Elena Mena

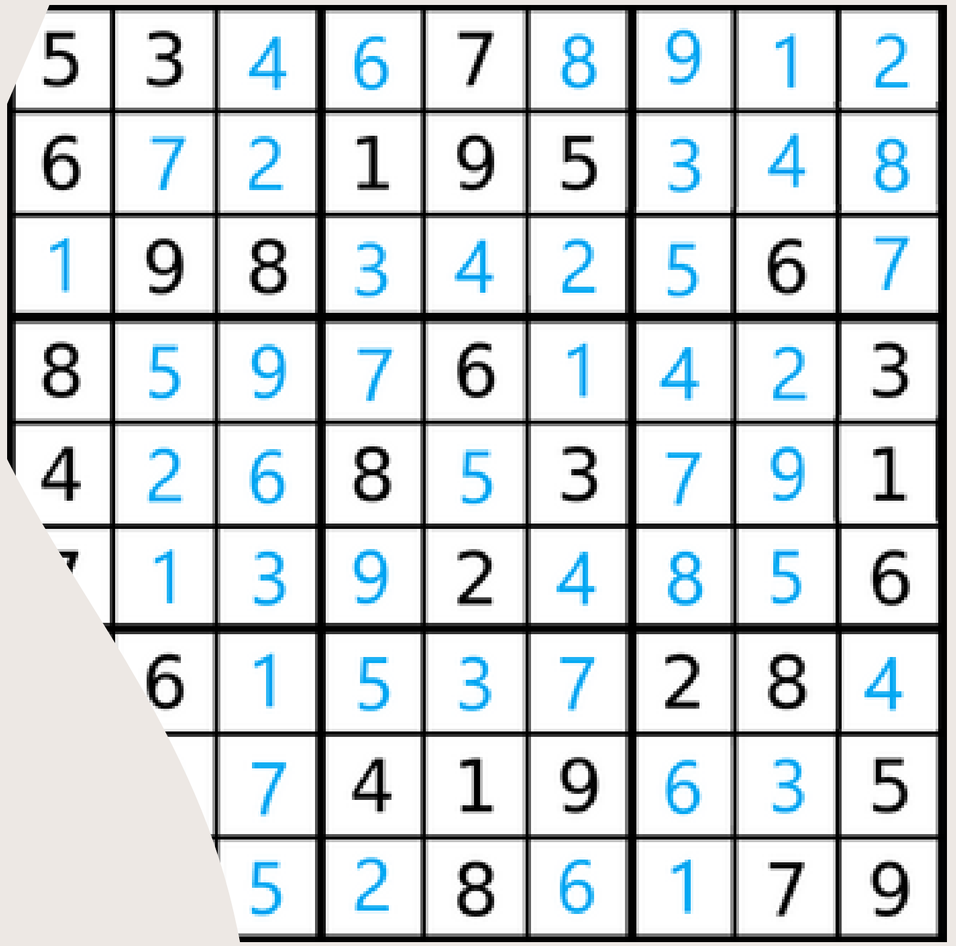
PRESENTACIÓN DEL SUDOKU

Sistemas Informáticos y Entornos de Desarrollo

TABLA DE **CONTENIDO**

¿QUÉ ES EL **SUDOKU?**

El Sudoku es un rompecabezas lógico de colocación de números que se popularizó en Japón en 1986 y se dio a conocer en el ámbito internacional en 2005 a través de los pasatiempos en los periódicos.
El objetivo es rellenar una cuadrícula de 9x9 celdas dividida en subcuadrículas de 3x3, con las cifras del 1 al 9 partiendo de algunos números ya dispuestos en algunas celdas.
No se debe repetir ninguna cifra en una misma fila, columna y subcuadrícula.



OBJETIVOS

01

Objetivo 01

Desarrollar una aplicación de escritorio en Java que permita a los usuarios jugar al clásico juego de Sudoku.

02

Objetivo 02

Realizar un análisis y documentación que abarque desde la definición de objetivos, pasando por la identificación de requisitos, hasta la definición de la arquitectura y casos de uso del sistema.

