DOC: Proyecto - Juego Sudoku (Java)

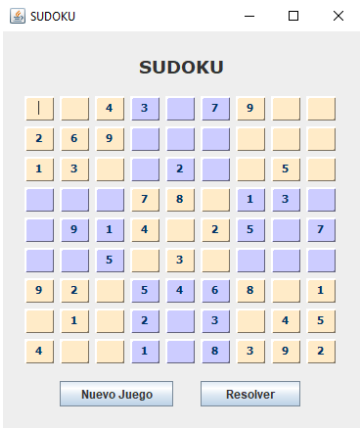
Proyecto desarrollado en NetBeans 8.2

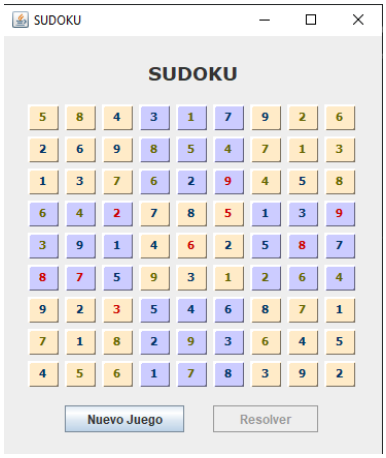
Diseño del juego

En primera instancia, se debe seleccionar el nivel que determinará la dificultad de la grilla. Se abre una ventana para que el jugador elija.

Una vez seleccionado, la ventana se cierra para aparecer la correspondiente a la ventana donde se desarrolla el juego. En esta se encuentra la grilla con los números fijos y el resto de las celdas vacías para ser completadas a través del teclado por el jugador.

Debajo se encuentran dos botones. El botón "Nuevo Juego" regresa a la ventana de selección de nivel para que el jugador pueda comenzar el juego de nuevo. El botón "Resolver" compara lo ingresado por el usuario con los números de la grilla resuelta. De esta manera cambia el color del número a verde en caso de ser el correcto y a rojo cuando no es coincidente o la celda se encuentra vacía (cargando el número correcto). Deshabilita el botón "Resolver" al igual que las celdas de la grillla.





Algoritmo para generar la grilla del juego

El proyecto está compuesto por cuatro archivos .java los cuales solo tres son los utilizados para su correcto funcionamiento. Estos son GameInterface.java, LevelInterface.java y Sudoku.java.

Las funciones para la construcción de la grilla, carga de números en la ventana, carga de solución del juego y nuevo juego se desarrollan en la clase GameInterface.java (extendida de JFrame)

Para llenar la grilla primero se inicializa un arreglo cargado con números del 1 al 9 sin repetir ordenados de forma aleatoria. Esto se logra con el método *llenarArregloNumerosRandom()*, el cual devuelve el arreglo para ser utilizado en el siguiente método.

```
public int[] llenarArregloNumerosRandom(int[] array) {
        Random random = new Random();
        int posicion = 0;
        while (posicion < array.length) {
            int numeroRandom = random.nextInt(9);
            if (numeroRandom == 0) {
                numeroRandom = 9;
            }
```

Con un bucle *while* se recorre la longitud del arreglo para primero definir una variable int con un número aleatorio del 0 al 8, razón por la cual se entra luego a un *if* para remplazar el 0 por un 9.

Se determina que al no haber sido comprobado si el número ya está repetido en el arreglo, se lo define como falso. Entrando al segundo *while* se recorre el arreglo recibido para comparar y determinar si el número que se quiere ingresar en esa posición ya existe. Si está repetido, vuelve a cargar otro número aleatorio, si no lo guarda en la posición y pasa a la siguiente.

