Instituto Tecnológico de Costa Rica

Compiladores e interpretes

Proyecto LogoTec

Andres Blanco 2022108841

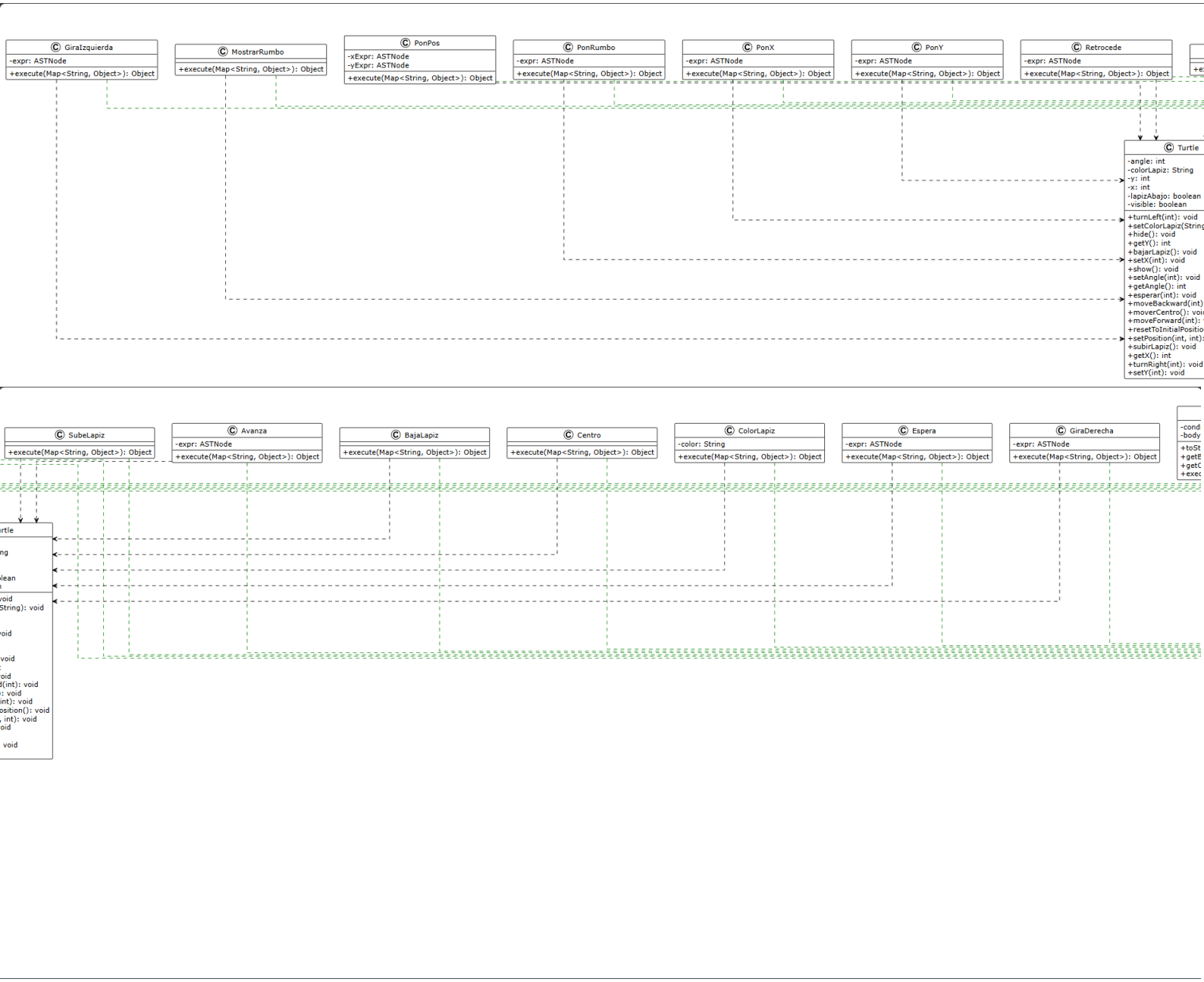
Giancarlo Vega 2020195338

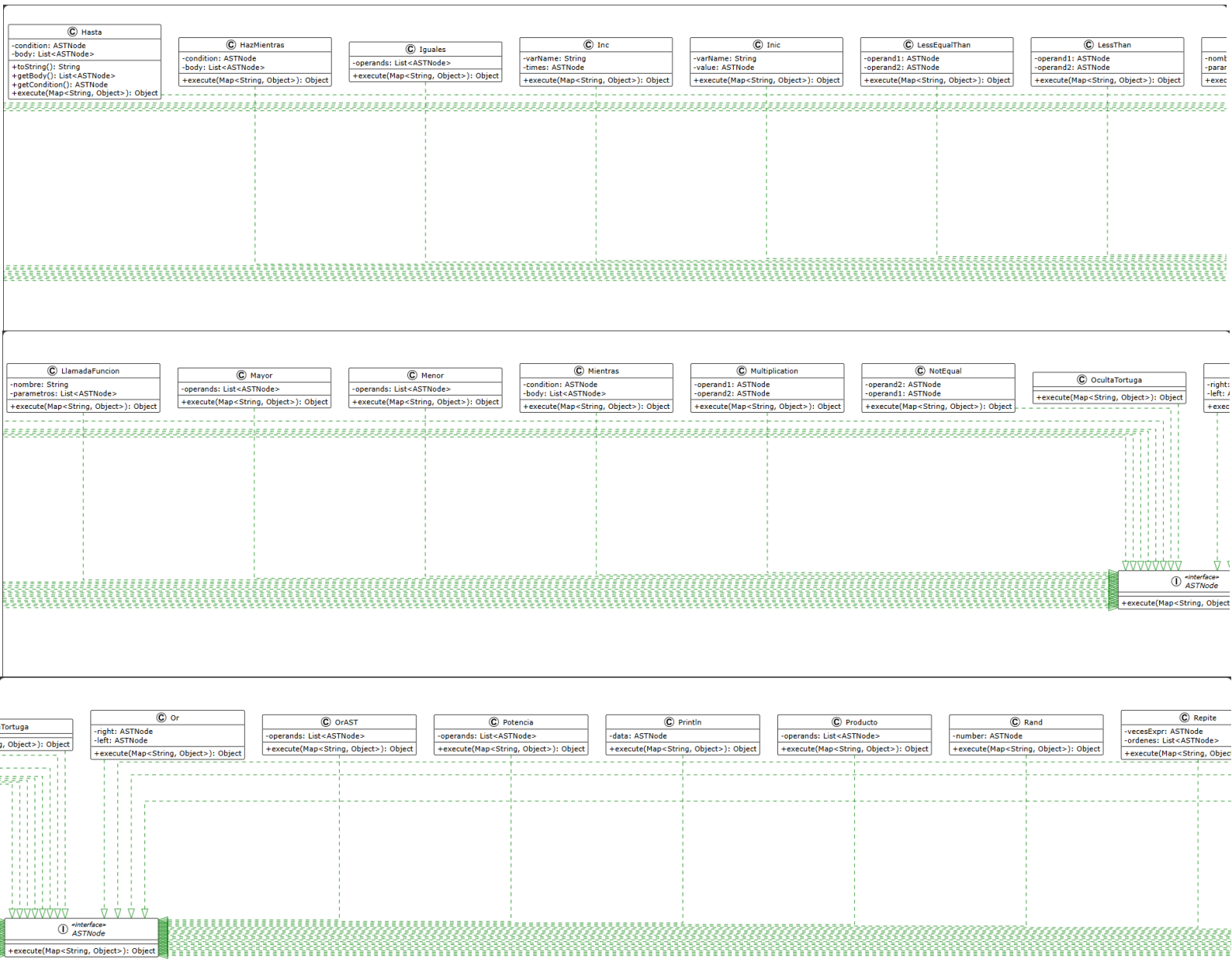
Sebastián Chaves 2021032506

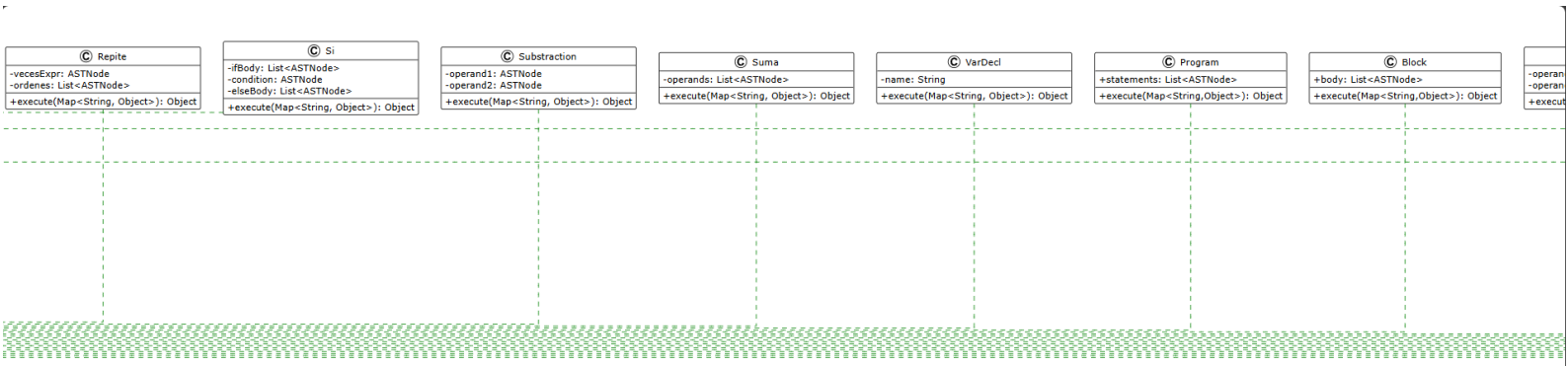
II Semestre

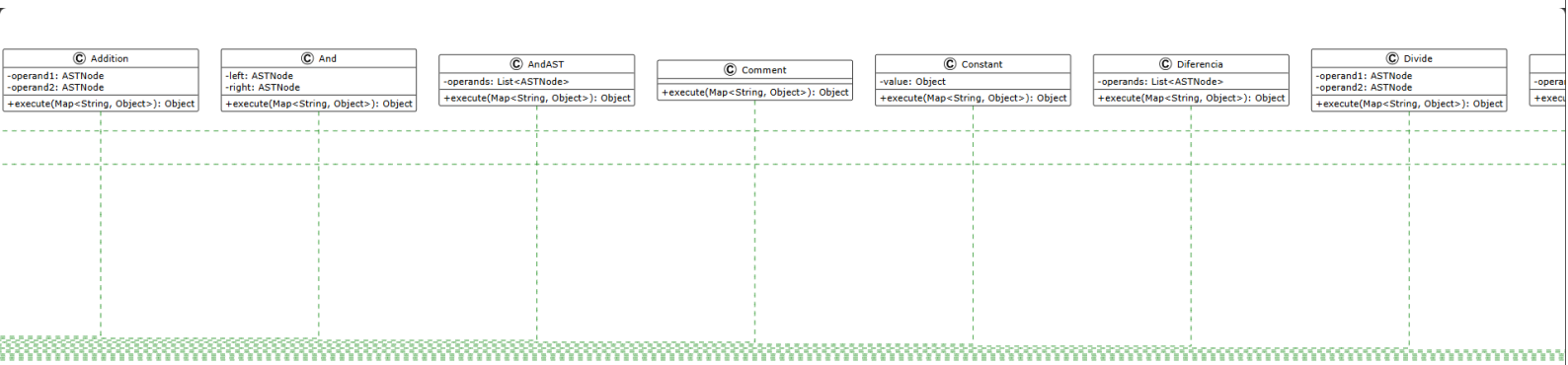
2025

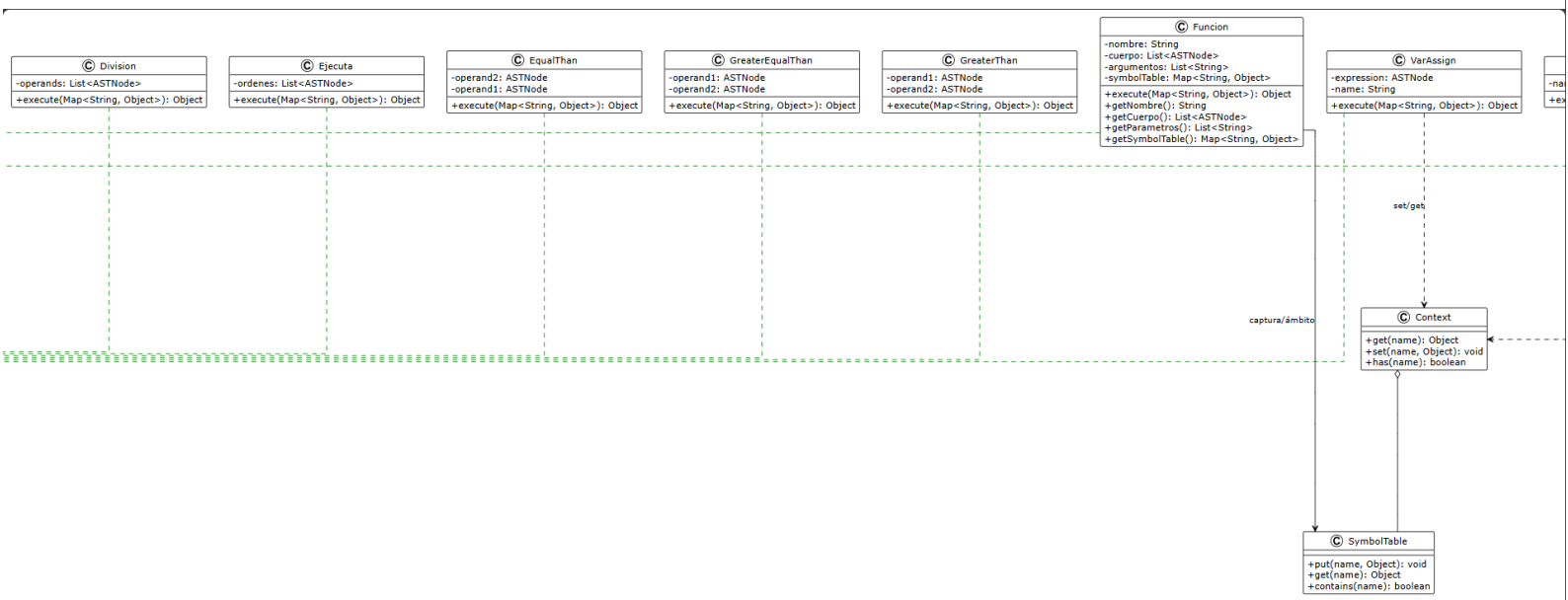
Diagrama de Arquitectura

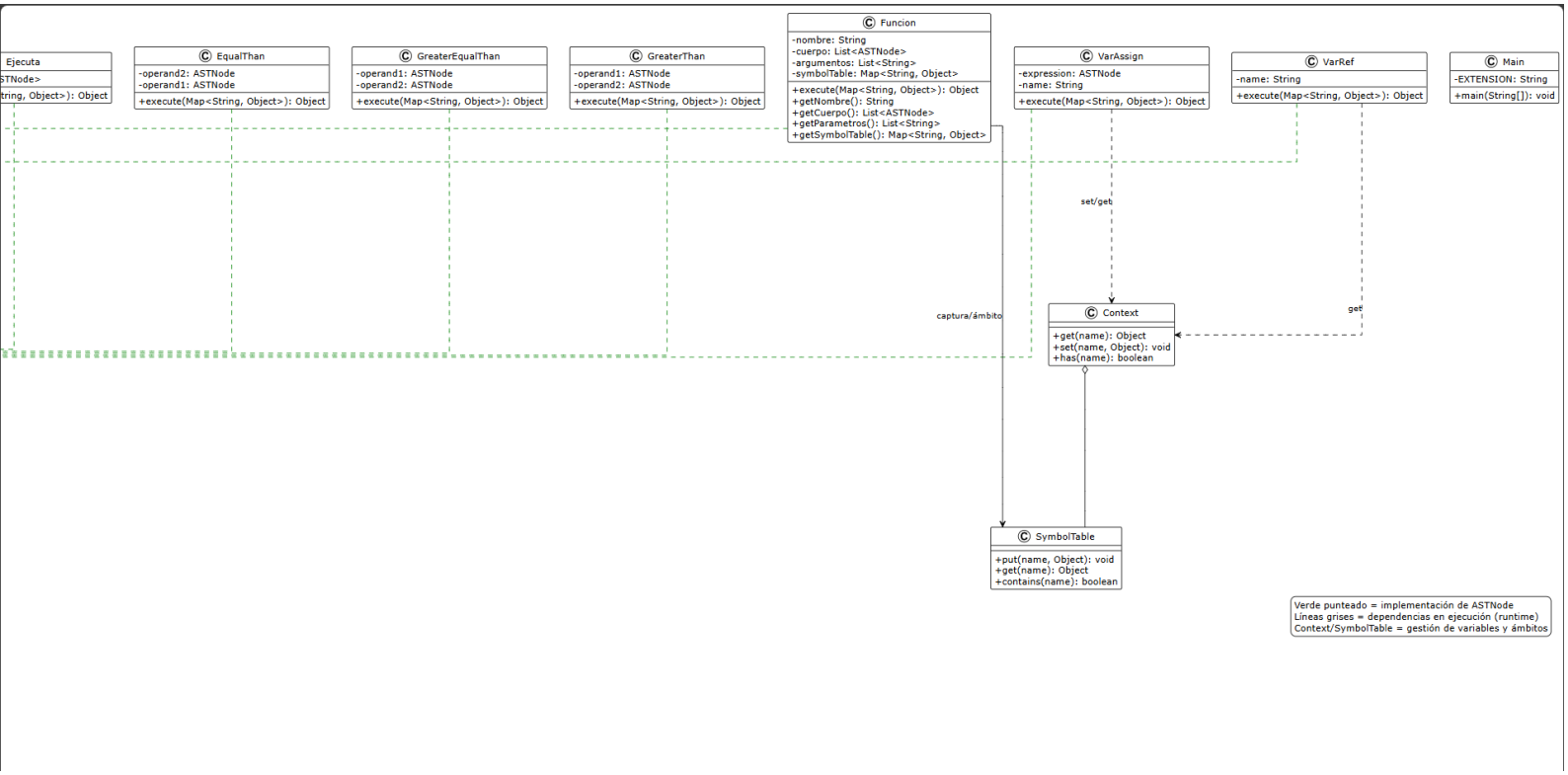



Hasta
-condition: ASTNode
-body: List<ASTNode>
+toString(): String
+getBody(): List<ASTNode>
