Juego del laberinto

Alejandro Rodríguez Arguimbau /
Sergi Mayol Matos /
Enlace video: https://youtu.be/MQpI1R5VHSs

Programación II

Juan Antonio Montes de Oca Durán

Índice

1. Introducción

En esta práctica se nos propone diseñar el típico juego de un laberinto en Java, con la diferencia de que este se genera a partir de unos datos. Estos datos consisten en 4 ficheros diferentes, que representan un mapa del laberinto cada uno. Todos los ficheros tienen la misma estructura, la primera línea representa las filas y la segunda las columnas. Después viene el mapa, que consiste en una serie de 1s y 0s, los 1s representan las paredes y los 0s los caminos disponibles para la ficha. Finalmente, las dos últimas líneas representan en qué fila y columna se encuentra la salida.

Para realizar esta práctica utilizaremos la Programación Orientada a Objetos (POO), ya que nos permite modelar de una manera sencilla los datos y su comportamiento. Si podemos manejar independientemente los datos de cada objeto, evitamos tener que mantener datos globales y coordinarlos.

Por tanto, utilizaremos 5 clases para poder aplicar la POO. Estas clases son las siguientes:

- Clase Laberinto: En esta clase se encuentran todos los atributos y métodos relacionados con el laberinto.
- Clase Casilla: En esta clase se encuentran todos los atributos y métodos relacionados con las casillas.
- Clase Mapa: En esta clase se encuentran todos los atributos y métodos relacionados con el mapa.
- Clase FicheroLecturaMapa: En esta clase se encuentran todos los atributos y métodos relacionados con la lectura del fichero que genera el mapa.
- Clase Ficha: En esta clase se encuentran todos los atributos y métodos relacionados con la ficha.

2. Diseño

2.1 Clase Laberinto

2.1.1 Atributos

Dentro de la clase encontramos los siguientes atributos:

```
public class Laberinto extends JFrame {

    //DECLARACIÓN OBJETO Container PARA REPRESENTAR EL PANEL DE
    //CONTENIDOS DEL OBJETO JFrame
    private Container panelContenidos;
    //DECLARACIÓN JMenuItem
    private JMenuItem seleccionLab;
    private JMenuItem reiniciar;
    private JMenuItem salir;
    //Atributo String que contiene el nombre del mapa que se va a dibujar
    private static String fichero;
    //Atributo Mapa que contiene mapa que se va a dibujar
    private Mapa mapa;
    //Atributo int que contiene el número de filas que contiene el laberinto
    private int filas;
    //Atributo int que contiene el número de columnas que contiene el laberinto
    private int columnas;
```

Todos los atributos serán privados, ya que solo se van a utilizar dentro de esta clase. Dentro de esta encontramos: un atributo Container el cual contendrá el panel de contenidos del JFrame, tres de tipo JMenuItem empleados para la gestión de opciones en el laberinto, uno de tipo String que contendrá el nombre del mapa que se va a pintar, uno de la clase Mapa y dios de tipo int para indicar las filas y columnas.

2.1.2 Constructor

Esta clase contiene un único constructor, el cual contiene las configuraciones del JFrame y el método principal, además de tener el foco principal para el movimiento de la ficha.

```
public Laberinto() {
    //Asignación del gestorDesplazamientoFicha al constructor, el cual tiene
    //el foco de atención
    addKeyListener(new gestorDesplazamientoFicha());
    setFocusable(true);

    //Se configura la ventana
    configuracionJFrame();

    //Se incia el juego
    inicio();

    //Metodo encargado de la gestion de la interface
    metodoPrincipal();

    //Activar visualización contenedor JFrame ventana
    setVisible(true);
}
```

2.1.3 Métodos

Esta clase contiene cinco métodos, dos clases internas y el main del programa: private void inicio(), private void configuracionJFrame(), private void metodoPrincipal(), public static String getFicheroNombre(), private void seleccionarLaberinto(), public class gestorDesplazamientoFicha, private class gestorEventosMenu y public static void main(String[] args).

El método private void inicio(), es el método principal del programa, el cual es encargado de ir repintando el mapa y la ficha, a medida que estos se van modificando.

```
//Método inicial del programa
private void inicio() {
    while (true) {
        dibujo.repaint();
        try {
            Thread.sleep(10);
        } catch (InterruptedException ex) {
            Logger.getLogger(Laberinto.class.getName()
            ).log(Level.SEVERE, null, ex);
        }
    }
}
```