03

Objetivo 03

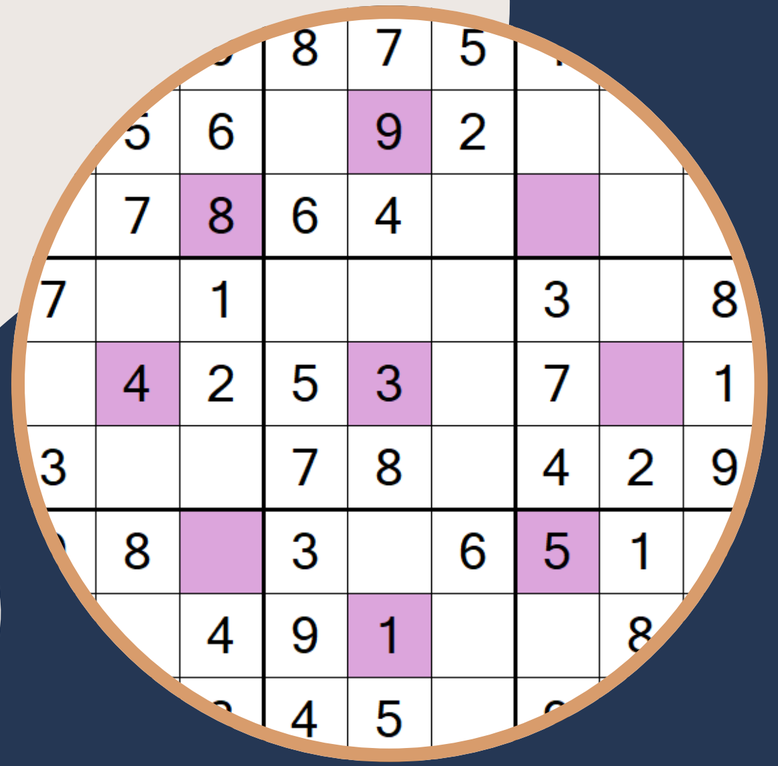
Generar tableros con diferentes niveles de dificultad, validar las jugadas del usuario y comprobar si el tablero ha sido completado correctamente.

04

Objetivo 04

Aplicar conocimientos de programación orientada a objetos, estructuras de datos, control de errores, e interfaces gráficas.

PROYECTO A
DESARROLLAR



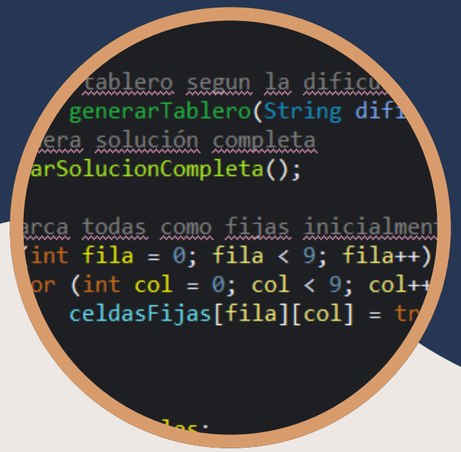
1 **Clase Sudoku**
- Atributos
- Métodos clave
  - Dificultad
  - Movimientos
  - Resultado
  - Mostrar tablero

2 **Clase GeneradorSudoku:**
- Métodos para generar tableros:
  - Fácil
  - Medio
  - Difícil

3 **Clase JuegoSudoku:**
- Método iniciar:
  - Elegir dificultad
  - Ver tablero
  - Finalizar

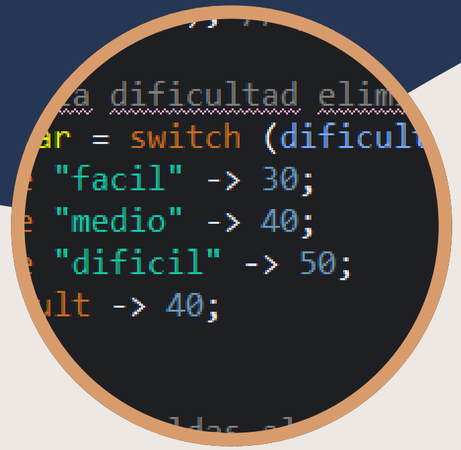
4 **Clase SudokuGUI:**
- Desarrollo de interfaz gráfica
- Interfaz intuitiva y accesible

MI PROYECTO



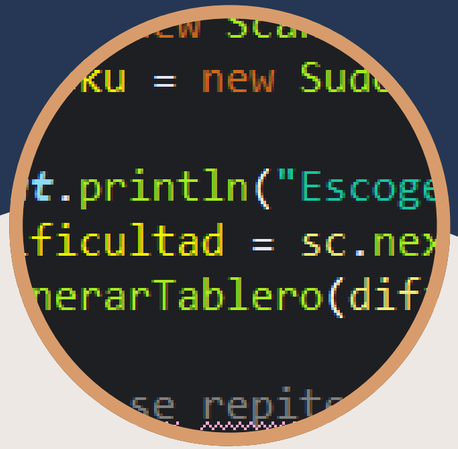
Clase Sudoku

- Para acceder al tablero
- Para la dificultad
- Señaliza las celdas
- Genera una solución
- Muestra el tablero