+getCondition(): ASTNode
+execute(Map<String, Object>): Object
HazMientras
-condition: ASTNode
-body: List<ASTNode>
+execute(Map<String, Object>): Object
Iguales
-operands: List<ASTNode>
+execute(Map<String, Object>): Object
Inc
-varName: String
-times: ASTNode
+execute(Map<String, Object>): Object
Inic
-varName: String
-value: ASTNode
+execute(Map<String, Object>): Object
LessEqualThan
-operand1: ASTNode
-operand2: ASTNode
+execute(Map<String, Object>): Object
LessThan
-operand1: ASTNode
-operand2: ASTNode
+execute(Map<String, Object>): Object
LlamadaFuncion
-nombre: String
-parametros: List<ASTNode>
+execute(Map<String, Object>): Object
Mayor
-operands: List<ASTNode>
+execute(Map<String, Object>): Object
Menor
-operands: List<ASTNode>
+execute(Map<String, Object>): Object
Mientras
-condition: ASTNode
-body: List<ASTNode>
+execute(Map<String, Object>): Object
Multiplication
-operand1: ASTNode
-operand2: ASTNode
+execute(Map<String, Object>): Object
NotEqual
-operand2: ASTNode
-operand1: ASTNode
+execute(Map<String, Object>): Object
OcultaTortuga
+execute(Map<String, Object>): Object
«interface»
ASTNode
