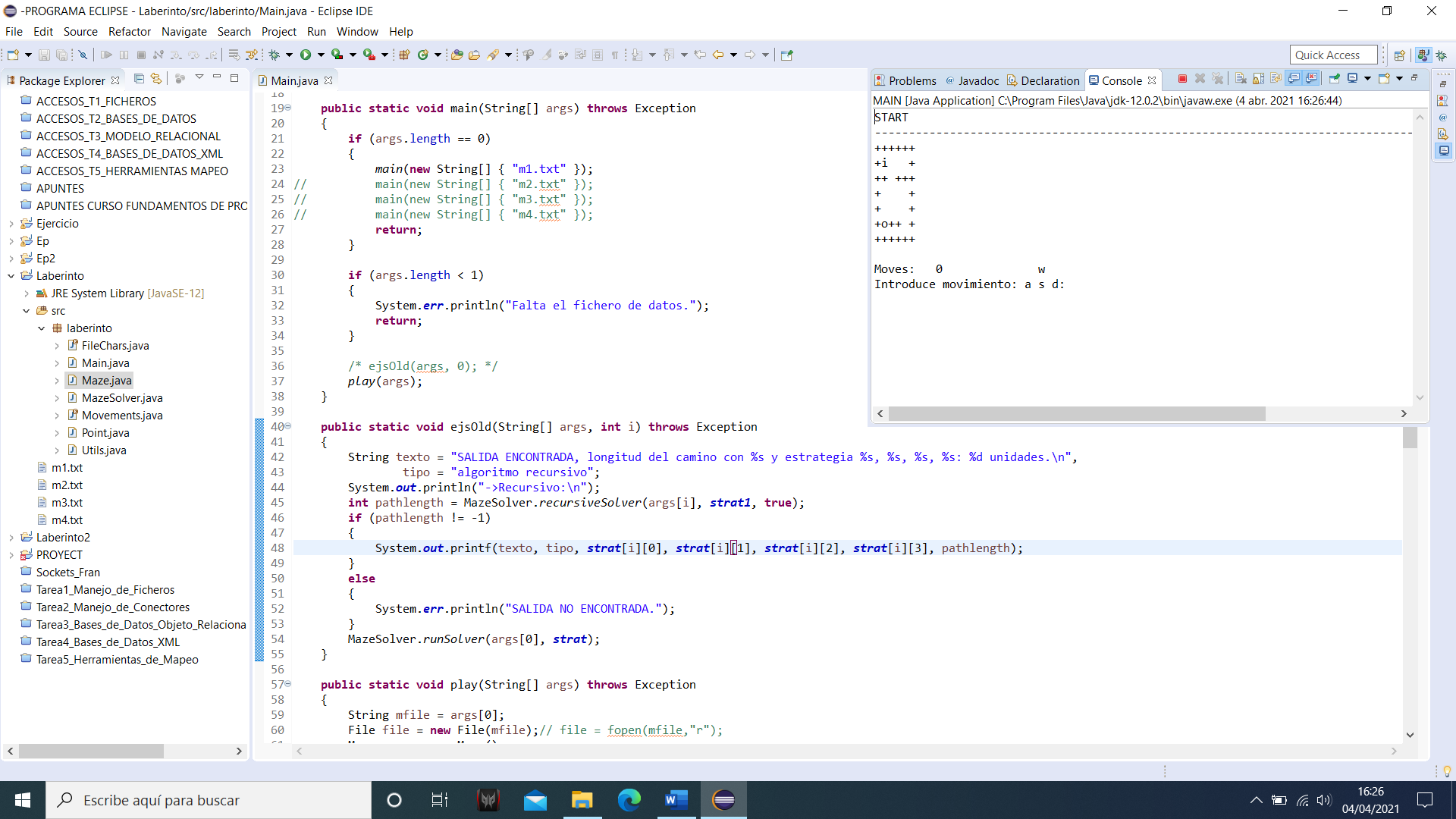
**3.7. Imagen del main clase Main**



**3.7. Imagen de la función play clase Main**

1. **MUESTRA PARCIAL DEL LABERINTO EN PANTALLA**

