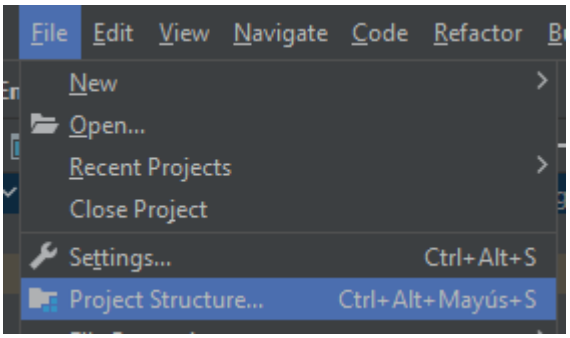
Informe de Etapa 4
Reentrega
COMPILADORES E INTERPRETES

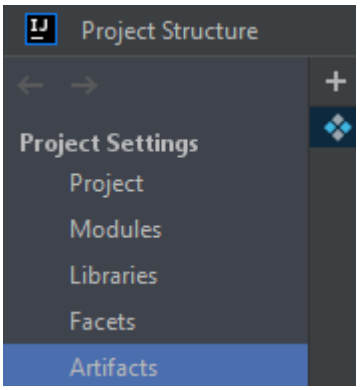
Alumno: Ilz Matias
Iu: 118683

Cómo compilar el código fuente y ejecutarlo

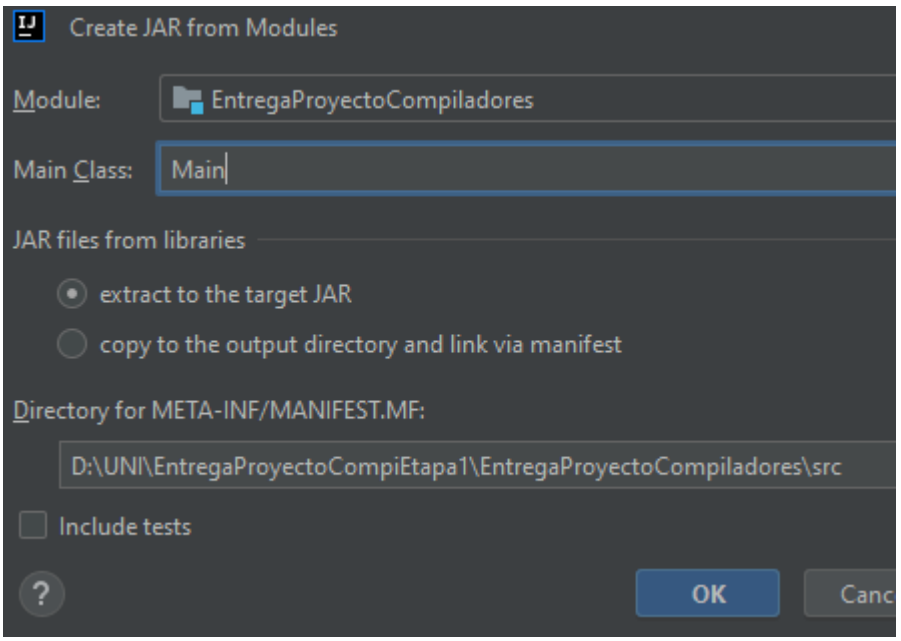
Para crear el jar del proyecto, dentro del mismo:
File > Project Structure



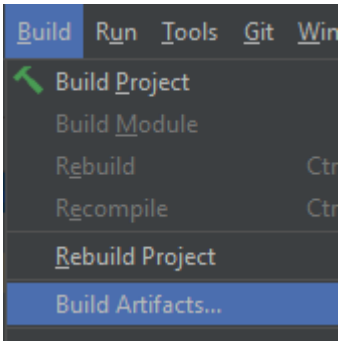
> Artifacts > +



> JAR > With dependencies... > Apply > OK



Build > Build Artifacts



Luego, desde la terminal del root, utilizo el comando -jar [Ruta del jar] [archivo a probar] . sin los corchetes.