+execute(Map<String, Object
Or
-right: ASTNode
-left: ASTNode
+execute(Map<String, Object>): Object
OrAST
-operands: List<ASTNode>
+execute(Map<String, Object>): Object
Potencia
-operands: List<ASTNode>
+execute(Map<String, Object>): Object
Println
-data: ASTNode
+execute(Map<String, Object>): Object
Producto
-operands: List<ASTNode>
+execute(Map<String, Object>): Object
Rand
-number: ASTNode
+execute(Map<String, Object>): Object
Repite
-vecesExpr: ASTNode
-ordenes: List<ASTNode>
+execute(Map<String, Objec
«interface»
ASTNode
+execute(Map<String, Object>): Object


Repite
-vecesExpr: ASTNode
-ordenes: List<ASTNode>
+execute(Map<String, Object>): Object
Si
-ifBody: List<ASTNode>
-condition: ASTNode
-elseBody: List<ASTNode>
+execute(Map<String, Object>): Object
Substraction
-operand1: ASTNode
-operand2: ASTNode
+execute(Map<String, Object>): Object
Suma
-operands: List<ASTNode>
+execute(Map<String, Object>): Object
VarDecl
-name: String
+execute(Map<String, Object>): Object
Program
+statements: List<ASTNode>
+execute(Map<String,Object>): Object
Block
+body: List<ASTNode>
+execute(Map<String,Object>): Object