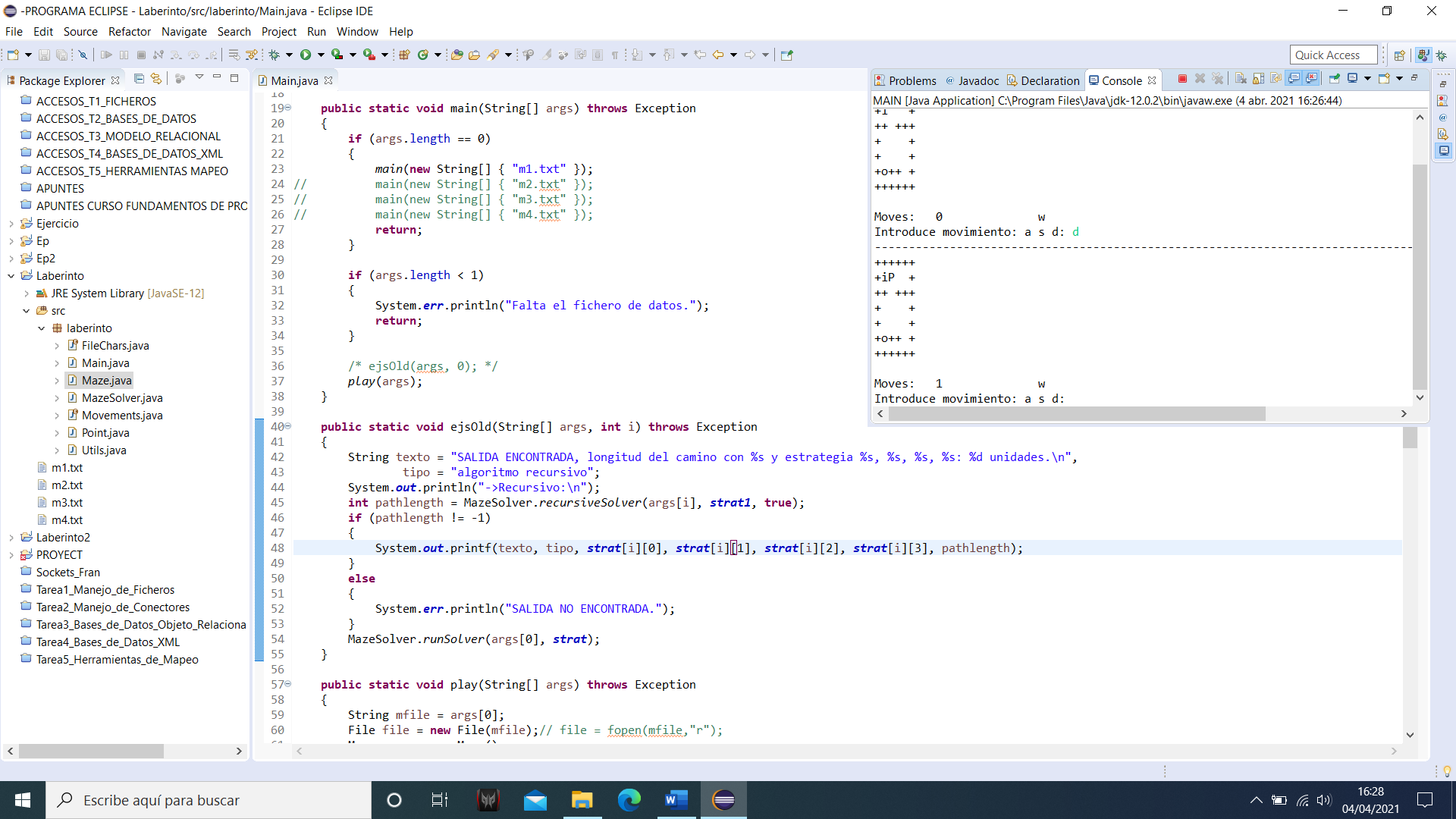
A continuación, veremos una visualización inicial de la ejecución de nuestro programa en Java.