Tipos de errores semánticos que puede detectar el analizador

Sobre Chequeos de Declaraciones:

-**Duplicación de Nombres de Clases**: Evita que se declaren dos clases con el mismo nombre.

-**Duplicación de Nombres de Métodos**: Garantiza que ninguna clase declare dos métodos con el mismo nombre.

-**Duplicación de Nombres de Variables de Instancia**: Evita que una clase defina dos variables de instancia con el mismo nombre.

-**Herencia Circular**: Detecta la herencia circular, lo que significa que una clase no puede heredar de sí misma ni de una jerarquía de clases donde la herencia forma un ciclo.

-**Constructores en Clases Concretas**: Garantiza que una clase concreta tenga solo un constructor o, si no tiene ninguno, le genera automáticamente un constructor predefinido sin parámetros y cuerpo vacío.

-**Métodos Sobrescritos**: Verifica que si un método se declara en una clase concreta y tiene el mismo nombre que un método en la superclase, sus modificadores, tipos de argumentos y tipo de retorno coinciden.

-**Sobrescritura de Métodos de Interfaces**: Garantiza que una clase concreta implemente todos los métodos de las interfaces que declara.

-**Nombres de Variables de Instancia en Clases Derivadas**: Evita que se defina una variable de instancia en una clase derivada con el mismo nombre que una variable de instancia en una clase ancestral.
-**Duplicación de Nombres de Parámetros**: En una unidad, asegura que ningún parámetro tenga el mismo nombre que otro parámetro.
-**Métodos Estáticos en Interfaces**: Detecta que las interfaces no pueden tener métodos estáticos.
-**Método Estático Main**: Verifica que al menos una clase contenga un método estático "main" sin parámetros.

En cuanto al Chequeo de Sentencias:

Uso de Nombres en el Cuerpo de Métodos y Constructores:

- Chequear que un nombre utilizado en una sentencia sea una variable, clase, método, etc.
- Controlar la existencia y recuperar información sobre entidades utilizadas en el cuerpo de unidades (variables locales, clases, métodos, etc.).

Resolviendo Nombres de Métodos:

- Chequear que un método llamado sea visible en el contexto de la clase actual.
- Buscar en la tabla de métodos de la clase que contiene la expresión donde se está utilizando el nombre.

- Manejar herencia: agregar todos los métodos heredados en la tabla de métodos.

Resolviendo Nombres de Variables y Parámetros:

- Determinar si se hace referencia a un parámetro o variable local del método.
- Buscar en la tabla de variables locales y parámetros asociada al método.
- Buscar en la tabla de atributos asociada a la clase si no es un parámetro.

Resolviendo Nombres de Atributos:

- Determinar la clase en la cual hay que buscar el atributo.
- Buscar en la tabla de atributos del tipo de la izquierda del encadenado.
- Manejar herencia: agregar todos los atributos heredados en la tabla de atributos.

Resolviendo Nombres de Constructores:

- Buscar en la tabla de símbolos de clase por la clase a la cual hace referencia el constructor.
- Verificar si existe el constructor al que se hace referencia.

Resolviendo Nombres de Clases:

- Verificar que el nombre sea de una clase existente en la tabla de clases.

Chequeos Semánticos de Expresiones y Sentencias:

Tipos Válidos:

- Definir tipos válidos: tipos primitivos (int, char, boolean) y tipos clase (concretos e interfaces).

- Subtipos: definir la relación de subtipos entre clases y tipos primitivos.

Chequeando Expresiones:

- Chequear tipos de subexpresiones.
- Propagar información de tipos hacia arriba en el AST.
- Chequear literales, expresiones unarias, variables, expresiones binarias, "this", paréntesis, llamadas a métodos y constructores, expresiones encadenadas punto, etc.

Chequeando Sentencias:

- Chequear que todas las expresiones utilizadas en las sentencias sean correctamente tipadas.
- Chequear sentencias vacías, sentencias de llamada, sentencias de asignación, sentencias de declaración de variables locales, sentencias "if", "while", "return", bloques, etc.