El método private void configuracionJFrame(), es el encargado de asignar una configuración de la ventana.

```
private void configuracionJFrame() {
    //Añadimos un titulo a la ventana
    setTitle("Laberinto");
    //La ventana no podrá redimensionarse
    setResizable(false);
    //Alamacena en la variable nuestro sistema nativo de ventanas
    Toolkit pantalla = Toolkit.getDefaultToolkit();
    //Añadimos un icono a la ventana
    Image Icono = pantalla.getImage("lab.png");
    setIconImage(Icono);
    //Tamañano de la pantalla del usuario
    Dimension tampant = pantalla.getScreenSize();
    //Obtener el alto de resolución de pantalla
    int altpant = tampant.height;
    //Obtener el ancho de resolución de pantalla
    int anchopant = tampant.width;
    //Localización(x,y) + tamaño(ancho,alto). De esta manera siempre
    //la ventana estará situada en el centro
    setBounds(anchopant / 3, altpant / 4, 406, 658);
    //Activar el cierre interactivo del contenedor JFrame ventana para finalizar
    //ejecución
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    //Asignación al objeto Container panelContenidos DEL PANEL JContentPane
    //del JFrame
    panelContenidos = getContentPane();
    //asignación administrador de Layout BordeLayout al panel de contenidos
    //del JFrame
    panelContenidos.setLayout(new BorderLayout());
}
```

El método private void metodoPrincipal(), es el encargado de la gestión de acciones y configuración de los botones del menuBar.

```
private void metodoPrincipal() {
    //CONFIGURACIÓN CONTENEDOR JPanel contenedor del panelMenu
    JPanel panelMenu = new JPanel();
    panelMenu.setBackground(Color.LIGHT_GRAY);
    panelMenu.setLayout(new BorderLayout());

    //DECLARACIÓN COMPONENTE JMenuBar (barra de menu)
    JMenuBar barraMenu = new JMenuBar();

    //DECLARACIÓN Y CONFIGURACIÓN COMPONENTES JMENUS DE LA BARRA DE MENU
    JMenu opciones = new JMenu("Opciones");

    //DECLARACIÓN OPCIONES JMenuItem
    seleccionLab = new JMenuItem("Seleccionar laberinto");
    reiniciar = new JMenuItem("Reiniciar laberinto");
    salir = new JMenuItem("Salir");

    //Asociación componente JMenuItem, con gestor de evento gestorEventoMenu
    seleccionLab.addActionListener(new gestorEventosMenu());
    reiniciar.addActionListener(new gestorEventosMenu());
    salir.addActionListener(new gestorEventosMenu());

    //Asignación componentes JMenuItem con su correspondiente JMenu
    opciones.add(seleccionLab);
    opciones.add(reiniciar);
    opciones.add(salir);

    //Introducción de la componentes JMenu en el JMenuBar
    barraMenu.add(opciones);
    //Introducción de la componentes JMenuBar en el JPanel
    panelMenu.add(barraMenu, BorderLayout.SOUTH);

    //Adicion del mapa y barraMenu al JPanel panelContenidos
    panelContenidos.add(panelMenu, BorderLayout.NORTH);
}
```

El método public static String getFicheroNombre(), tiene la funcionalidad de devolver el nombre del fichero que contiene el mapa a pintar.

```
//Devuelve el nombre del fichero que contiene el mapa
public static String getFicheroNombre() {
    return fichero;
}
```

El método private void seleccionarLaberinto(), permite seleccionar el fichero que contiene el mapa del Laberinto a través de una ventana JFileChooser, la cual aplicamos un filtro para que el usuario solo pueda ver archivos “.txt” o “archivos de texto”.

```
//Este método permite seleccionar el fichero
private void seleccionarLaberinto() {
    //DECLARACIÓN Y CONFIGURACIÓN JFileChooser
    JFileChooser ventanaSeleccion = new JFileChooser();
    //HACEMOS QUE LO ÚNICO QUE SE VEAN SEA LOS .txt Y ARCHIVOS DE TEXTO
    FileNameExtensionFilter filtro
            = new FileNameExtensionFilter("archivos de texto", "txt");
    //Se añade el filtro a la JFileChooser
    ventanaSeleccion.setFileFilter(filtro);
    try {
        int x = ventanaSeleccion.showOpenDialog(ventanaSeleccion);
        fichero = null;
        if (x == JFileChooser.APPROVE_OPTION) {
            fichero = ventanaSeleccion.getSelectedFile().getName();
        }
    } catch (HeadlessException error) {
        System.out.println("Error 1: " + error.toString());
    } catch (Exception error) {
        System.out.println("Error 2: " + error.toString());
    }
}
```

