

## PROYECTO 2 - MANUAL TECNICO

202111134 Mario René Mérida Taracena

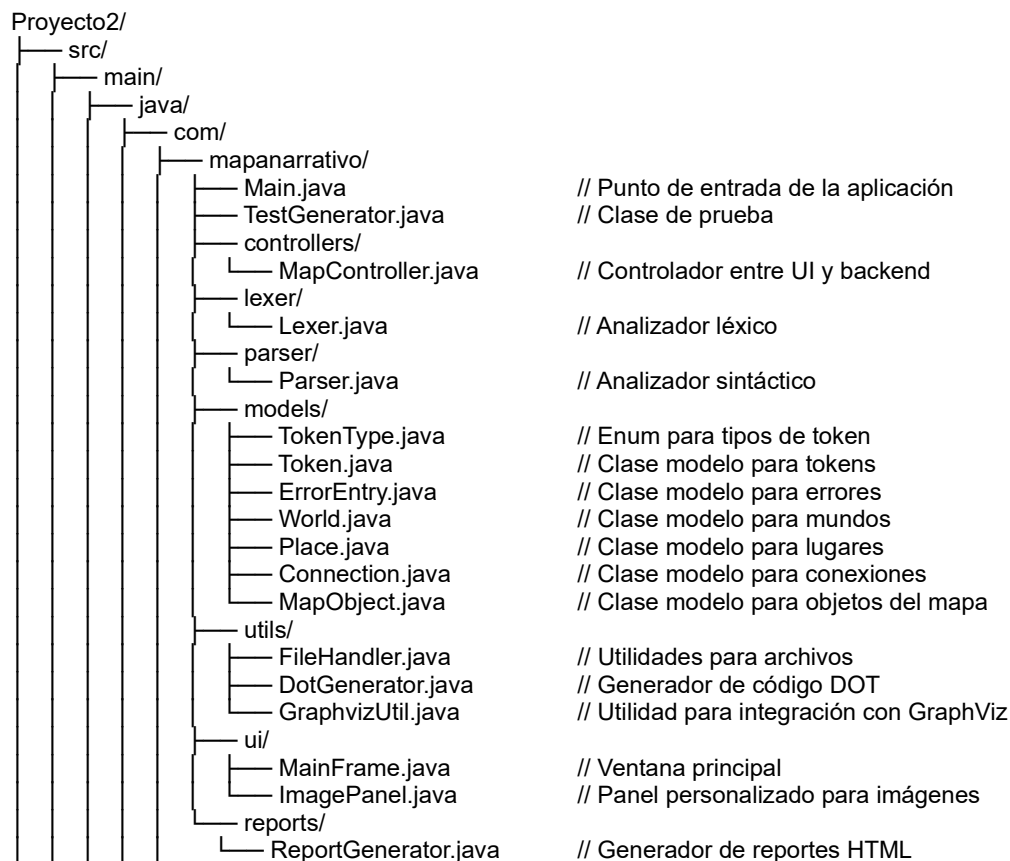
### 1. Introducción

El Generador Visual de Mapas Narrativos es una aplicación desarrollada en Java que permite analizar un lenguaje descriptivo para crear representaciones gráficas de mundos ficticios. La aplicación implementa un analizador léxico y sintáctico personalizado, sin usar expresiones regulares ni librerías que realicen estas funciones, para procesar el lenguaje de entrada y generar archivos en formato DOT para Graphviz.

### 2. Arquitectura del Sistema

La aplicación sigue un patrón de diseño Modelo-Vista-Controlador (MVC) para separar la lógica de negocio de la interfaz de usuario:

#### 2.1 Estructura del Proyecto



### 3. *Análisis Léxico y Sintáctico*

#### 3.1 Tokens y Expresiones Regulares

Aunque el proyecto especifica no usar la clase Regex de Java, a continuación se definen los patrones que el analizador léxico reconoce de forma manual:

Token	Descripción	Patrón (representación conceptual)
WORLD	Palabra clave "world"	world
PLACE	Palabra clave "place"	place
CONNECT	Palabra clave "connect"	connect
TO	Palabra clave "to"	to
WITH	Palabra clave "with"	with
OBJECT	Palabra clave "object"	object
AT	Palabra clave "at"	at
IDENTIFIER	Identificador	[a-zA-Z][a-zA-Z0-9_]*
STRING	Cadena entre comillas	"[^"]*"
NUMBER	Número entero	[0-9]+
LBRACE	Llave apertura	{
RBRACE	Llave cierre	}
LPAREN	Paréntesis apertura	(
RPAREN	Paréntesis cierre	)
COMMA	Coma	,
COLON	Dos puntos	:
EOF	Fin de archivo	\0

### 3.2 Implementación del Analizador Léxico

El analizador léxico (Lexer.java) procesa el texto de entrada carácter por carácter, identificando los tokens según los patrones definidos. Los principales métodos implementados son:

- advance(): Avanza al siguiente carácter en la entrada.
- skipWhitespace(): Ignora espacios en blanco.
- readIdentifier(): Lee un identificador completo.
- readNumber(): Lee un número entero.
- readString(): Lee una cadena entre comillas.
- tokenize(): Método principal que realiza el análisis léxico completo.

