# INFORME
# ETAPA 4
# ANALIZADOR SEMÁNTICO II
# COMPILADORES E INTÉRPRETES

*Razuc Gonzalo*

## Aclaraciones

Los nombres utilizados para hacer referencia a cada nodo del AST dentro del informe no se corresponden con los nombres utilizados en la implementación de cada clase en Java para realizar el análisis semántico.

En un nodo llamada, en caso de que esta contenga un encadenado, primero se verifica que el último elemento de este sea “llamable”, es decir, el último elemento del encadenado no puede ser, por ejemplo, una variable. Luego del control mencionado, se chequea el nodo de la llamada, por lo que, por ejemplo, si dentro de un método con alcance estático se realiza la siguiente llamada:

```
static void método() {
        this.x;
}
```

siendo x una variable de instancia de la clase donde está implementado el método, el error que se lanzará será que x no es “llamable” (la llamada es incorrecta), en lugar de señalar que un acceso en un bloque de un método estático no puede comenzar con this.

# Controles realizados en cada tipo de nodo

## Nodos sentencia:

**Nodo asignación:**
Se controla que el lado izquierdo de la asignación sea asignable.
Si el lado izquierdo es un encadenado, se controlará que el último nodo del encadenado sea asignable. También, se controla que el lado derecho de la asignación sea compatible con el lado izquierdo, y que ambos lados sean compatibles con el operador de la asignación.

**Nodo bloque:**
Se controla que cada sentencia sea válida. También, se realiza el control sobre los nombres de las variables locales que un bloque contiene, verificando que no existan nombres repetidos, ni de otra variable ni de un parámetro.

**Nodo llamada:**
Posee un nodo acceso como atributo y se controla que este sea "llamable". Si el acceso tiene un encadenado, el último nodo del encadenado deberá poder ser llamable. En caso contrario, se controla que el nodo acceso sea llamable.

**Nodo if:**
Se controla que la condición (expresión) sea correcta, es decir, que sea de tipo boolean. En caso de que el nodo contenga una sentencia else, se controla que la sentencia del else sea correcta. En caso contrario, solo se controlará la sentencia del then.

**Nodo declaración de variable local:**
Se verifica que el nombre utilizado para la declaración de la variable local no sea un parámetro del método en cuestión. Se realiza el chequeo sobre la expresión asociada a la declaración controlando que se pueda inferir el tipo de esta, es decir, que sea un tipo primitivo o un tipo referencia.

**Nodo return:**
Se controla que el tipo de la expresión asociada al nodo sea compatible con el tipo de retorno del método que se esté chequeando. En caso de que la expresión asociada al return sea nula, significa que se trata de una expresión vacía,por lo que habrá que chequear que el método no tenga retorno. Caso contrario, se verificará la compatibilidad entre el tipo de la expresión asociada al return y el tipo de retorno del método (en caso de que el método no tenga retorno, los tipos no serán compatibles).
También, se verifica que el Constructor no tenga retorno, en dicho caso, reportará un error semántico.

*Razuc Gonzalo*

**Nodo while:**
Se controla que la expresión (condición) asociada al nodo sea de tipo boolean. También, se chequea la sentencia asociada al while.

## Nodos expresión

**Nodo expresión binaria:**
Se controla que los tipos del lado izquierdo y del lado derecho sean compatibles con el operador asociado a la expresión y que los tipos sean compatibles entre sí.

**Nodo expresión unaria:**
Se verifica que el operando y operador asociados a la expresión sean compatibles.

En cada nodo operando lo que se hace al chequear es retornar su tipo. Por ejemplo, *el nodo boolean retorna un tipo primitivo boolean*.

## Nodos Acceso

**Nodo acceso constructor:**
Se controla que el nombre con el que se invocó al constructor sea una clase concreta declarada y que exista el constructor dentro de esa clase. Si el constructor tiene un encadenado, se invoca al método que chequea al encadenado pasándole por parámetro un tipo referencia que se corresponde con el tipo del constructor invocado.

**Nodo acceso a método:**
Se controla que sea un método de la clase. Además, si el método que se está chequeando tiene alcance estático, se realiza el chequeo de que la llamada a método no llame a un método con alcance dinámico. Por último, se realiza el chequeo sobre la cantidad y tipos entre los argumentos formales y actuales.Si el acceso a método tiene un encadenado, se controla que el método accedido retorne un tipo no primitivo.

**Nodo expresión parentizada:**
Realiza el chequeo sobre la expresión asociada al nodo.

**Nodo acceso a método con alcance estático:**
Se controla que el nombre de la clase con el que se invocó a un método, sea una clase concreta declarada. También, se controla que el método invocado sea un método de esa clase y que su alcance sea estático. Por último, se realiza el chequeo entre la cantidad y los tipos de los argumentos formales y actuales.
En caso de que el acceso a método estático tenga encadenado, se chequea que el método en cuestión retorne un tipo no primitivo.