La clase public class gestorDesplazamientoFicha gestiona el desplazamiento de la ficha sobre el laberinto, esta clase implementa la interface KeyListener, la cual nos permitirá a partir del teclado mover la ficha.

```
public class gestorDesplazamientoFicha implements KeyListener {

    @Override
    public void keyPressed(KeyEvent ke) {
        try {
            //Boolean que indica si se realiza algún cambio
            boolean cambio = false;
            //Variable para comparar si el dato leido del fichero equivale a '1'
            Character cero = '0';
            //Bucles para recorrer las filas y columnas de la matriz
            for (int i = 0; i < filas; i++) {
                for (int j = 0; j < columnas; j++) {
                    //Se comprueba que la casilla leida no este ocupada
                    if (mapa.getMatriz(i, j).estado()) {
                        //Mover si se pulsa las teclas correspodientes y la
                        //columna no es el límite izquierdo del mapa
                        if (((ke.getKeyCode() == KeyEvent.VK_LEFT)
                                || (ke.getKeyCode() == KeyEvent.VK_A))
                                && j != 0) {
                            System.out.println("IZQUIERDA");
                            //En el caso que la posicion actual de la ficha no
                            //coincida con un muro se permite el desplazamineto
                            //y se actualiza el estado de la casilla
                            if (mapa.getMatriz(i, j).getParedes(3) == cero) {
                                mapa.getMatriz(i, j - 1).setCasillaOcupada();
                                mapa.getMatriz(i, j).setCasillaLibre();
                            }
                            //Se cambia el booleano a true para indicar
                            //que hay un cambio
                            cambio = true;
                        }
                        //Mover si se pulsa las teclas correspodientes y la
                        //columna no es el límite derecho del mapa
                        if (((ke.getKeyCode() == KeyEvent.VK_RIGHT)
                                || (ke.getKeyCode() == KeyEvent.VK_D))
                                && j != columnas) {
                            System.out.println("DERECHA");
                            //En el caso que la posicion actual de la ficha no
                            //coincida con un muro se permite el desplazamineto
                            //y se actualiza el estado de la casilla
                            if (mapa.getMatriz(i, j).getParedes(1) == cero) {
                                mapa.getMatriz(i, j + 1).setCasillaOcupada();
                                mapa.getMatriz(i, j).setCasillaLibre();
                            }
```

```
                    }
                    //Se cambia el booleano a true para indicar
                    //que hay un cambio
                    cambio = true;
                }
                //Mover si se pulsa las teclas correspodientes y la
                //fila no es el limite superior del mapa
                if (((ke.getKeyCode() == KeyEvent.VK_UP)
                        || (ke.getKeyCode() == KeyEvent.VK_W))
                        && i != 0) {
                    System.out.println("ARRIBA");
                    //En el caso que la posicion actual de la ficha no
                    //coincida con un muro se permite el desplazamineto
                    //y se actualiza el estado de la casilla
                    if (mapa.getMatriz(i, j).getParedes(0) == cero) {
                        mapa.getMatriz(i - 1, j).setCasillaOcupada();
                        mapa.getMatriz(i, j).setCasillaLibre();
                    }
                    //Se cambia el booleano a true para indicar
                    //que hay un cambio
                    cambio = true;
                }
                //Mover si se pulsa las teclas correspodientes y la
                //columna no es el limite inferior del mapa
                if (((ke.getKeyCode() == KeyEvent.VK_DOWN)
                        || (ke.getKeyCode() == KeyEvent.VK_S))
                        && i != filas) {
                    System.out.println("ABAJO");
                    //En el caso que la posicion actual de la ficha no
                    //coincida con un muro se permite el desplazamineto
                    //y se actualiza el estado de la casilla
                    if (mapa.getMatriz(i, j).getParedes(2) == cero) {
                        mapa.getMatriz(i + 1, j).setCasillaOcupada();
                        mapa.getMatriz(i, j).setCasillaLibre();
                    }
                    //Se cambia el booleano a true para indicar
                    //que hay un cambio
                    cambio = true;
                }
                //Añadimos un break para que no se ejecute varias veces
                //una misma instrucción
                break;
            }
        }

            //si ha habido ya un cambio se finaliza el tratamiento
            if (cambio) {
                break;
            }
        }
        //Se actualiza el mapa con cada movimineto
        repaint();
    } catch (Exception error) {
        System.out.println("Error: " + error.toString());
        error.printStackTrace();
    }
}

@Override
public void keyReleased(KeyEvent ke) {
}

@Override
public void keyTyped(KeyEvent ke) {
}
}
```

La clase private class gestorEventosMenu gestiona los eventos de la barra de menu, implementando la interface ActionListener que a partir del método public void actionPerformed(ActionEvent evento), proporcionado por la interface, configuramos cada evento que queremos que se realice.

```
//MANIPULADOR EVENTOS COMPONENTES JMenu
private class gestorEventosMenu implements ActionListener {

    @Override
    public void actionPerformed(ActionEvent evento) {
        switch (evento.getActionCommand()) {
            case "Seleccionar laberinto":
                //Seleccion del mapa que sea desea jugar
                seleccionarLaberinto();
                mapa.setVisible(false);
                inicio();
                break;
            case "Reiniciar laberinto":
                mapa.setVisible(false);
                inicio();
                break;
            case "Salir":
                //Sale de la aplicación con el cierre de la ventana y la
                //finalización de la ejecución
                System.exit(0);
                break;
        }
    }
}
```

Y por último, public static void main(String[] args), inicia el programa llamando al constructor de la clase Laberinto.

```
public static void main(String[] args) {
    new Laberinto();
}
```

2.2 Clase Casilla

Esta clase es la encargada de la gestión y configuración de cada casilla del Laberinto.