Addition
-operand1: ASTNode
-operand2: ASTNode
+execute(Map<String, Object>): Object
And
-left: ASTNode
-right: ASTNode
+execute(Map<String, Object>): Object
AndAST
-operands: List<ASTNode>
+execute(Map<String, Object>): Object
Comment
+execute(Map<String, Object>): Object
Constant
-value: Object
+execute(Map<String, Object>): Object
Diferencia
-operands: List<ASTNode>
+execute(Map<String, Object>): Object
Divide
-operand1: ASTNode
-operand2: ASTNode
+execute(Map<String, Object>): Object


Division
-operands: List<ASTNode>
+execute(Map<String, Object>): Object
Ejecuta
-ordenes: List<ASTNode>
+execute(Map<String, Object>): Object
EqualThan
-operand2: ASTNode
-operand1: ASTNode
+execute(Map<String, Object>): Object
GreaterEqualThan
-operand1: ASTNode
-operand2: ASTNode
+execute(Map<String, Object>): Object
GreaterThan
-operand1: ASTNode
-operand2: ASTNode
+execute(Map<String, Object>): Object
Funcion
-nombre: String
-cuerpo: List<ASTNode>
-argumentos: List<String>
-symbolTable: Map<String, Object>
+execute(Map<String, Object>): Object
+getNombre(): String
+getCuerpo(): List<ASTNode>
+getParametros(): List<String>
+getSymbolTable(): Map<String, Object>
VarAssign
-expression: ASTNode
-name: String
+execute(Map<String, Object>): Object
set/get
captura/ámbito
Context
+get(name): Object
+set(name, Object): void
+has(name): boolean
SymbolTable
+put(name, Object): void
+get(name): Object
+contains(name): boolean


Ejecuta
STNode>
tring, Object>): Object
EqualThan
-operand2: ASTNode
-operand1: ASTNode
+execute(Map<String, Object>): Object
GreaterEqualThan
-operand1: ASTNode
-operand2: ASTNode
+execute(Map<String, Object>): Object
GreaterThan
-operand1: ASTNode
-operand2: ASTNode
+execute(Map<String, Object>): Object
Funcion
-nombre: String
-cuerpo: List<ASTNode>
-argumentos: List<String>
-symbolTable: Map<String, Object>
+execute(Map<String, Object>): Object
+getNombre(): String
+getCuerpo(): List<ASTNode>
+getParametros(): List<String>
+getSymbolTable(): Map<String, Object>
VarAssign
-expression: ASTNode
-name: String
+execute(Map<String, Object>): Object
VarRef
-name: String
+execute(Map<String, Object>): Object
Main
-EXTENSION: String
+main(String[]): void
set/get
captura/ámbito
get
Context
+get(name): Object
+set(name, Object): void
+has(name): boolean
SymbolTable
+put(name, Object): void
+get(name): Object
+contains(name): boolean
Verde punteado = implementación de ASTNode
Líneas grises = dependencias en ejecución (runtime)
Context/SymbolTable = gestión de variables y ámbitos

Problemas conocidos:

1. **Precedencia y asociatividad de expresiones**

   Descripción: La gramática mezcla +, -, *, /, ^, comparadores y lógicos en niveles que aún producen parseos inesperados en casos mixtos.

   Impacto: Resultados semánticos incorrectos o difíciles de depurar.

   Evidencia: Expresiones como a + b > c && d no siempre respetan la intención del usuario.

   Estado/Workaround: Pruebas manuales; se recomienda paréntesis explícitos.

   Siguiente paso: Refactor por niveles (OR > AND > relacionales > suma/resta > mult/div > potencia > unarios) y suite de pruebas de precedencia.

2. **Tipado débil en la ejecución**

   Descripción: El contexto usa Map<String,Object>; no hay verificación de tipos centralizada.

   Impacto: Errores en tiempo de ejecución (p. ej., INC "hola") y mensajes poco claros.

   Evidencia: Excepciones genéricas en execute() de varios nodos.

   Estado/Workaround: Comprobaciones ad-hoc por nodo.

   Siguiente paso: Introducir Context/SymbolTable tipados y verificador semántico previo a ejecutar.

3. **Diagnósticos de errores insuficientes**

   Descripción: Falta un sistema uniforme con línea/columna, código y sugerencia.

   Impacto: Dificulta depuración y la UX en IDE/CLI.