El método de árbol implementado para la construcción del analizador léxico sigue estos pasos:

1. Leer el carácter actual.
2. Si es un espacio en blanco, avanzar y continuar.
3. Si es una letra, leer un identificador y verificar si es una palabra clave.
4. Si es un dígito, leer un número.
5. Si es una comilla doble, leer una cadena.
6. Si es un carácter especial (como {, }, (, ), ,, :), crear el token correspondiente.
7. Si no coincide con ningún patrón, registrar un error léxico.

### 3.3 Gramática Libre de Contexto

La gramática que reconoce el lenguaje de entrada se define formalmente como sigue:

<Programa> ::= <ListaMundos>

<ListaMundos> ::= <Mundo> | <Mundo> "," <ListaMundos>

<Mundo> ::= "world" <String> "{" <ListaDefiniciones> "}"

<ListaDefiniciones> ::= <Definicion> | <Definicion> <ListaDefiniciones>

<Definicion> ::= <DefLugar> | <DefConexion> | <DefObjeto>

<DefLugar> ::= "place" <Identificador> ":" <Identificador> "at" "(" <Numero> "," <Numero> ")"

<DefConexion> ::= "connect" <Identificador> "to" <Identificador> "with" <String>

$\langle \text{DefObjeto} \rangle ::= \text{"object"} \langle \text{String} \rangle \text{"."} \langle \text{Identificador} \rangle \text{"at"} \langle \text{UbicacionObjeto} \rangle$

$\langle \text{UbicacionObjeto} \rangle ::= \langle \text{Identificador} \rangle \mid \text{"("} \langle \text{Numero} \rangle \text{"}, " \langle \text{Numero} \rangle \text{"}"$

$\langle \text{String} \rangle ::= \text{""} \langle \text{Caracteres} \rangle \text{"}"$

$\langle \text{Caracteres} \rangle ::= \langle \text{Caracter} \rangle \mid \langle \text{Caracter} \rangle \langle \text{Caracteres} \rangle$

$\langle \text{Caracter} \rangle ::= \text{cualquier carácter excepto "}"$

$\langle \text{Identificador} \rangle ::= \langle \text{Letra} \rangle \mid \langle \text{Letra} \rangle \langle \text{RestIdent} \rangle$

$\langle \text{RestIdent} \rangle ::= \langle \text{Letra} \rangle \langle \text{RestIdent} \rangle \mid \langle \text{Digito} \rangle \langle \text{RestIdent} \rangle \mid \text{"_"} \langle \text{RestIdent} \rangle \mid \epsilon$

$\langle \text{Letra} \rangle ::= \text{"a"} \mid \text{"b"} \mid \dots \mid \text{"z"} \mid \text{"A"} \mid \text{"B"} \mid \dots \mid \text{"Z"}$

$\langle \text{Numero} \rangle ::= \langle \text{Digito} \rangle \mid \langle \text{Digito} \rangle \langle \text{Numero} \rangle$

$\langle \text{Digito} \rangle ::= \text{"0"} \mid \text{"1"} \mid \text{"2"} \mid \text{"3"} \mid \text{"4"} \mid \text{"5"} \mid \text{"6"} \mid \text{"7"} \mid \text{"8"} \mid \text{"9"}$

### 3.4 Implementación del Analizador Sintáctico

El analizador sintáctico (Parser.java) implementa un parser descendente recursivo que sigue la gramática libre de contexto definida anteriormente. Los principales métodos implementados son:

- `parse()`: Método principal que inicia el análisis sintáctico.
- `parseWorld()`: Analiza la estructura de un mundo.
- `parsePlace()`: Analiza la definición de un lugar.
- `parseConnection()`: Analiza la definición de una conexión.
- `parseObject()`: Analiza la definición de un objeto.

El parser utiliza los siguientes métodos auxiliares:

- `advance()`: Avanza al siguiente token.
- `match(TokenType)`: Verifica si el token actual coincide con el tipo esperado.
- `expect(TokenType)`: Espera un tipo de token específico o registra un error.

## 4. Modelo de Datos

El sistema utiliza las siguientes clases para representar la información del mapa narrativo:

- `World`: Representa un mundo completo con lugares, conexiones y objetos.
- `Place`: Representa un lugar en el mapa con nombre, tipo y coordenadas.

- **Connection:** Representa una conexión entre dos lugares.
- **MapObject:** Representa un objeto que puede estar en un lugar o en coordenadas específicas.