**Nodo acceso this:**
Se controla que el método en el cual se realizó el acceso this no tenga alcance estático.
En caso de que la palabra reservada this tenga un encadenado, se invoca al método que chequea el encadenado, pasándole por parámetro un tipo referencia asociado al nombre de la clase que invocó al this.

**Nodo acceso variable:**
Se controla que la variable sea un parámetro del método que está siendo chequeado, que sea una variable local del bloque actual que se está chequeando, o que sea un atributo de instancia de la clase en la que se implementa el método. En caso de ser un atributo, se controla que el método no tenga alcance estático ya que un método estático no puede acceder a un atributo de instancia.
En caso de que el acceso a variable tenga encadenado, se controla que el tipo de la variable no sea primitivo.

## Nodos Encadenado

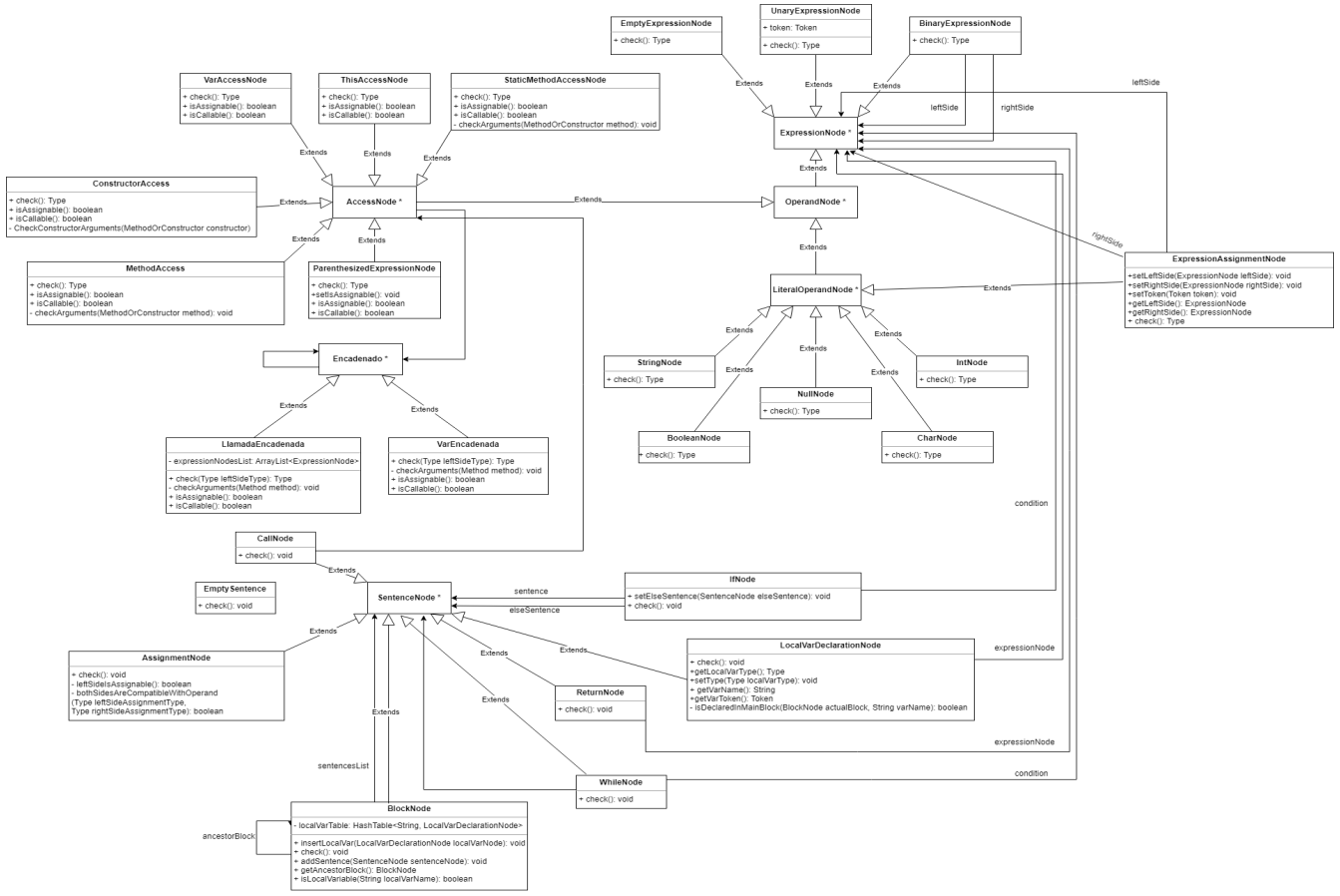
**Nodo variable encadenada:**
Se controla que el tipo del lado izquierdo de la variable encadenada sea una clase concreta. También, se controla que el nombre de la variable se corresponda con un atributo de la clase asociada al tipo del lado izquierdo de la variable encadenada.
Por último, si el tipo de la variable encadenada no es primitivo y la variable tiene un encadenado, se invoca al método que chequea el encadenado pasándole el tipo de la variable como parámetro. En caso contrario, se lanza una excepción.