2.2.1 Atributos

Dentro de la clase encontramos los siguientes atributos:

```
public class Casilla {

    //ATRIBUTO QUE PERMITE SABER SI UNA CASILLA ESTA OCUPADA O NO
    private boolean ocupada;
    //ARTIBUTO QUE PERMITE SABER SI LA FICHA ESTA SITUADA EN
    //LA CASILLA DE SALIDA
    private boolean casillaSalida;
    //ATRIBUTOS QUE DETERMINAN LA POSICIÓN DE LA CASILLA(EJES (X,Y))
    private int posCasillaX;
    private int posCasillaY;
    //ATRIBUTO PARA DETERMINAR QUE LADOS CONTIENE UNA CASILLA
    private int[] ladosCasilla;
    //ATRIBUTO QUE DETERMINA LA LONGITUD DEL LADO DE UNA CASILLA
    private final static int LONGITUD_LADO_CASILLA = 40;
    //ATRIBUTO QUE DTERMINA EL ANCHO DE CADA PARED DEL LABERINTO
    private final int ANCHO_MURO = 5;
```

Todos los atributos serán privados, ya que solo se van a utilizar dentro de esta clase. Dentro de esta encontramos: cuatro de tipo int, dos para la localización de cada casilla, uno para la longitud de lado de cada casilla, y el otro para el ancho de las paredes; una de tipo int[] para determinar los lados de una casilla; y dos de tipo boolean para saber si una casilla está ocupada y el otro para saber si la ficha está en la casilla de salida.

2.2.2 Constructor

Esta clase contiene un único constructor con tres parámetros, dos de tipo int, los cuales darán el valor a los atributos posCasillaX y posCasillaY; y uno de tipo int[], el cual dará valor al atributo ladosCasilla.

```
public Casilla(int posCasillaX, int posCasillaY, int[] ladosCasilla) {
    this.posCasillaX = posCasillaX;
    this.posCasillaY = posCasillaY;
    this.ladosCasilla = ladosCasilla;
    casillaSalida = false;
    ocupada = false;
}
```

2.2.3 Métodos

Esta clase contiene trece métodos: public boolean estado(), public void setCasillaOcupada(), public void setCasillaLibre(), public void setX(int x), public void setY(int y), public int getX(), public int getY(), public int getParedes(int i), public int[] getParedes(), public static int getLongitudLado(), public void setcasillaSalida(), public void gameOver() y public void paintComponent(Graphics g).

El método public boolean estado() devuelve el estado en el que se encuentra una casilla.

```
//MÉTODO QUE DEVUELVE EL ESTADO DE UNA CASILLA
public boolean estado() {
    return ocupada;
}
```

El método public void setCasillaOcupada() cambia el estado de una casilla a ocupada.

```
//MÉTODO QUE CAMBIA EL ESTADO A OCUPADA DE UNA CASILLA
public void setCasillaOcupada() {
    ocupada = true;
}
```

El método public void setCasillaLibre() cambia el estado de una casilla a no ocupada.

```
//MÉTODO QUE LIBERA UNA CASILLA
public void setCasillaLibre() {
    ocupada = false;
}
```

Los métodos public void setX(int x) y public void setY(int y), permiten seleccionar la posición de una casilla a través del parámetro de los métodos.

```
//MÉTODO QUE MODIFICA LA COORDENADA X DE UNA CASILLA
public void setX(int x) {
    posCasillaX = x;
}

//MÉTODO QUE MODIFICA LA COORDENADA Y DE UNA CASILLA
public void setY(int y) {
    posCasillaY = y;
}
```

Los métodos public int getX() y public int getY() devuelven el valor del atributo posCasillaX y posCasillaY.

```
//MÉTODO QUE DA ACCESO A LA COORDENADA X DE UNA CASILLA
public int getX() {
    return posCasillaX;
}

//MÉTODO QUE DA ACCESO A LA COORDENADA Y DE UNA CASILLA
public int getY() {
    return posCasillaY;
}
```

El método public int getParedes(int i) devuelve el lado especificado por parámetro de una casilla.

```
//Método que devuelve el lado especificado por parametro de una casilla
public int getParedes(int i) {
    return ladosCasilla[i];
}
```

El método public int[] getParedes() devuelve los lados que contiene una casilla.

```
//Método que devuelve los lados que contiene una casilla
public int[] getParedes() {
    return ladosCasilla;
}
```