**4. Imagen terminal clase Main**

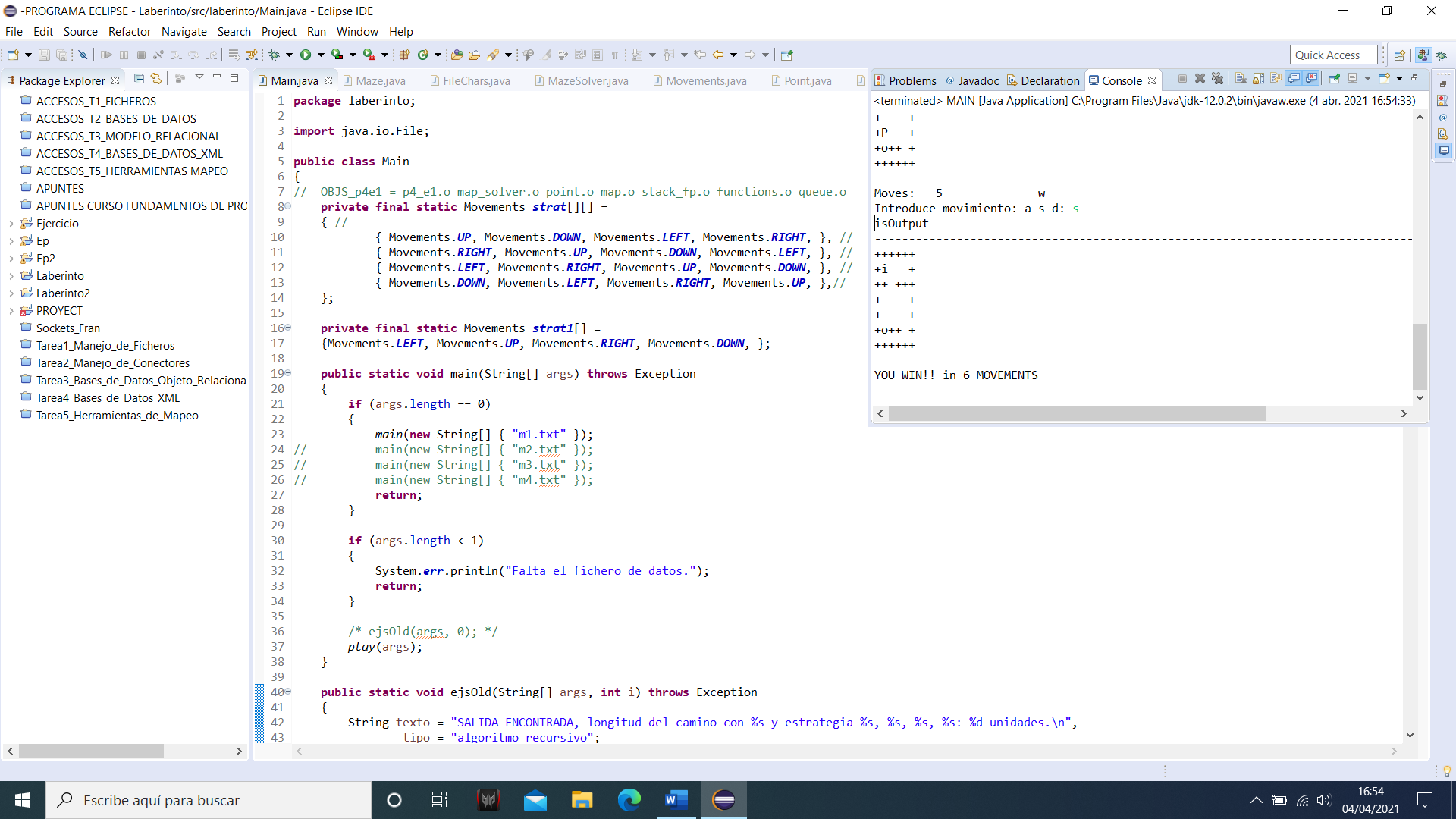
En la siguiente imagen podemos observar que, al introducir el movimiento **d**, nuestro personaje (**P**) se mueve a la derecha.

Una vez realizado el movimiento, cuenta dicho movimiento y vuelve a imprimir la instrucción para insertar un nuevo movimiento.



**4. Imagen terminal clase Main**

En la siguiente imagen podemos observar que, una vez que el personaje llega al punto o, finaliza el programa, mostrando un mensaje y el numero de movimientos realizados en total.



**4. Imagen terminal clase Main**