**Nodo llamada encadenada:**
Se controla que la clase o interface asociada al tipo recibido como parámetro contenga el método invocado y que la cantidad y tipo de parámetros de la invocación coincidan con el encabezado del método.
En caso de que la llamada encadenada contenga un encadenado, se controla que el retorno del método invocado no sea primitivo. En caso de no serlo, se realiza la llamada al método que chequea el encadenado de la llamada encadenada, pasándole por parámetro el tipo de retorno del método que fue invocado.

Además, se controlan todos los casos propuestos en el PDF llamado "*MiniJava-Semantica-Completo*"

***Razuc Gonzalo***

# Diagrama de los AST

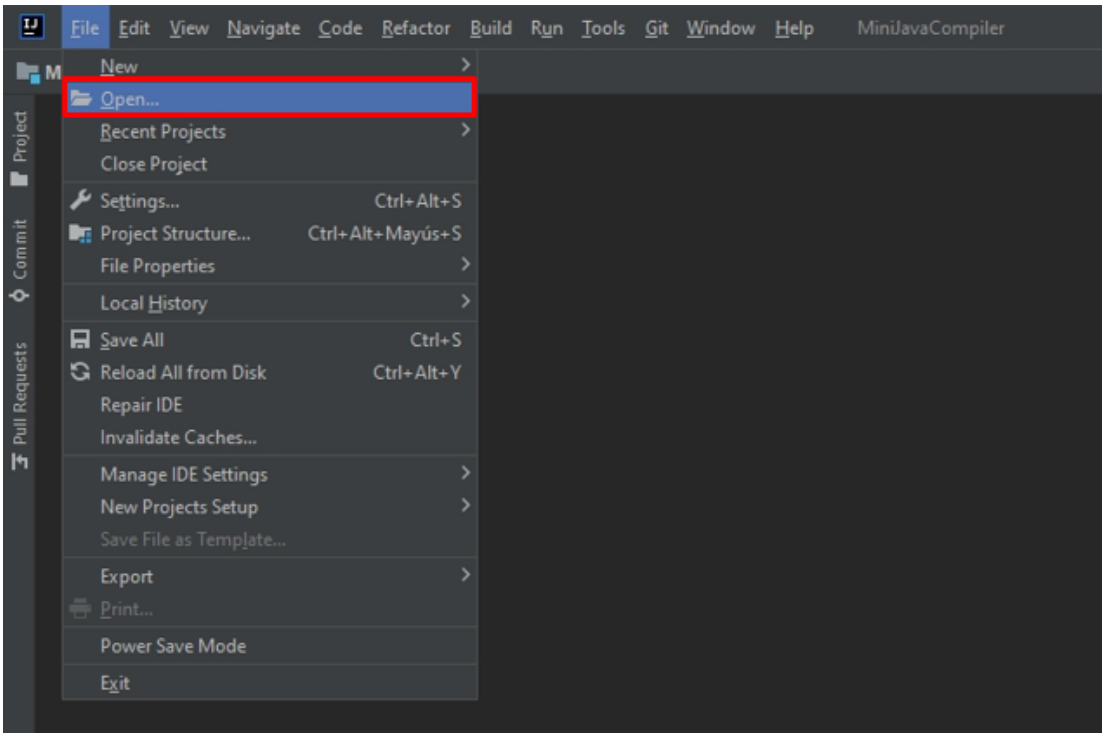



*En el .rar entregado se encuentra el Diagrama en formato .drawio para su mejor visualización. También se adjunto en formato PNG.*

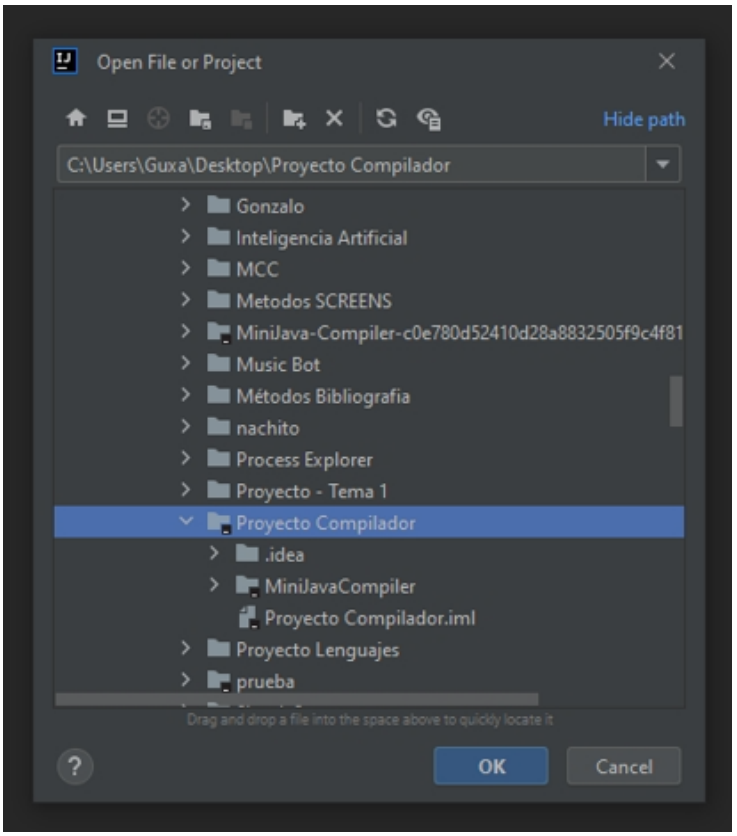
***Razuc Gonzalo***

# Compilación del Proyecto

Abrir Intellij Idea, dirigirse a “Open...”