El método public static int getLongitudLado() devuelve la longitud que tiene una casilla.

```
//Método que devuelve la longitud que tiene una casilla
public static int getLongitudLado() {
    return LONGITUD_LADO_CASILLA;
}
```

El método public void setcasillaSalida() asigna la casilla de salida.

```
//Método que asigna la casilla de salida
public void setcasillaSalida() {
    casillaSalida = true;
}
```

El método public void gameOver() avisa al usuario que esté ejecutando el programa que se ha llegado a la salida de mapa.

```
//Mensaje al llegar a la salida del mapa
public void gameOver() {
    JOptionPane.showMessageDialog(null, "        HAS GANADO!!!");
    //Cerrar el juego al ganar
    System.exit(0);
}
```

El método public void paintComponent(Graphics g) se encarga de dibujar correctamente la casilla según los datos leídos del fichero, además de dibujar la ficha en la casilla correspondiente. Para ello empleamos la clase Graphics2D donde con ella pintamos las casillas del laberinto, donde dependiendo del array “ladosCasilla” si contiene un 1 se pintará un muro, en caso de contener un 0 no se dibujara la pared. Este array “ladosCasilla” tendrá cuatro posibles componentes en el que cada posición indica la pared a dibujar, es decir, Norte será el valor de la posición en el array número 0, este 1, sur 2 y oeste 3.

```
public void paintComponent(Graphics g) {
    //Variable para comparar si el dato leido del fichero equivale a '1'
    Character uno = '1';
    //Usamos la clase Graphics2D para dibujar la casilla
    Graphics2D g2D = (Graphics2D) g;
    //Instancia una ficha
    Ficha ficha = new Ficha();
    //Si la casilla esta ocupada significa que la ficha se
    //encuentra en esa posición
    if (ocupada) {
        //Se le asigna la posición a la ficha
        ficha.setCoordX(posCasillaX + 7);
        ficha.setCoordY(posCasillaY + 7);
        //Se pinta la ficha
        ficha.paintComponent(g2D);
        //Si la ficha se encuentra en la casilla de salida se acaba la partida
        if (casillaSalida) {
            //Se llama al método de fin de partida
            gameOver();
        }
    }
    //Rectangulo2D para formar las paredes de caada casilla
    Rectangle2D.Float paredCasilla;
    //Cada casilla se compone de 4 lados Norte[0], este[1], sur[2] y oeste[3],
    //donde si en el fichero leido hay un 0 no hay lado y 1 viceversa
    if (ladosCasilla[0] == uno) { //Norte = [0]
        //Norte = (x,y,ladoCasilla,5)
        paredCasilla = new Rectangle2D.Float(posCasillaX,
                posCasillaY, LONGITUD_LADO_CASILLA, ANCHO_MURO);
        g2D.setColor(Color.BLACK);
        g2D.fill(paredCasilla);
    }
    if (ladosCasilla[1] == uno) { //Este = [1]
        //Este = (x+ladoCasilla,y,5,ladoCasilla)
        paredCasilla = new Rectangle2D.Float(posCasillaX
                + LONGITUD_LADO_CASILLA - ANCHO_MURO, posCasillaY,
                ANCHO_MURO, LONGITUD_LADO_CASILLA);
        g2D.setColor(Color.BLACK);
        g2D.fill(paredCasilla);
    }
    if (ladosCasilla[2] == uno) { //Sur = [2]
        //Sur = (x,y+ladoCasilla,ladoCasilla,5)
        paredCasilla = new Rectangle2D.Float(posCasillaX,
                posCasillaY + LONGITUD_LADO_CASILLA - ANCHO_MURO,
                LONGITUD_LADO_CASILLA, ANCHO_MURO);
        g2D.setColor(Color.BLACK);
        g2D.fill(paredCasilla);
    }
    if (ladosCasilla[3] == uno) { //Oeste = [3]
        //Oeste = (x,y,5,ladoCasilla)
        paredCasilla = new Rectangle2D.Float(posCasillaX,
                posCasillaY, ANCHO_MURO, LONGITUD_LADO_CASILLA);
        g2D.setColor(Color.BLACK);
        g2D.fill(paredCasilla);
    }
}
```

2.3 Clase Mapa

Esta clase es la encargada de generar y pintar el mapa del laberinto que se esté jugando, a partir de un documento de texto seleccionado por el usuario.