## 5. Generación de Gráficos







La clase DotGenerator convierte los modelos de datos en código DOT para Graphviz, aplicando los estilos visuales específicos para cada tipo de elemento:

### 5.1 Estilos de Lugares

Tipo	Figura (shape)	Color (fillcolor)
playa	ellipse	lightblue
cueva	box	gray
templo	octagon	gold
jungla	parallelogram	forestgreen
montaña	triangle	sienna
pueblo	house	burlywood
isla	invtriangle	lightgoldenrod
río	hexagon	deepskyblue
volcán	doublecircle	orangered
pantano	trapezium	darkseagreen

### 5.2 Estilos de Objetos

Tipo	Figura (shape)	Color (fillcolor)	Emoji	Código Unicode
tesoro	box3d	gold		\uD83C\uDF81
llave	pentagon	lightsteelblue		\uD83D\uDD11
arma	diamond	orangered		\uD83D\uDDE1\uFE0F
objeto mágico	component	violet		\u2728

Tipo	Figura (shape)	Color (fillcolor)	Emoji	Código Unicode
poción	cylinder	plum		\u2697\uFE0F
trampa	hexagon	crimson		\uD83D\uDCA3
libro	note	navajowhite		\uD83D\uDCD5
herramienta	folder	darkkhaki		\uD83D\uDEE0\uFE0F
bandera	tab	white		\uD83D\uDEA9
gema	egg	deepskyblue		\uD83D\uDC8E

### 5.3 Estilos de Conexiones

Tipo	Línea (style)	Color
camino	solid	black
puente	dotted	gray
sendero	dashed	saddlebrown
carretera	solid	darkgray
nado	dashed	deepskyblue
lancha	solid	blue
teleférico	dotted	purple

## 6. Generación de Reportes

La clase ReportGenerator crea dos tipos de reportes en formato HTML:

### 6.1 Reporte de Tokens

Muestra información detallada sobre los tokens identificados durante el análisis léxico:

Token	Lexema	Línea	Columna
Cadena	"Campo minado"	1	6
Llave apertura (		8	10

Token	Lexema	Línea Columna	
Llave Cierre	)	12	1
num	10	12	14

## 6.2 Reporte de Errores

Muestra los errores léxicos y sintácticos encontrados durante el análisis:

### Error Línea Columna

i	10	7
¿	45	4

## 7. Interfaz Gráfica

La interfaz gráfica se implementa utilizando Swing, con los siguientes componentes principales:

- MainFrame: Ventana principal de la aplicación.
- ImagePanel: Panel personalizado para mostrar imágenes de mapas.

La interfaz incluye:

- Área de texto para mostrar el código del mapa.
- Área de imagen para visualizar el mapa generado.
- Botones para cargar archivos, limpiar, analizar y generar reportes.
- Selector de mapas para elegir el mapa a visualizar.

## 8. Integración con Graphviz

La aplicación utiliza la clase GraphvizUtil para interactuar con Graphviz:

1. Genera un archivo DOT temporal con el código generado.
2. Ejecuta el comando dot de Graphviz para convertir el archivo DOT a una imagen PNG.
3. Carga y muestra la imagen generada en la interfaz de usuario.

## **9. Proceso de Análisis y Generación**

El flujo completo de análisis y generación de mapas es el siguiente:

1. El usuario carga o ingresa el código del mapa.
2. Al presionar "Analizar Archivo", se realiza el análisis léxico para identificar tokens.
3. Los tokens se pasan al analizador sintáctico para validar la estructura y construir el modelo de datos.
4. Si el análisis es exitoso, se cargan los nombres de los mundos en el selector.
5. Al seleccionar un mundo, se genera el código DOT correspondiente.
6. El código DOT se procesa con Graphviz para generar la imagen del mapa.
7. La imagen se muestra en el área de imagen de la interfaz.

## **10. Manejo de Errores**

El sistema implementa recuperación de errores léxicos y sintácticos:

- **Errores Léxicos:** Cuando se encuentra un carácter no reconocido, se registra el error y se avanza al siguiente carácter.
- **Errores Sintácticos:** Cuando se encuentra una estructura sintáctica incorrecta, se registra el error y se intenta continuar con el análisis.

## **11. Requisitos del Sistema**

- Java Development Kit (JDK) 8 o superior
- Graphviz instalado y disponible en el PATH del sistema

## **12. Consideraciones para Desarrollo Futuro**

- Implementación de un editor con resaltado de sintaxis.
- Soporte para estilos personalizados de lugares y objetos.
- Exportación de mapas en diferentes formatos de imagen.
- Animación de rutas o recorridos en el mapa.

## **13. Conclusiones**

El Generador Visual de Mapas Narrativos demuestra la aplicación práctica de conceptos de lenguajes formales y técnicas de programación:



- Análisis léxico y sintáctico sin usar expresiones regulares predefinidas.
- Implementación de un analizador descendente recursivo.
- Generación de representaciones visuales a partir de descripciones textuales.
- Manejo de errores y recuperación durante el análisis.

La aplicación proporciona una herramienta útil para crear mapas visuales para historias, juegos, aventuras o experiencias educativas, facilitando la visualización de mundos ficticios descritos en texto.