Clase GeneradorSudoku

- Genera un tablero
- Elimina casillas
- Si es válido

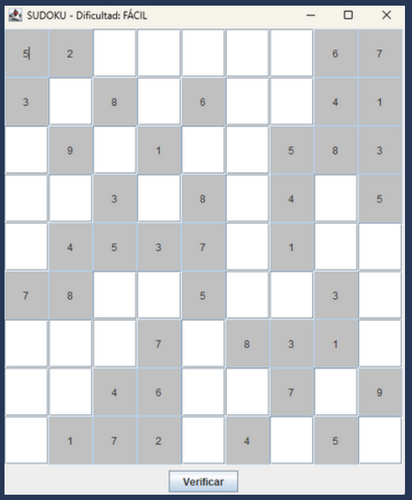


Clase JuegoSudoku

- Inicia
- Dificultad
- Finaliza

Clase SudokuGUI

- Interfaz gráfica
- Ventanas



PRUEBAS UNITARIAS

Test GenerarTableroFacil

Genera un tablero con la dificultad fácil y va validando

```
 testGenerarTableroFacil() {
Sudoku sudoku = new Sudoku();
sudoku.generarTablero("facil"); // Genera el
int[][] tablero = sudoku.getTablero(); // Ma
boolean[][] celdasFijas = sudoku.getCeldasFi

int visibles = 0; // Contador
```

Test DimensionesTablero

Verifica que haya 9 filas y 9 columnas con .length

```
ca que haya 9 filas y 9 column
ionesTablero() {
ku = new Sudoku();
lero = sudoku.getTablero();
s(9, tablero.length, "El table
fila : tablero) {
```

Test DificultadInvalida

Se ejecuta el programa con una dificultad inválida y da mensaje de error

```
 Ejecuta con dificultad inválida
tDificultadInvalida() {
ku sudoku = new Sudoku();
rtDoesNotThrow(() -> sudoku.gener
```

DOCUMENTACIÓN

Requisitos funcionales

ID	Descripción	Prioridad	Fuente	Estado
RF-01	El sistema debe generar un tablero Sudoku válido.	Alta	Usuario	Implementado
RF-02	El usuario puede rellenar celdas con números del 1 al 9.	Alta	Usuario	Implementado
RF-03	Validar si la solución introducida por el usuario es correcta.	Alta	Usuario	Implementado
RF-04	Resolver automáticamente el tablero actual.	Media	Usuario	Implementado

Requisitos no funcionales

ID	Descripción	Categoría	Métrica	Nivel Objetivo	Comentarios
RNF-01	La interfaz debe responder en menos de 1 segundo.	Rendimiento	Tiempo respuesta	< 1 segundo	Validado en entorno local.
RNF-02	Código documentado y comentado.	Mantenibilidad	Porcentaje doc.	≥ 90 %	Incluido en todas las clases.
RNF-03	Aplicación debe ejecutarse sin errores en Java 11+.	Compatibilidad	Versiones Java	Java 11 y 17	Probado con OpenJDK.

Casos de uso

ID	Nombre	Actor	Flujo Principal	Postcondición
CU-01	Iniciar partida	Usuario	1. Usuario abre aplicación 2. Sistema genera tablero	Tablero visible
CU-02	Introducir número en celda	Usuario	1. Usuario hace clic en celda 2. Ingresa número 3. Valida restricción	Número insertado si es válido
CU-03	Verificar solución	Usuario	1. Usuario pulsa botón "Verificar" 2. Sistema analiza y da resultado	Feedback mostrado al usuario
CU-04	Resolver tablero automáticamente	Usuario	1. Usuario pulsa botón "Resolver" 2. Algoritmo llena el tablero	Tablero resuelto completamente

Requisitos funcionales

Describen **qué** debe hacer el sistema.
- Ejemplos: iniciar sesión, buscar datos, procesar pagos.

Requisitos no funcionales

Describen **cómo** debe comportarse el sistema o restricciones del mismo.
- Ejemplos: rendimiento, usabilidad, seguridad, escalabilidad.