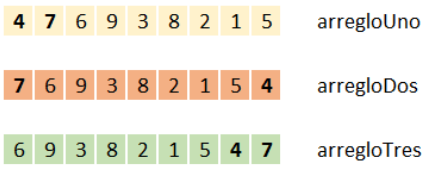
```
            boolean repetido = false;
            while (!repetido) {
                for (int i = 0; i < posicion; i++) {
                    if (numeroRandom == array[i]) {
                        repetido = true;
                        break;
                    }
                }
                if (!repetido) {
                    array[posicion] = numeroRandom;
                    posicion++;
                }
            }
        }
        return array;
    }
```

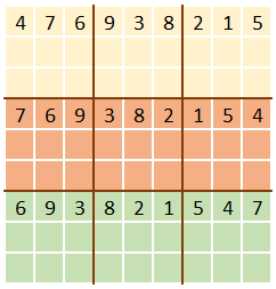
Teniendo el arreglo, el siguiente método *llenarGrillaConArregloRandom()* lo utiliza para cargar otros dos arreglos: *arregloDos* y *arregloTres*, mandarlos a otro arreglo para reorganizarlos y devolver la grilla completa.

```
public int[][] llenarGrillaConArregloRandom(int[][] grid, int[] arregloUno) {
        int[] arregloDos = new int[9];
        int[] arregloTres = new int[9];

        for (int i = 0; i < 8; i++) {
            arregloDos[i] = arregloUno[i + 1];
        }
        arregloDos[8] = arregloUno[0];

        for (int i = 0; i < 7; i++) {
            arregloTres[i] = arregloUno[i + 2];
        }
        arregloTres[7] = arregloUno[0];
        arregloTres[8] = arregloUno[1];

        cambiarColumnasCadaTres(grid, arregloUno, 0);
        cambiarColumnasCadaTres(grid, arregloDos, 3);
        cambiarColumnasCadaTres(grid, arregloTres, 6);

        return grid;
    }
```

El *arregloUno* se almacenará en el *arregloDos* a partir de su segundo número, colocando el primero al final del *arregloDos*.

Para el *arregloTres* ocurriría lo mismo, exceptuando que se llena a partir del tercer número del *arregloUno* y sus primeros dos números será almacenados al final del *arregloTres*.

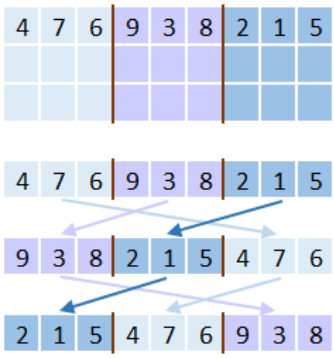




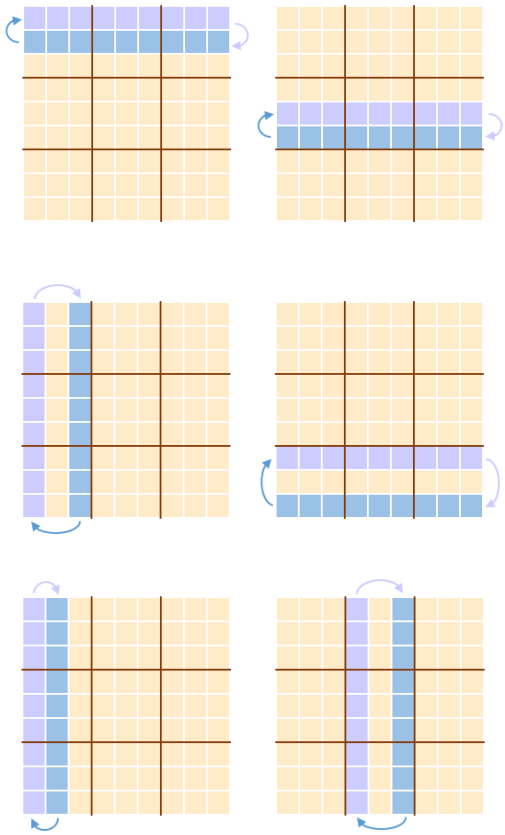
Estos arreglos serán usados en el método *cambiarColumnasCadaTres()* según su posición en la grilla. El método se encarga de ir cargando la grilla dividiéndola en tres para cada arreglo → primera sección (amarilla) para *arregloUno*, segunda sección (naranja) para *arregloDos*, tercera sección (verde) para *arregloTres*.

```
public int[][] cambiarColumnasCadaTres(int[][] grid, int[]
array, int fila) {
        //agrega arregloUno a la primera fila de la grilla
        System.arraycopy(array, 0, grid[fila], 0, 9);

        for (int i = 0; i < 3; i++) {
            //agrega intercalado arregloUno a 2da fila
            grid[fila + 1][i] = array[i + 3];
            grid[fila + 1][i + 3] = array[i + 6];
            grid[fila + 1][i + 6] = array[i];

            //agrega intercalado arregloUno a 3ra fila
            grid[fila + 2][i] = array[i + 6];
            grid[fila + 2][i + 3] = array[i];
            grid[fila + 2][i + 6] = array[i + 3];
        }
        return grid;
    }
```