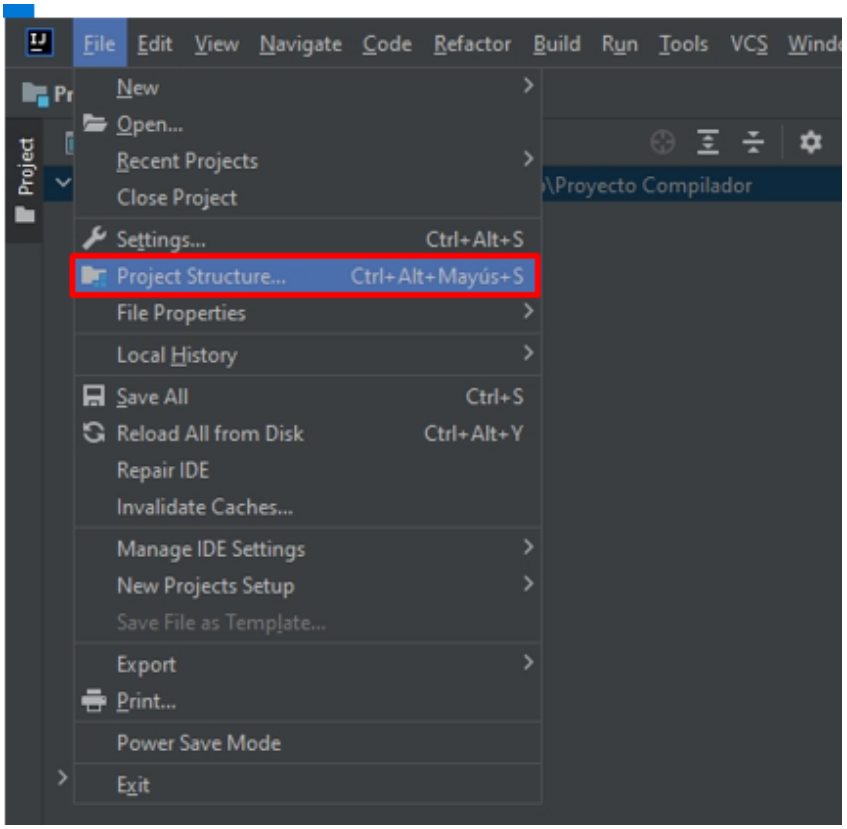



Elegir la carpeta base del proyecto

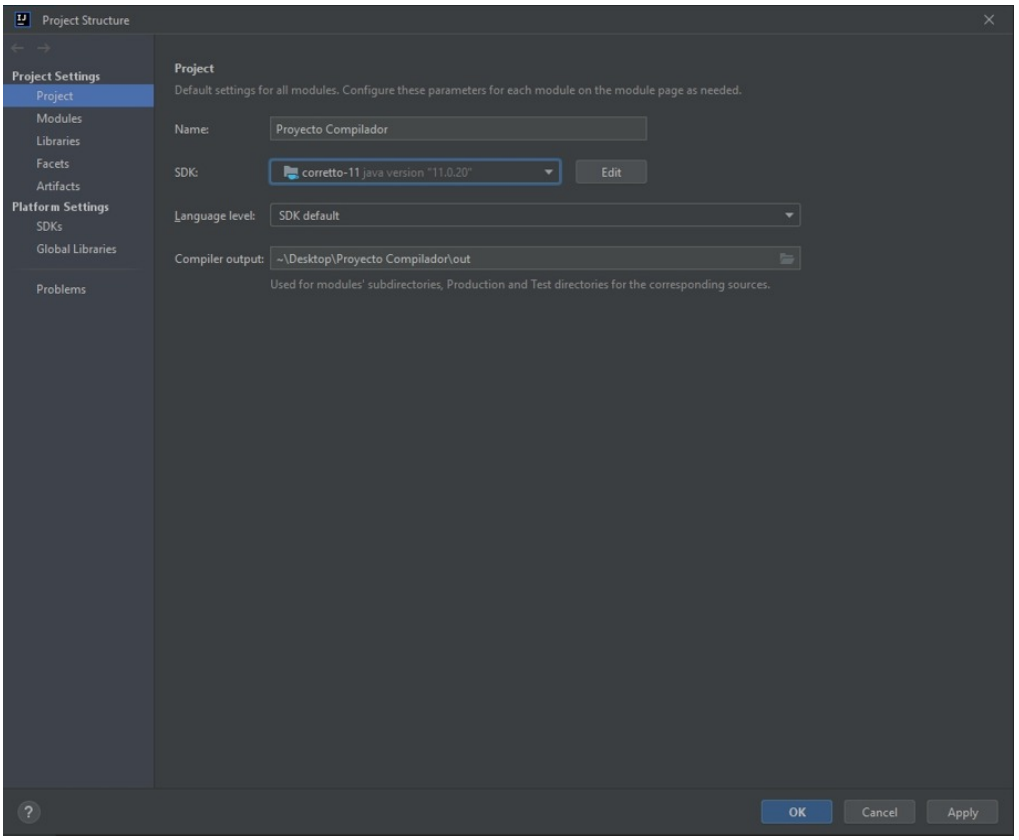



***Razuc Gonzalo***

Una vez abierto el Proyecto, verificar que el proyecto se encuentre configurado correctamente, entrando a “Project Structure...”