   Evidencia: Mensajes desde RuntimeException sin ubicación.

   Estado/Workaround: Logs en consola.

   Siguiente paso: Módulo de diagnósticos (léxico, sintaxis, semántica) y estandarización de mensajes.

4. **Regla de “comentario en primera línea” frágil**

Descripción: La validación depende de espacios/BOM/CRLF y puede fallar según el editor.

Impacto: Rechazo de programas válidos o falsos positivos.

Evidencia: Archivos con BOM o CRLF en Windows.

Estado/Workaround: Reguardar en UTF-8 con LF; comentario manual.

Siguiente paso: Validar en lexer con modo inicial robusto (consumir BOM/espacios) y pruebas CRLF/LF.

5. **Ámbitos y "shadowing" en funciones**

   Descripción: Las funciones capturan/usan contexto global sin reglas claras de sombreado ni escritura en global.

   Impacto: Efectos laterales inesperados entre llamadas.

   Evidencia: Variables globales cambian tras invocar funciones.

   Estado/Workaround: Copia superficial del contexto.

   Siguiente paso: Definir contrato de alcance (solo lectura del global por defecto) y mecanismo explícito para escribir.

6. **Bucles potencialmente infinitos y control de tiempo**

   Descripción: Mientras, Haz.Mientras y Haz.Hasta pueden no terminar; ESPERA puede bloquear el hilo.

   Impacto: Congelamiento del intérprete; mala UX.

   Evidencia: Programas sin cambio de condición o esperas largas.

   Estado/Workaround: Interrupción manual.

   Siguiente paso: Watchdog de iteraciones/tiempo, cancel token y esperas no bloqueantes.

7. **Tortuga sin límites ni validación geométrica**

   Descripción: No existe mundo/canvas; PONPOS/PONX/PONY aceptan valores fuera de rango sin advertencia.

   Impacto: Resultados no reproducibles al migrar a GUI o hardware.

   Evidencia: Coordenadas arbitrarias aceptadas.

Estado/Workaround: Solo logs.

Siguiente paso: Definir sistema de coordenadas, límites y política de clamp/clip; avisos cuando se exceden.

8. **Estado global de la tortuga entre ejecuciones**

   Descripción: La instancia de Turtle puede persistir dentro del mismo proceso/IDE.

   Impacto: Una ejecución hereda posición/ángulo/lápiz de la anterior.

   Evidencia: La segunda corrida empieza donde terminó la primera.

   Estado/Workaround: Llamar a resetToInitialPosition() manualmente.

   Siguiente paso: Inicialización determinística al inicio de cada Program.execute().

9. **Sin GUI/canvas ni back-end .lgo (requisito futuro)**

   Descripción: Sprint 1 opera solo en consola; archivo objeto y ejecutor externo aún no existen.

   Impacto: No hay validación visual ni ejecución fuera del IDE.

   Evidencia: Solo salida textual.

   Siguiente paso: Diseñar IR/bytecode, especificar .lgo, crear stub de ejecutor y mock de canvas.

10. **Acoplamiento entre comandos y Turtle**

    Descripción: Nodos AVANZA/GIRA/etc. llaman directamente a Turtle; no hay interfaz abstracta.

    Impacto: Dificulta cambiar a GUI u hardware (ESP32) sin tocar AST.

    Evidencia: Dependencias directas Nodo → Turtle.

    Siguiente paso: Definir interfaz TurtlePort/ITurtle e inyectar implementación (consola/canvas/hardware).

11. **Semilla de aleatoriedad y reproducibilidad**

    Descripción: RAND no expone ni fija semilla.

    Impacto: Tests no deterministas.

Evidencia: Diferentes salidas con el mismo input.

Siguiente paso: Permitir setSeed() o semilla por configuración/CLI.

12. Cobertura de pruebas insuficiente

Descripción: No hay suites automatizadas por capas (léxico, parser, semántica, programas de sistema).

Impacto: Regresiones difíciles de detectar.

Evidencia: Validación manual con ejemplos.

Siguiente paso: CI con Gradle/JUnit; golden tests de entrada/salida.

13. Riesgo de StackOverflow por recursión profunda

Descripción: Evaluación recursiva del AST en programas muy anidados.

Impacto: Caídas en casos límite.

Evidencia: Profundidades artificiales reproducen el fallo.

Siguiente paso: Trampolining o iterativizar algunos nodos; límite de profundidad con diagnóstico.