Se almacena el arreglo en la primera fila del bloque correspondiente, subdividida en tres partes, para intercalar el almacenamiento de los números



Lo siguiente es cambiar de lugar tanto filas como columnas de la grilla. Esto se realiza para que los números se dispongan de manera desordenada a ojos del jugador. La metodología es simple: utilizar un arreglo donde almacenar la fila o columna que se vaya a remplazar por otra que vaya a ocupar su lugar.



```
public int[][] cambiarOrdenFilasYColumnas(int[][] grid, int[] arra
y) {
        // cambio de posicion de filas
        System.arraycopy(grid[0], 0, array, 0, 9);
        System.arraycopy(grid[1], 0, grid[0], 0, 9);
        System.arraycopy(array, 0, grid[1], 0, 9);

        System.arraycopy(grid[5], 0, array, 0, 9);
        System.arraycopy(grid[4], 0, grid[5], 0, 9);
        System.arraycopy(array, 0, grid[4], 0, 9);

        System.arraycopy(grid[6], 0, array, 0, 9);
        System.arraycopy(grid[8], 0, grid[6], 0, 9);
        System.arraycopy(array, 0, grid[8], 0, 9);

        // cambio de posicion de columnas
        for (int i = 0; i < 9; i++) {
              array[i] = grid[i][0];
        }
        for (int i = 0; i < 9; i++) {
              grid[i][0] = grid[i][2];
        }
        for (int i = 0; i < 9; i++) {
              grid[i][2] = array[i];
        }

        for (int i = 0; i < 9; i++) {
              array[i] = grid[i][0];
        }
        for (int i = 0; i < 9; i++) {
              grid[i][0] = grid[i][1];
        }
        for (int i = 0; i < 9; i++) {
              grid[i][1] = array[i];
        }

        for (int i = 0; i < 9; i++) {
              array[i] = grid[i][3];
        }
        for (int i = 0; i < 9; i++) {
              grid[i][3] = grid[i][5];
        }
        for (int i = 0; i < 9; i++) {
              grid[i][5] = array[i];
        }

        return grid;
    }
```

```
public int[][] borrarNumerosAleatorios(int level) {
        switch (level) {
            case 1: level = 40;
                break;
            case 2: level = 50;
                break;
            case 3: level = 60;
                break;
            default: level = 40;
        }

        Random random = new Random();
        for (int i = 0; i < level; i++) {
            int n = random.nextInt(9);
            int m = random.nextInt(9);
            if (grilla[n][m] != 0) {
                grilla[n][m] = 0;
            } else {
                i--;
            }
        }
        return grilla;
    }
```

Por último, el borrado de números al azar se realiza en el método *borrarNumerosAleatorios()* el cual recibe un entero que corresponde al nivel escogido por el jugador. Para la dificultad mínima (*Fácil*), se establece el borrado de 40 números de la grilla, para dificultad intermedia se borran 50 números y finalmente para el último nivel se borran 60 números.