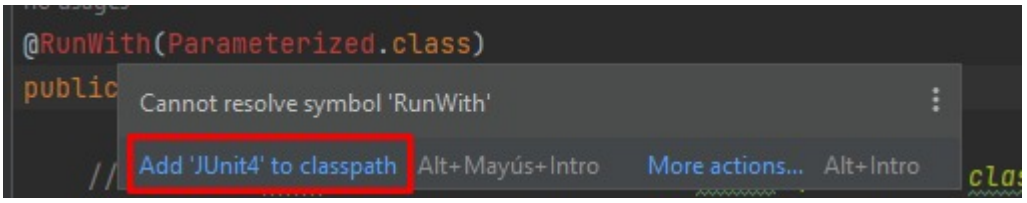



Debería verse de la siguiente manera

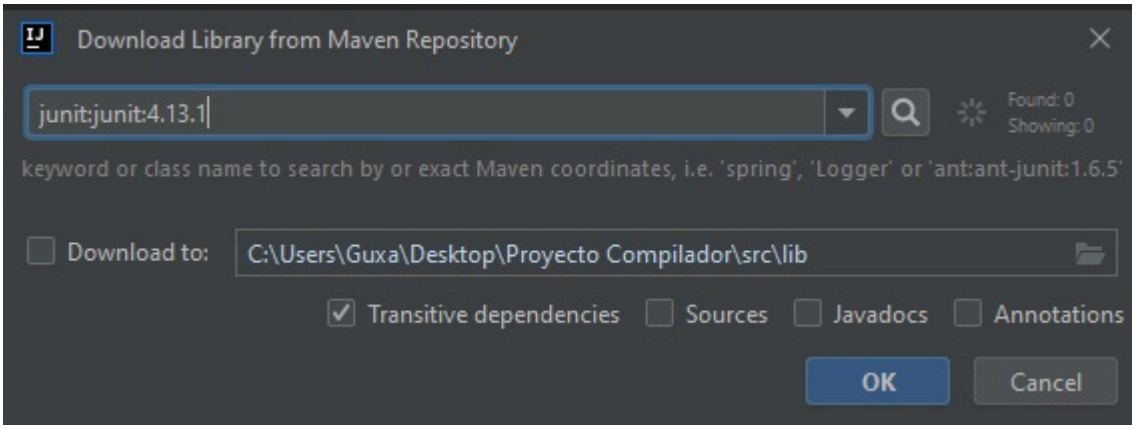



***Razuc Gonzalo***

Luego, en caso de querer realizar testeos en Compilación, se debe agregar JUnit4 al Classpath.