2.3.1 Atributos

Dentro de la clase encontramos los siguientes atributos:

```
public class Mapa extends JPanel {

    //ATRIBUTO QUE CONTIENE EL NÚMERO DE FILAS DEL MAPA
    private static int filas;
    //ATRIBUTO QUE CONTIENE EL NÚMERO DE COLUMNAS DEL MAPA
    private static int columnas;
    //ATRIBUTO QUE CONTIENE LA FILA DE SALIDA DEL LABERINTO
    private static int filaSalida;
    //ATRIBUTO QUE CONTIENE LA COLUMNA DE SALIDA DEL LABERINTO
    private static int columnaSalida;
    //ATRIBUTO QUE ALMACENA LOS 1's y 0's PARA GENERAR UNA CASILLA
    private int[] ladosCasilla;
    //ATRIBUTO QUE CONTIENE EL NUMERO DE FILAS Y COLUMNAS DEL MAPA
    private Casilla matriz[][];
```

Todos los atributos serán privados, ya que solo se van a utilizar dentro de esta clase. Dentro de esta, encontramos cuatro atributos de tipo int para determinar el número de filas y columnas que contiene el laberinto, y para determinar la posición de fila y columna de salida; uno de tipo int[] para saber que paredes de la casilla hay que dibujar; y uno de tipo Casilla que contiene la matriz del mapa.

2.3.2 Constructor

Esta clase contiene un único constructor, en el cual se llama al método crearMapa para que cada vez que se instancie un objeto de tipo Mapa se genere un nuevo Laberinto.

```
//CONSTRUCTOR
public Mapa() {
    crearMapa(Laberinto.getFicheroNombre());
}
```

2.3.3 Métodos

Esta clase contiene diez métodos: private void crearMapa(String nombreMapa), public static int getFilaSalida(), public static int getColumnaSalida(), public static int getFilas(), public static int getColumnas(), public Casilla getMatriz(int i, int j), private void setPosAleatoriaFicha(), private void setCasillaSalida(), public static Rectangle2D getCasillaSalida() y public void paintComponent(Graphics g).

El método private void crearMapa(String nombreMapa) permite crear el mapa del Laberinto que se ha seleccionado a partir de un fichero de texto. Para crear el laberinto lo que se hace es leer las dos primeras líneas que contienen el número de filas y columnas, seguidamente se lee el contenido de cada casilla y finalmente se leen las últimas dos líneas del archivo las cuales contienen la posición de la casilla de salida. Al final de la lectura del mapa se sitúa la ficha en una posición aleatoria con el método setPosAleatoriaFicha() y se asigna a una casilla la casilla de salida con el método setCasillaSalida().

```
private void crearMapa(String nombreMapa) {
    try {
        //Cuando se ejecuta por primera vez el programa  el mapa por defecto
        //será "maze1.txt"
        if (nombreMapa == null) {
            nombreMapa = "maze1.txt";
        }
        //DECLARACIÓN E INTANCIACIÓN de la clase FicheroLecturaMapas, para leer del
        //fichero los datos, para generar el mapa
        FicheroLecturaMapas lecturaDatos = new FicheroLecturaMapas(nombreMapa);
        //Leemos la primera linea del fichero de la cual obtenemos el número
        //de filas que contiene el mapa y pasamos su valor a numérico
        filas = Integer.parseInt(lecturaDatos.lectura());
        //Leemos la siguiente linea del fichero la cual contiene el nu´número
        //de columnas que contiene el mapa y pasamos su valor a numérico
        columnas = Integer.parseInt(lecturaDatos.lectura());
        //A partor de los datos obtenidos anteriormenete ya sabemos las
        //dimensiones del mapa a generar
        matriz = new Casilla[filas][columnas];
        //Variables para determinar la posición de cada casilla a generar
        int x, y = 0;
        //Bucle para generar el mapa
        for (int i = 0; i < filas; i++) {
            //Cada fila se reinicia la variable x;
            x = 0;
            for (int j = 0; j < columnas; j++) {
                ladosCasilla = new int[4];
                //Bucle para generar los lados de cada casilla
                for (int k = 0; k < 4; k++) {
                    ladosCasilla[k] = lecturaDatos.leer();
                }
                matriz[i][j] = new Casilla(x, y,ladosCasilla);
                //Aumentamos las coordenadas de x, para que la siguiente casilla
                //este posicionada correctamente
                x = x + Casilla.getLongitudLado();
            }
            //Avanzar linea de lectura
            lecturaDatos.lectura();
            //Aumentamos las coordenadas de y, para que la siguiente casilla
            //este posicionada correctamente
            y = y + Casilla.getLongitudLado();
        }
        //Leemos las dos últimas lineas que contienen la salida del laberinto
        filaSalida = Integer.parseInt(lecturaDatos.lectura());
        columnaSalida = Integer.parseInt(lecturaDatos.lectura());
        //Cierre enlace con el fichero
        lecturaDatos.close();
        //Aginación de la casilla de salida
        setPosAleatoriaFicha();
        setCasillaSalida();
    } catch (NumberFormatException error) {
        System.out.println("Error: " + error.toString());
        error.printStackTrace();
    } catch (Exception error) {
        System.out.println("Error: " + error.toString());
        error.printStackTrace();
    }
}
```