Se salta de celda en celda determinado por posiciones aleatorias para guardar un cero.

Cargar grilla

11	12	13	14	15	16	17	18	19
12	22	23	24	25	26	27	28	29
31	32	33	34	35	36	37	38	39
41	42	43	44	45	46	47	48	49
51	52	53	54	55	56	57	58	59
61	62	63	64	65	66	67	68	69
71	72	73	74	75	76	77	78	79
81	82	83	84	85	86	87	88	89
91	92	93	94	95	96	97	98	99

La interfaz de la grilla está compuesta por 81 Text Files cuyo nombre es asignado según la posición que ocupan en la matriz. Cada una de estas celdas se carga de manera individual en el método *cargarGrilla()*, setteándole la devolución del método *cargarCelda()*, el cual recibe la posición de la celda para coincidir con los números de la grilla. Por ejemplo, para la celda 11:

```
txtCelda11.setText(cargarCelda(this.grilla, txtCelda11, 0,0));
```

A través de un *if* se verifica que el número de la celda en la posición ingresada al método no sea igual a cero y configurar la celda. Se la deshabilita para que el jugador no pueda modificarla, se le cambia la fuente (*bold*) y el color (azul) y finalmente se guarda el valor en la celda.

En caso contrario, la celda queda habilitada y se le cambia fuente (default) y color (negro)

```
public String cargarCelda(int[][]grid, javax.swing.JTextField celda, int i, int j){
        String variableCelda="";
        if (grilla[i][j] != 0) {
            celda.setEditable(false); //deshabilita la celda
            celda.setFont(new java.awt.Font("Tahoma", 1, 11)); //fuente en negrita
            celda.setForeground(new java.awt.Color(0, 51, 102)); //color azul
            variableCelda=Integer.toString(grilla[i][j]);
        } else {
            celda.setEditable(true);
            celda.setFont(new java.awt.Font("Tahoma", 0, 11));
            celda.setForeground(new java.awt.Color(0, 0, 0));
        }
        return variableCelda;
    }
```

Para evitar que el usuario ingrese valores que no sean números, a cada una de los txt files se le incorpora el método *soloNumeros()* creándole el evento **KeyTyped**. El método sólo permite el ingreso de números desde el 0 al 9.

```
public void soloNumeros(KeyEvent evt){
        char car = evt.getKeyChar();
```

```
        if ((car < '0' || car > '9') && (car != (char) KeyEvent.VK_BACK_SPACE)) {
            evt.consume();
        }
    }
```

```
private void txtCelda11KeyTyped(java.awt.event.KeyEvent evt) {
        soloNumeros(evt);
}
```

El método principal que ejecuta los demás métodos es *nuevoJuego()*, el cual recibe el nivel para determinar la dificultad del juego.

Se hace uso de dos variables de uso compartido para toda la clase: *grilla* y *grillaResuleta*. Dos matrices 9x9. En la *grillaResuelta* se carga la grilla con todos los números para luego copiarlos en la *grilla* la cual será sometida al remplazo aleatorio de números.

El botón *btnResolver* es habilitado en caso de no estarlo ya que nos encontramos en una ronda de juego nueva.

El método *mostrarGrilla()* se encarga de escribir en consola la solución del juego actual (*grillaResuelta*)

```
public void nuevoJuego(int nivel) {

        int[] arreglo = new int[9];

        llenarArregloNumerosRandom(arreglo);
        llenarGrillaConArregloRandom(grillaResuelta, arreglo);
        cambiarOrdenFilasYColumnas(grillaResuelta, arreglo);

        for (int i = 0; i < 9; i++) {
            System.arraycopy(grillaResuelta[i], 0, grilla[i], 0,
 9);
        }

        borrarNumerosAleatorios(nivel);
        cargarGrilla();
        btnResolver.setEnabled(true);

        System.out.println("--------SOLUCIÓN---------");
        mostrarGrilla(grillaResuelta);
}
```