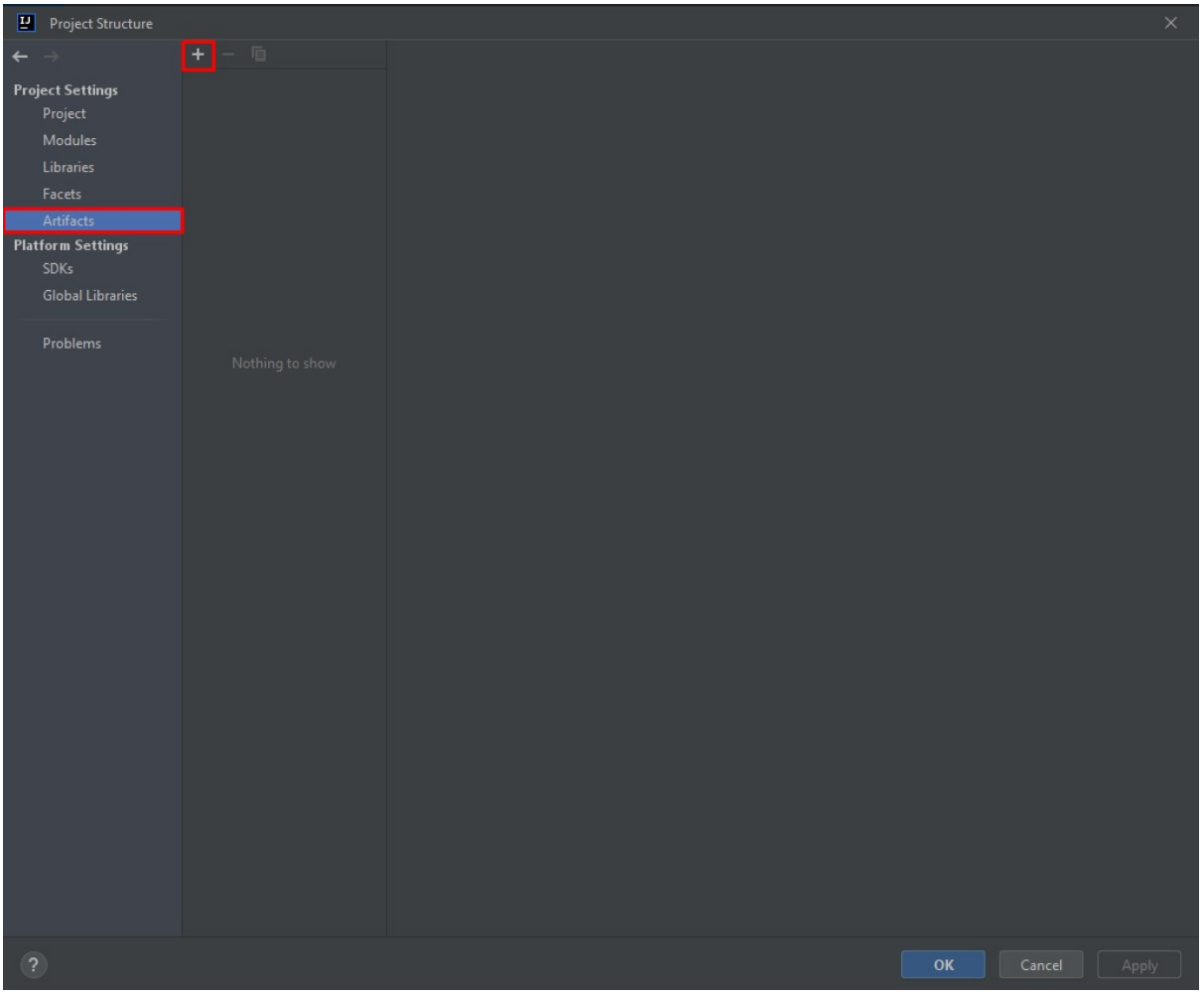



Y apretar “OK”

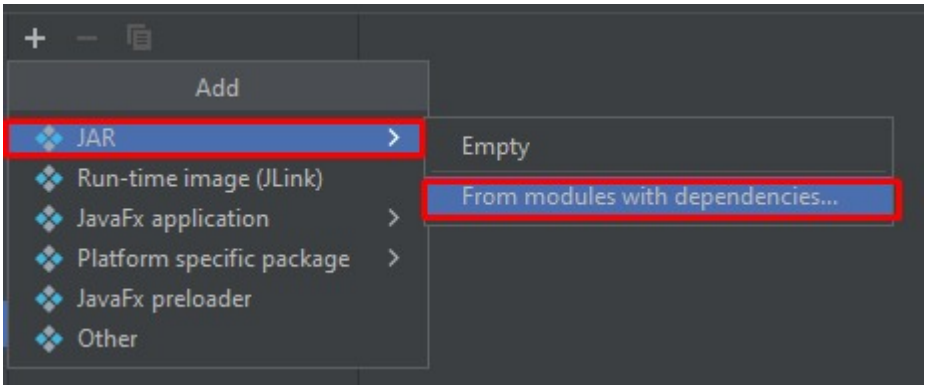


# Generación del Jar

Para generar el archivo .jar, primero debe dirigirse a “Project Structure...” como indicado anteriormente
Una vez allí debemos dirigirnos a “Artifacts” y seleccionar el signo “+” como está indicado en la siguiente foto.