Chequeos Especiales:

- En métodos estáticos, controlar que no haya acceso a "this", variables de instancia o llamadas a métodos dinámicos.

Tipos de Chequeos

Chequeo de tipos:

- Identificaran si las expresiones usadas en las sentencias del cuerpo son correctamente tipadas

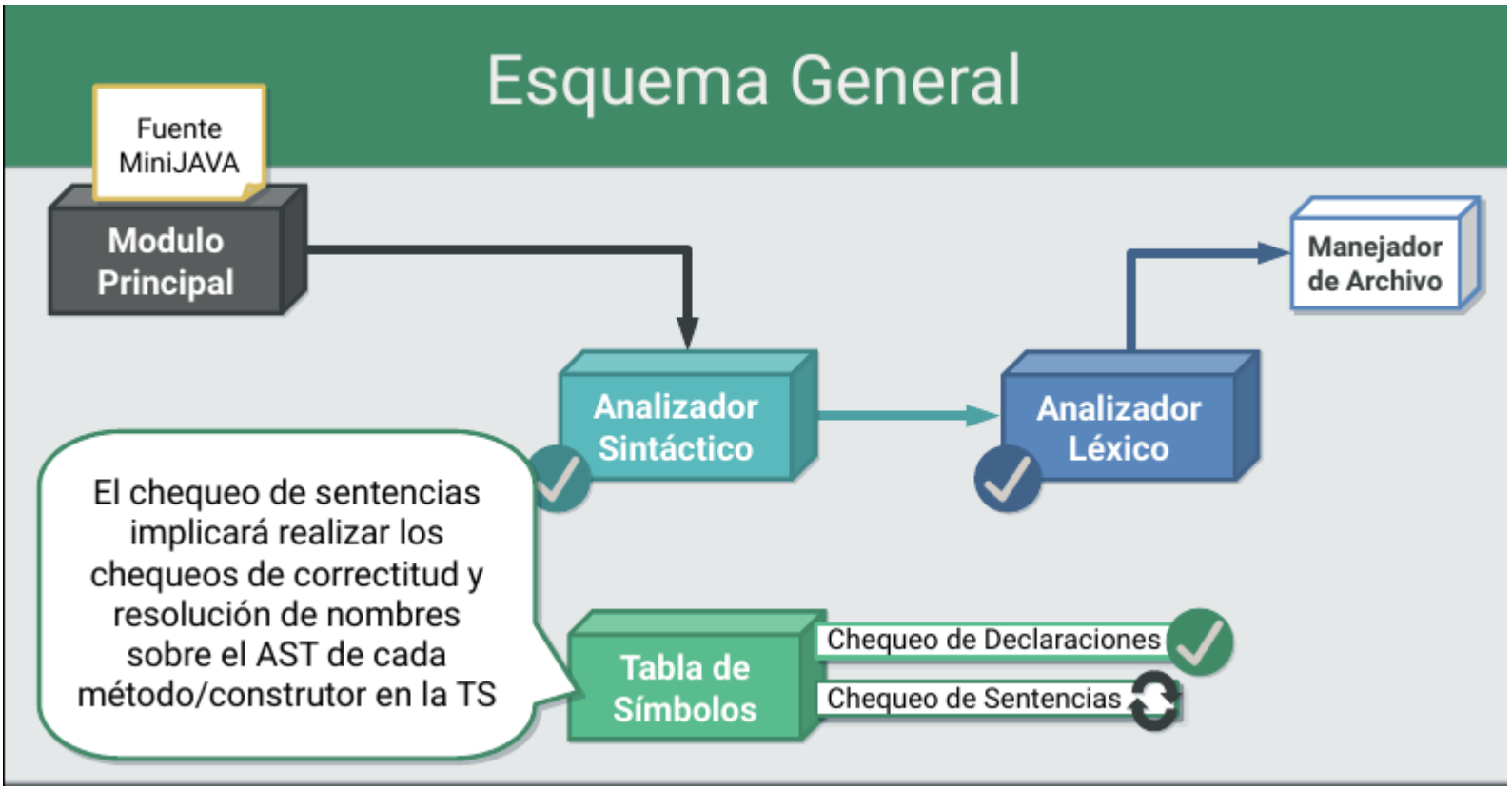
Resoluciones de nombres:

- identificaran a que entidad se hace referencia cuando se utiliza un identificador en el cuerpo de un método/constructor

Se realizan en conjunto, el chequeo de tipos dependerá de que los nombres que aparecen en las expresiones y sentencias que se están chequeando tienen que estar resueltos.