Resolver juego

A través del evento **MouseClicked** del botón *btnResolver* se ejecuta el método *comprobarGrilla()* y se deshabilita dicho botón. El funcionamiento de éste método es igual que la carga de la grilla a la interfaz. Ejemplo:

```
txtCelda11.setText(comprobarCelda(this.grillaResuelta, txtCelda11, 0,0));
```

El método *comprobarCelda()* comprueba el valor de la celda realiza lo siguiente:

```
public String comprobarCelda(int[][]grillaResuleta, javax.swing.JTextField celda, int i, int j){
        String variableCelda="";
        // Debido a que el método recarga la grilla una vez clickeado el botón "Resolver", la validación
        // se hace sobre los números ingresados por el usuario en los text field de la interfaz gráfica
        // y no los de la grilla generada al iniciar el juego
        if(!celda.getText().equals(Integer.toString(grilla[i][j]))){
            // Entrado al if se ha validado que no se está trabajando sobre los números deshabilitados de
            // la grilla

            // Si el número ingresado en la celda es correcto
            if(celda.getText().equals(Integer.toString(grillaResuelta[i][j]))){
                celda.setForeground(new java.awt.Color(102, 102, 0)); //color verde
                celda.setFont(new java.awt.Font("Tahoma", 1, 11)); //negrita
                variableCelda = Integer.toString(grillaResuelta[i][j]);

            // Si el número ingresado en la celda es incorrecto
            } else if (!celda.getText().equals(Integer.toString(grillaResuelta[i][j]))){
                celda.setText(Integer.toString(grillaResuelta[i][j]));
                celda.setForeground(new java.awt.Color(204, 0, 0)); //color rojo
                celda.setFont(new java.awt.Font("Tahoma", 1, 11));
                variableCelda = Integer.toString(grillaResuelta[i][j]);

            // Si la celda se encuentra vacía
            } else if(celda.getText().isEmpty()) {  //celda.getText()==""
                celda.setText(Integer.toString(grillaResuelta[i][j]));
                celda.setFont(new java.awt.Font("Tahoma", 1, 11));
```

```
                variableCelda = Integer.toString(grillaResuelta[i][j]);
            }
        } else {
            variableCelda = Integer.toString(grilla[i][j]);
        }
        celda.setEditable(false); // se deshabilitan todas las celdas para que el usuario no las modifique
        return variableCelda;
    }
```

Nuevo juego

A través del evento **MouseClicked** del botón *btnNuevoJuego* se cierra la ventana de la grilla (interface del juego) y se genera una nueva instancia para la selección del nivel para luego aparecer la ventana de niveles (interfaz para seleccionar nivel del nuevo juego)

```
private void btnNuevoJuegoMouseClicked(java.awt.event.Mous
eEvent evt) {
        this.setVisible(false);
        LevelInterface elegirNivel = new LevelInterface();
        elegirNivel.setVisible(true);
}
```

Elegir nivel

Esta función se encuentra en la clase LevelInterface.java y cada botón ejecuta el método elegirNivel(), dependiendo dificultad enviará un número del 1 al 3. La ventana desaparece y crea una nueva instancia de juego (GameInterface). Se le settea el nivel escogido al método que se encarga de iniciar la grilla para el jugador. Luego vuelve visible la ventana donde se cargará la grilla.

```
public void elegirNivel(int nivel){
        this.setVisible(false);
        GameInterface juego = new GameInterface();
        juego.nuevoJuego(nivel);
        juego.setVisible(true);
    }
```

Ejecución

El juego se ejecuta desde la clase Sudoku.java se instancia la interfaz para la elección del nivel y se abre la ventana de LevelInterface

```
public class Sudoku {
    public static void main(String[] args) {
        LevelInterface elegirNivel = new LevelInterface();
        elegirNivel.setVisible(true);
    }
}
```