Los métodos public static int getFilaSalida(), public static int getColumnaSalida(), public static int getFilas() y public static int getColumnas(), devuelven la fila, la columna de salida, el número de filas y el número de columnas del mapa correspondientemente.

```
//Método que devuelve la fila de salida
public static int getFilaSalida() {
    return filaSalida;
}

//Método que devuelve la columna de salida
public static int getColumnaSalida() {
    return columnaSalida;
}

//Método que devuelve el número de filas que contiene el laberinto
public static int getFilas() {
    return filas;
}

//Método que devuelve el número de columnas que contiene el laberinto
public static int getColumnas() {
    return columnas;
}
```

El método public Casilla getMatriz(int i, int j) devuelve la casilla indicada por los parámetros del método.

```
//Método que devuelve la matriz que contiene el laberinto
public Casilla getMatriz(int i, int j) {
    return matriz[i][j];
}
```

El método private void setPosAleatoriaFicha() asigna una posición aleatoria inicial a la ficha, mediante la clase Random.

```
private void setPosAleatoriaFicha() {
    //Variable tipo random para determinar una posición aleatoria de la ficha
    //al iniciar un mapa
    Random posicion = new Random();
    //Variable que contendrá una fila aleatoria entre 0 y el número de filas
    //que contenga el mapa
    int p = posicion.nextInt(filas);
    //Se crea un nuevo random para poder determinar de forma aleatoria la
    //columna
    posicion = new Random();
    //Variable que contendrá una columna aleatoria entre 0 y el número de
    //columnas que contenga el mapa
    int q = posicion.nextInt(columnas);
    //Se asigna a la casilla de salida que está ocupada
    matriz[p][q].setCasillaOcupada();
}
```

El método private void setCasillaSalida() asigna la casilla de salida.

```
//Método que asigna la casilla de salida
private void setCasillaSalida() {
    matriz[filaSalida][columnaSalida - 1].setcasillaSalida();
}
```

El método public static Rectangle2D getCasillaSalida() devuelve la casilla de salida del Laberinto.

```
//Método que devuelve la casilla de salida del Laberinto
private static Rectangle2D getCasillaSalida() {
    return new Rectangle2D.Float((Casilla.getLongitudLado() * columnaSalida) - Casilla.getLongitudLado(),
            Casilla.getLongitudLado() * filaSalida,
            Casilla.getLongitudLado(), Casilla.getLongitudLado());
}
```

El método public void paintComponent(Graphics g) permite dibujar el mapa del Laberinto a partir de la clase Graphics2D. Para dibujar el laberinto se recorre con dos bucles for la matriz que contiene el laberinto y se va pintando.

```
//Método que permite dibujar el mapa del Laberinto
@Override
public void paintComponent(Graphics g) {
    try {
        Graphics2D g2D = (Graphics2D) g;
        //Pintar el un rectangulo que será el fondo del mapa, desde la posición
        //0,0 y con dimensión ancho = num de columnas * la longuitud de una casilla
        // y el alto = num de filas * la longuitud de una casilla
        g2D.setColor(Color.PINK);
        g2D.fillRect(0, 0, columnas * Casilla.getLongitudLado(),
                filas * Casilla.getLongitudLado());
        //Dibujar la casilla de salida del laberinto
        g2D.setColor(Color.LIGHT_GRAY);
        g2D.fill(getCasillaSalida());
        //Bucle para pintar el laberinto
        for (int i = 0; i < filas; i++) {
            for (int j = 0; j < columnas; j++) {
                matriz[i][j].paintComponent(g2D);
            }
        }
    } catch (Exception error) {
        System.out.println("Error dibujando mapa: " + error.toString());
        error.printStackTrace();
    }
}
```

2.4 Clase FicheroLecturaMapa

Esta clase es la encargada de la lectura del fichero de texto que contiene los datos sobre la estructura del mapa del Laberinto.

2.4.1 Atributos

Dentro de la clase encontramos los siguientes atributos:

```
public class FicheroLecturaMapas {

    private BufferedReader br;
    private FileReader fr;
```

Todos los atributos serán privados, ya que solo se van a utilizar dentro de esta clase. Dentro de esta, encontramos un atributo de tipo bufferedreader y otra de filereader, utilizadas para la lectura del fichero de texto.

2.4.2 Constructor

Esta clase contiene un único constructor, con un parámetro de tipo String que será el que contenga el nombre del fichero de texto a leer. Dentro del constructor se inicializan los atributos bufferedreader y filereader para el enlace con el fichero.

```
    public FicheroLecturaMapas(String nombreFichero) {
        try {
            fr = new FileReader(nombreFichero);
            br = new BufferedReader(fr);
        } catch (FileNotFoundException ex) {
            System.out.println("Exception abriendo fichero:"
                    + " " + nombreFichero + " error: " + ex);
        }
    }
```

2.4.3 Métodos

Esta clase contiene tres métodos: public String lectura(), public int leer() y public void close().