Decisiones de diseño para esta etapa

Clases



En la etapa anterior nos centramos en el Chequeo de Declaraciones, por lo que en esta etapa vamos a realizar el Chequeo de Sentencias.

Ahora se presentan los distintos Nodos que se utilizaron para poder realizar el Chequeo, con sus respectivas relaciones y usos.

NodoBase:

- Descripción: Interfaz que representa la base para todos los nodos del Árbol de Sintaxis Abstracta (AST).
- Métodos:
  - check(): Método para realizar el análisis semántico del nodo.
  - getType(): Método para obtener el tipo del nodo.
  - getToken(): Método para obtener el token asociado al nodo.

NodoExpBinaria:

- Descripción: Representa una expresión binaria, como una suma o una resta.
- Control: Verifica la compatibilidad de tipos y la aplicación correcta del operador binario.
- Relaciones: Tiene dos nodos hijos, uno para la expresión izquierda y otro para la expresión derecha.

NodoAsignacion:

- Descripción: Representa una sentencia de asignación.
- Control: Verifica la compatibilidad de tipos y si la variable de la izquierda es asignable.
- Relaciones: Tiene dos nodos hijos, uno para la expresión o variable de la izquierda y otro para la expresión de la derecha.

NodoBloque:

- Descripción: Representa un bloque de código que puede contener varias sentencias y definiciones de variables locales. Se utiliza para agrupar instrucciones dentro de una estructura de código más grande.
- Métodos:
  - check(): Realiza el análisis semántico del bloque, verificando las sentencias contenidas y lanzando una excepción si se encuentra una operación suelta (expresión binaria o unaria) dentro del bloque.
  - addSentencia(NodoBase s): Agrega una sentencia al bloque.
  - getType(): Obtiene el tipo del bloque (en este caso, siempre null).
  - getToken(): Obtiene el token asociado al bloque.
- Control:
  - Verifica la compatibilidad de tipos en las sentencias contenidas.
  - Evita la presencia de operaciones sueltas dentro del bloque.
- Relaciones:
  - Tiene una relación con los nodos NodoBase, NodoExpBinaria, NodoExpUnaria, Tipo, Atributo, Clase, Metodo y TablaDeSimbolos. Utiliza nodos NodoBase para representar las sentencias dentro del bloque. Utiliza Tipo para representar el tipo del bloque y lanzar excepciones. Utiliza Atributo para definir y acceder a variables locales.

Utiliza Clase y Método para acceder a atributos y parámetros de métodos. Utiliza TablaDeSimbolos para gestionar la información de los símbolos.

NodoExpAccesoConstructor:

- Descripción: Representa una instancia de acceso a un constructor de clase en una expresión.
- Control: Verifica la existencia de la clase a construir, la cantidad y compatibilidad de los argumentos del constructor.
- Relaciones: Se relaciona con otros nodos como NodoAcceso (padre) y NodoBloque. Además, interactúa con la tabla de símbolos para obtener información sobre la clase y su constructor.

NodoExpAccesoMetodoEstatico:

- Descripción: Representa la invocación de un método estático de una clase. Se utiliza cuando se llama a un método que pertenece a la clase en lugar de a una instancia específica de la clase.
- Control: Verifica si la clase a la que pertenece el método estática existe y si el método es realmente estático. Luego, compara la cantidad y tipos de los argumentos proporcionados con los parámetros del método estático.
- Relaciones: Tiene relación con la tabla de símbolos para obtener información sobre la existencia y estructura de la clase. Además, tiene nodos hijos que representan los argumentos proporcionados al método estático.