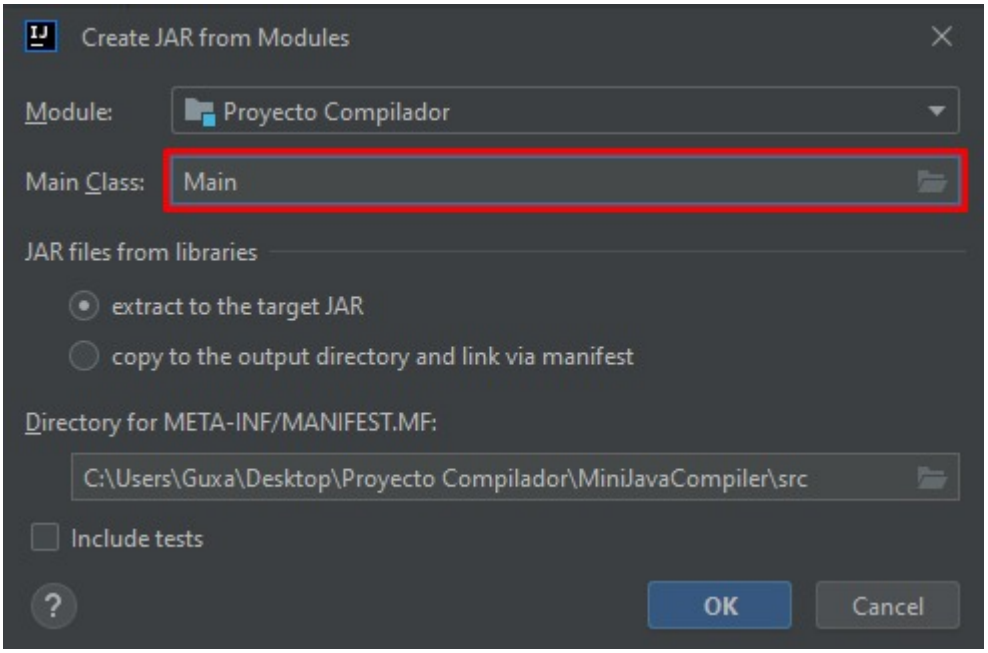



Una vez apretado el “+” debemos hacer lo siguiente

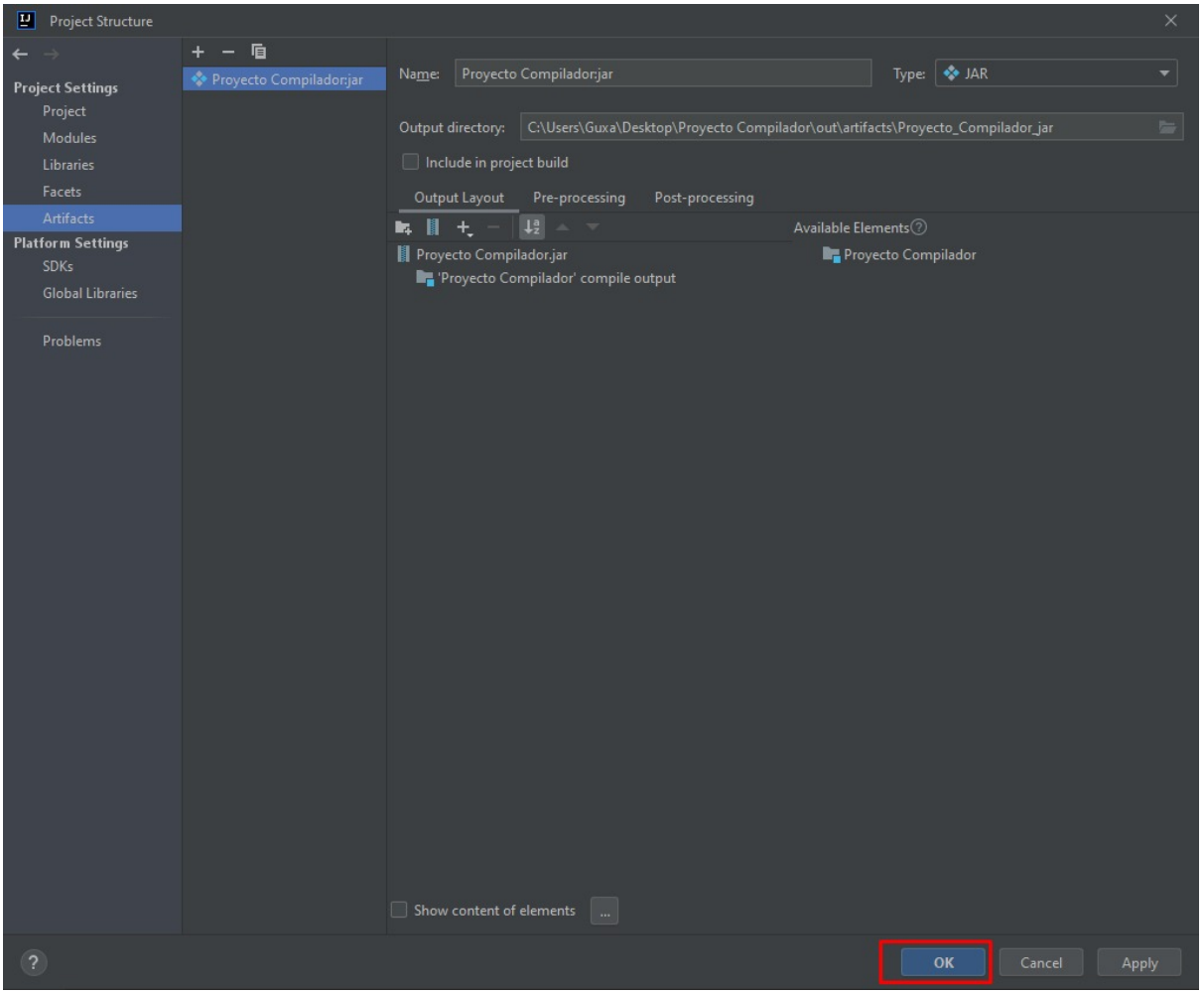



***Razuc Gonzalo***

Seleccionar la clase “Main”, y apretar “OK”.