Casos de usos

Un caso de uso describe una **interacción** entre uno o varios usuarios y el sistema para lograr un objetivo concreto.
Sirve para:
- Capturar requisitos funcionales a nivel de usuario
- Guiar el diseño de interfaces y flujos
- Base para pruebas de aceptación

DOCUMENTACIÓN

Matriz de trazabilidad

Requisito	Clase / Módulo	Método Principal	Caso de Uso	Test Unitario
RF-01	GeneradorSudoku	`generar()`	CU-01	`testGeneracion()`
RF-02	Celda / InterfazGrafica	`setValor()`, `onClick()`	CU-02	`testEntradaCelda()`
RF-03	Sudoku	`verificar()`, `esValido()`	CU-03	`testVerificacion()`
RF-04	Solucionador	`resolver()`	CU-04	`testSolucionador()`
RNF-01	InterfazGrafica	`render()`	Todos	`testRendimiento()`
RNF-02	Todo el código	Comentarios JavaDoc	—	Revisión manual
RNF-03	Proyecto en general	Estructura Maven o Ant	—	Probado en varios JDK

Matriz de trazabilidad

La matriz de trazabilidad **relaciona** de forma bidireccional los **requisitos** con otros artefactos:

- Objetivos
- Casos de uso/diseño
- Casos de prueba

Ayuda a garantizar que todos los requisitos y objetivos estén cubiertos por diseño y pruebas, y facilita el control de cambios.

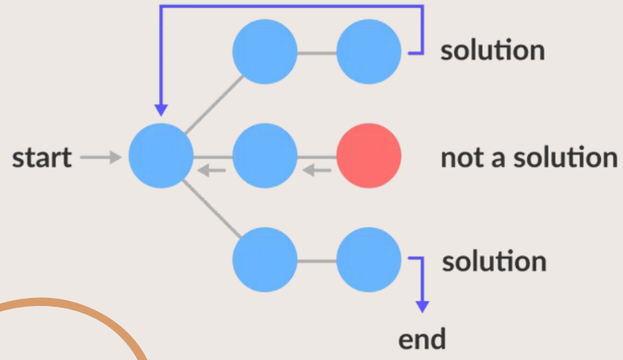
```
@startuml
package "proyectoFinal" {

  class Principal {
    + main(String[]): void
  }

  class SudokuGUI {
    - sudoku: Sudoku
    - campos: JTextField[][]
    + SudokuGUI(dificultad: String)
    + SudokuGUI()
  }

  class Sudoku {
    - tablero: int[][]
    - celdasFijas: boolean[][]
    + getTablero(): int[][]
    + getCeldasFijas(): boolean[][]
    + generarTablero(dificultad: String): void
    + colocarNumero(fila: int, columna: int, valor: int): boolean
    + estaResuelto(): boolean
    + esMovimientoValido(fila: int, columna: int, valor: int): boolean
    + mostrarTablero(): void
  }

  class SudokuException {
    + SudokuException(msg: String)
  }

  class MovimientoInvalidoException {
    + MovimientoInvalidoException(msg: String)
  }

  class EntradaFueraDeRangoException {
    + EntradaFueraDeRangoException(msg: String)
  }

  class CeldaFijaException {
    + CeldaFijaException()
  }

  SudokuException <|-- MovimientoInvalidoException
  SudokuException <|-- EntradaFueraDeRangoException
  SudokuException <|-- CeldaFijaException

  Principal --> SudokuGUI
  SudokuGUI --> Sudoku
  Sudoku --> SudokuException
}
@enduml
```

Diagramas de casos de usos

Un diagrama de casos de uso es una **representación gráfica** utilizada en el desarrollo de software que muestra la **relación** entre los usuarios finales (actores) y los distintos casos de uso, o funcionalidades, del sistema.

BACKTRACKING

El backtracking es una **técnica** algorítmica que funciona como un proceso de prueba y error: pruebas diferentes piezas, y cuando encuentras que una no funciona, retrocedes (o "backtrack" en inglés) y pruebas otra opción.



EXCEPCIONES

El uso de excepciones en Java le permite hacer que sus programas sean más robustos al crear "rutas de respaldo" y usar bloques catch para separar el código principal del código de manejo de excepciones.

```
 {
  new SudokuGUI(dificultad.toLowerCa
atch (Exception e) { // Esta excepc
  e.printStackTrace(); // Mostrar er
  JOptionPane.showMessageDialog(null

{
```

Elena Mena

MUCHAS GRACIAS

menaromero.elena@alumnos.ilerna.com