NodoExpBinaria:

- Descripción: Representa una expresión binaria, como una suma, resta, multiplicación, división, módulo u operaciones lógicas y relacionales.
- Control: Verifica la compatibilidad de tipos entre los operandos, dependiendo del operador binario utilizado. Se controla que los operandos sean de tipo entero para operadores aritméticos y relacionales, de tipo booleano para operadores lógicos y de tipo compatible para operadores de igualdad.
- Relaciones: Tiene dos nodos hijos, uno para la expresión izquierda y otro para la expresión derecha, representando los operandos de la operación binaria. Además, tiene relación con la tabla de símbolos para obtener información sobre los tipos de los operandos y el resultado de la operación.

NodoExpInterface:

- Descripción: Define la estructura común para los nodos que representan expresiones en el árbol de sintaxis abstracta.
- Control: No realiza control específico, se trata de una interfaz que define métodos comunes para los nodos de expresiones.
- Relaciones: No tiene relaciones directas con otros nodos, pero se espera que sea implementada por nodos específicos de expresiones, como NodoExpBinaria, NodoExpUnaria, etc. Esta interfaz extiende la interfaz más general, NodoBase.

NodoExpLit:

- Descripción: Representa expresiones literales, como booleanos, nulos, enteros, caracteres y cadenas de texto.
- Control: Verifica el tipo de la expresión literal y devuelve el correspondiente TipoPrimitivo o TipoReferencia.
- Relaciones: No tiene relaciones directas con otros nodos en términos de jerarquía de clases, pero puede ser parte de expresiones más grandes en el árbol de sintaxis abstracta que involucren operaciones con literales. Este nodo implementa la interfaz NodoExpInterface.

NodoExpParentizada:

- Descripción: Representa una expresión parentizada, es decir, una expresión encerrada entre paréntesis.
- Control: Verifica y devuelve el tipo de la expresión dentro de los paréntesis.
- Relaciones: Hereda de NodoAcceso, pero no tiene relaciones directas con otros nodos en términos de jerarquía de clases. Puede ser parte de expresiones más grandes en el árbol de sintaxis abstracta que involucren operaciones o estructuras que requieran paréntesis.

NodoExpThis:

- Descripción: Representa la palabra clave "this" en el código fuente, que se refiere a la instancia actual de la clase en la que se encuentra.
- Control: Verifica si el uso de "this" es válido dentro de una clase y no se utiliza en un método estático. Devuelve el tipo de referencia asociado con "this".
- Relaciones: Este nodo hereda de NodoAcceso y está relacionado con la clase actual y el método actual en el contexto semántico. Puede ser parte de expresiones más grandes en el árbol de sintaxis abstracta que involucren el uso de "this" para acceder a atributos o métodos de la instancia actual.

NodoExpUnaria:

- Descripción: Representa una expresión unaria en el código fuente, que consiste en un operador unario aplicado a una expresión.
- Control: Verifica la conformidad de tipos entre el operador unario y la expresión a la que se aplica. Puede realizar operaciones unarias como negación lógica ("!") y operadores aritméticos unarios ("+" y "-"). En el caso de operadores aritméticos unarios, se asegura de que la expresión sea de tipo entero.
- Relaciones: Este nodo tiene una relación con la expresión sobre la cual se aplica el operador unario. Puede formar

parte de expresiones más grandes en el árbol de sintaxis abstracta que involucren operaciones unarias.

NodoIf:

- Descripción: Representa una estructura condicional "if" en el código fuente.
- Control: Verifica que la condición del "if" sea de tipo booleano. Luego, verifica y ejecuta la sentencia asociada al "if". También puede incluir una rama "else" representada por otra sentencia.
- Relaciones: Este nodo tiene relaciones con la condición evaluada en el "if", la sentencia ejecutada si la condición es verdadera y la sentencia ejecutada si la condición es falsa (en caso de tener un "else"). Puede formar parte de bloques más grandes en el árbol de sintaxis abstracta que involucren estructuras condicionales.