El método public String lectura(), es el encargado de leer una línea del fichero cada vez que es llamado.

```java
// Método de lectura de una linea del fichero.
    public String lectura() {
        String linea = null;
        try {
            linea = br.readLine();
        } catch (Exception error) {
            System.out.println("Error leyendo: " + error.toString());
        }
        return linea;
    }
```

El método public int leer(), es el encargado de leer un int del fichero cada vez que es llamado.

```java
// Método de lectura un int del fichero.
public int leer() {
    int x = 0;
    try {
        x = br.read();
    } catch (IOException e) {
        System.out.println("Exception leyendo fichero error: " + e);
    }
    return x;
}
```

El último método que encontramos es public void close(), encargado de cerrar el enlace con el fichero.

```java
//Método de cierre de enlace con fichero.
public void close() {
    try {
        fr.close(); //Cerrar FileReader
        br.close(); // Cerrar BufferedReader
    } catch (IOException ex) {
        System.out.println("Exception cerrando fichero: " + ex);
    } finally {
        try {
            fr.close();
        } catch (IOException e) {
            System.out.println("Exception cerrando fichero: " + e);
        }
    }
}
```

2.5 Clase Ficha

Esta clase es la encargada de la creación de un objeto ficha, la cual será la visualizada por el usuario al ejecutar el programa.

2.5.1 Atributos

Dentro de la clase encontramos los siguientes atributos:

```
public class Ficha {

    //ATRIBUTO QUE DEFINE LA POSICIÓN X DE LA FICHA
    private int x;
    //ATRIBUTO QUE DEFINE LA POSICIÓN Y DE LA FICHA
    private int y;
    //ATRIBUTO QUE DEFINE EL ANCHO Y ALTO DE LA FICHA
    private final int DIAMETRO_FICHA = 25;
```

Todos los atributos serán privados, ya que solo se van a utilizar dentro de esta clase. Dentro de esta, encontramos un dos de tipo int que definen la posición x,y de la ficha, las cuales irán variando dependiendo de donde se encuentre la ficha en el laberinto. También encontramos un int pero final, donde determina el tamaño de la ficha.

2.5.2 Constructor

En cuanto al constructor de la clase tenemos un constructor vacío.

```
    //CONSTURTOR
    public Ficha() {

    }
```

2.5.3 Métodos

Esta clase contiene tres métodos: public void setCoordX(int x), public void setCoordY(int y), y public void paintComponent(Graphics g).

El método public void setCoordX(int x), es el encargado de indicar las coordenadas de X de la ficha, a partir del parámetro del método, para así asignarle al atributo x un valor.

```
//Método que indica las coordenadas de X de la ficha, a partir del parámetro
//del método
public void setCoordX(int x) {
    this.x = x;
}
```

El método public void setCoordY(int y), es el encargado de indicar las coordenadas de Y de la ficha, a partir del parámetro del método, para así asignarle al atributo y un valor.

```
//Método que indica las coordenadas de Y de la ficha, a partir del parámetro
//del método
public void setCoordY(int y) {
    this.y = y;
}
```

El método public void paintComponent(Graphics g), permite pintar la ficha en el mapa, a partir de la clase Graphics2D.

```
//Método que permite dibujar la ficha sobre el mapa
public void paintComponent(Graphics g) {
    //Utilizamos Graphics2D para la visualización
    Graphics2D g2d = (Graphics2D) g;
    g2d.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
            RenderingHints.VALUE_ANTIALIAS_ON);
    //FICHA
    g2d.setColor(Color.RED);
    g2d.fillOval(x, y, DIAMETRO_FICHA, DIAMETRO_FICHA);
    //CONTORNO FICHA
    g2d.setColor(Color.BLACK);
    g2d.drawOval(x, y, DIAMETRO_FICHA, DIAMETRO_FICHA);
}
```

3. Conclusiones

Para concluir, queremos hacer énfasis en lo que para nosotros ha sido lo más importante, y es el trabajo y el esfuerzo. Durante el transcurso de este proyecto, hemos aplicado la programación orientada a objetos para desarrollar el programa, con un diseño descendiente, de donde partíamos de un problema grande, el cual, dividimos en diferentes problemas menores a resolver, ya que así nos sería más fácil resolver el enunciado planteado, donde cada problema menor sería una clase diferente. Esta manera de programar es muy conveniente ya que, podemos localizar fácilmente nuestro error, si hay que cambiar alguna parte del código no tenemos que cambiar el programa entero, y desarrollar el algoritmo del programa es más cómodo y simple de entender. Gracias a este proyecto, hemos podido consolidar los conocimientos adquiridos durante el curso sobre Java y la Programación Orientada a Objetos.