14. Normalización de números y cadenas

Descripción: Tokenización de negativos, decimales y escapes de cadena incompleta en casos borde.

Impacto: Interpretaciones erróneas (p. ej., -0.5, \n, ").

Evidencia: Resultados variables en pruebas manuales.

Estado/Workaround: Usar paréntesis; evitar escapes complejos.

Siguiente paso: Reglas léxicas explícitas y batería de casos; fijar Locale.US.

15. Empaquetado/CLI y códigos de salida

Descripción: La CLI no define contrato claro (flags, rutas, encoding, exit codes).

Impacto: Automatización y CI frágiles.

Evidencia: Códigos de salida no diferenciados; mensajes sin estructura.

16. **Análisis de errores semánticos**

Descripción: Se intentó implementar un módulo centralizado para detección y manejo de errores semánticos, pero no fue posible lograr una solución robusta debido a la diversidad de estructuras del AST.

Impacto: Dificultad para mantener coherencia y trazabilidad en los mensajes de error entre distintos tipos de nodos.

Evidencia: Inconsistencias en la propagación de errores durante la evaluación de expresiones y ejecuciones de instrucciones.

Estado/Workaround: Se optó por manejar los errores de manera individual en cada clase del AST, permitiendo control granular.

Siguiente paso: Diseñar un sistema común de excepciones semánticas con herencia y contexto de nodo para unificar el manejo.

Conclusiones y recomendaciones:

- La separación **Front-end (lexer/parser/AST)** vs **Runtime** permitió un **MVP intérprete** estable y fácil de depurar.
- Un **AST** con contrato execute(ctx) hizo sencilla la extensión del lenguaje (comandos y control de flujo).
- El uso de Map<String,Object> aceleró el avance, pero evidenció **límites de tipado y diagnósticos**.
- La ausencia de **GUI/canvas** y de un **verificador semántico** reduce la observabilidad y calidad de errores.
- La base actual está lista para evolucionar a **IR/.lgo** y **ejecutor externo** sin romper el front-end.
- **Priorizar**: precedencia/gramática clara y **verificador semántico** con diagnósticos (línea:columna, código, hint).
- **Desacoplar** Turtle con una interfaz e incorporar **watchdog/cancelación** para bucles y ESPERA.
- **CI + pruebas** por capas y “golden tests”; permitir **semilla** para RAND y resultados reproducibles.

Referencias

Aho, A. V., Lam, M. S., Sethi, R., & Ullman, J. D. (2006). *Compilers: Principles, techniques, and tools* (2.ª ed.). Pearson. https://www.pearson.com/en-us/subject-catalog/p/compilers-principles-techniques-and-tools/P200000003472/9780133002140

ANTLR Project. (n.d.). *ANTLR 4 documentation*. GitHub. Recuperado el 4 de octubre de 2025, de https://github.com/antlr/antlr4/blob/master/doc/index.md

Appel, A. W. (2002). *Modern compiler implementation in Java*. Cambridge University Press. https://www.cambridge.org/core/books/modern-compiler-implementation-in-java/34EACED718B1D6D5237705F9BFD7CD4A

Brown, S. (n.d.). *The C4 model for visualising software architecture*. Recuperado el 4 de octubre de 2025, de https://c4model.com/

Cooper, K. D., & Torczon, L. (2022). *Engineering a compiler* (3.ª ed.). Morgan Kaufmann/Elsevier. https://shop.elsevier.com/books/engineering-a-compiler/cooper/978-0-12-815412-0

ISO/IEC/IEEE. (2022). *ISO/IEC/IEEE 42010:2022 — Software, systems and enterprise — Architecture description*. International Organization for Standardization. https://www.iso.org/standard/74393.html

LLVM Project. (n.d.). *LLVM language reference manual*. Recuperado el 4 de octubre de 2025, de https://llvm.org/docs/LangRef.html

Nystrom, R. (2021). *Crafting interpreters*. Genever Benning. https://craftinginterpreters.com/

Parr, T. (2013). *The definitive ANTLR 4 reference*. Pragmatic Bookshelf. https://pragprog.com/titles/tpantlr2/the-definitive-antlr-4-reference/

PlantUML Team. (n.d.). *PlantUML language reference guide*. Recuperado el 4 de octubre de 2025, de https://pdf.plantuml.net/PlantUML_Language_Reference_Guide_en.pdf