NodoRetorno:

- Descripción: Representa una sentencia de retorno en un método.
- Control: Verifica la consistencia de la sentencia de retorno en relación con el tipo de retorno del método actual. Se asegura de que, si hay una expresión asociada al retorno, su tipo sea compatible con el tipo de retorno del método.
- Relaciones: Este nodo tiene una relación directa con la expresión que se está retornando, el método actual en el que

se encuentra y el bloque padre al que pertenece la sentencia de retorno. Puede formar parte de la estructura general de un bloque de método en el árbol de sintaxis abstracta.

NodoSentencia:

- Descripción: Interfaz que indica que un nodo pertenece a la categoría de sentencia en la estructura del árbol de sintaxis abstracta.
- Control: No tiene un control específico, ya que es una interfaz. Los nodos que implementan esta interfaz representan diversas sentencias en el código fuente.
- Relaciones: No tiene relaciones específicas con otros nodos, pero los nodos que implementan esta interfaz pueden formar parte de la estructura general del árbol de sintaxis abstracta y están asociados con la ejecución de instrucciones en un programa.

NodoVarLocal:

- Descripción: Representa una declaración de variable local.
- Control: Verifica la compatibilidad de tipos entre la expresión asignada a la variable y el tipo de la variable local. Agrega la variable local a la tabla de símbolos del bloque actual.
- Relaciones:
- Relación con otros nodos: Puede tener una relación con el nodo que representa la expresión asignada a la variable local (NodoBase expresion).

- Relación con el árbol de sintaxis abstracta: Forma parte de la estructura general del árbol de sintaxis abstracta y está asociado con la declaración y asignación de variables locales en un programa.
- Relación con la ejecución del programa: Está asociado con la ejecución de la declaración y asignación de variables locales durante la ejecución del programa.

NodoWhile:

- Descripción: Representa una estructura de bucle while.
- Control: Verifica que la condición del bucle sea de tipo booleano. Realiza el control semántico de la sentencia asociada al bucle.
- Relaciones:
- Relación con otros nodos: Tiene relaciones con los nodos que representan la condición del bucle (NodoBase condicion) y la sentencia que se ejecuta dentro del bucle (NodoBase sentencia).
- Relación con el árbol de sintaxis abstracta: Forma parte de la estructura general del árbol de sintaxis abstracta y está asociado con la implementación de bucles while en un programa.
- Relación con la ejecución del programa: Está asociado con la ejecución repetida de la sentencia mientras la condición sea verdadera.

TablaDeSimbolos: Se utiliza para almacenar información sobre símbolos, en esta etapa nos centramos en almacenar una colección de clases, interfaces, métodos, atributos y parámetros para garantizar que el código cumple con las reglas semánticas del lenguaje y para detectar errores y ambigüedades en el programa.

SemanticoExcepcion: En esta etapa las excepciones se presentan a partir de extender de la clase Exception, guardando el token y el mensaje que causó la excepción.

Main (Módulo Principal): Esta clase se encarga de:
-Asociar el código fuente al gestor de archivos y el gestor al analizador léxico.
-Crear el analizador Sintáctico con dependencia al analizador Léxico. En esta etapa, inicializa el analizador Sintáctico con las modificaciones para procesar declaraciones de clases, para ir agregando clases a la tabla de símbolos de clase. Cuando procese las declaraciones de miembros de una clase (métodos y atributos) los agrega a las tablas de métodos y atributos de esa clase. Además, en particular, cuando procese las declaraciones de una unidad se agregarán todos los parámetros a la tabla de parámetros de esa unidad.
-Si el análisis finaliza sin errores muestra por pantalla el mensaje correspondiente con el código de éxito. Caso contrario si se produce un error semántico, sintáctico o léxico durante el análisis, se va a propagar una excepción la cual va a ser

capturada por el Main, mostrando el mensaje de error correspondiente acompañado del código de error.