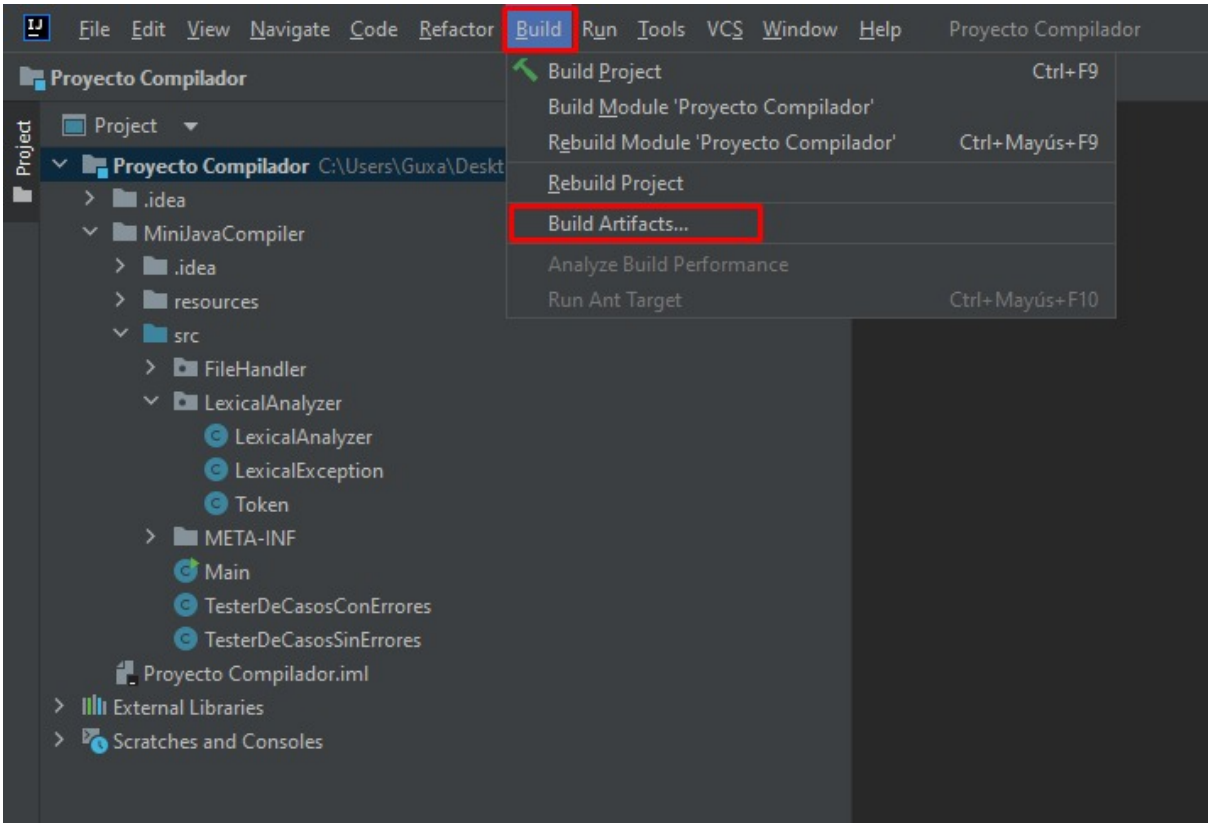



Una vez hecho esto, la pantalla debería quedar así, en la cual debemos apretar OK nuevamente.

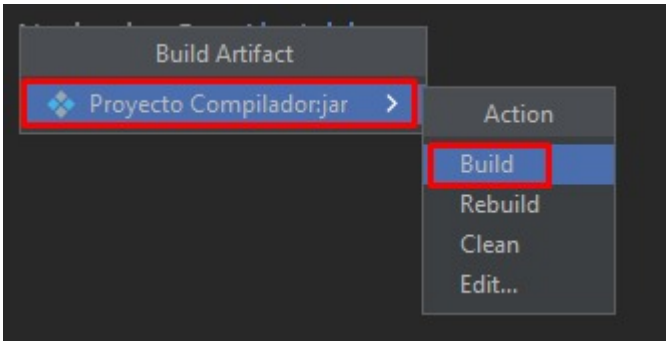



***Razuc Gonzalo***

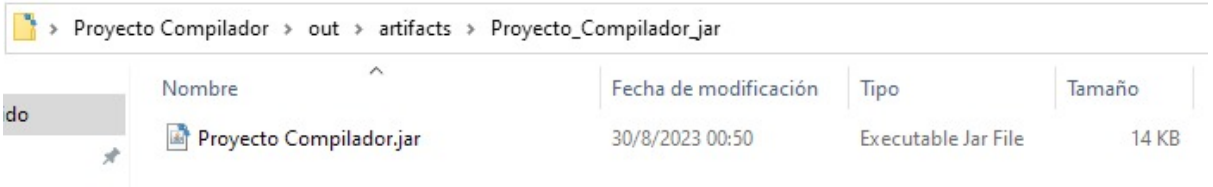
Luego, debemos ir a “Build” y seleccionar “Build Artifacts...”




Una vez hecho esto, realizar los siguientes pasos




Si se siguieron los pasos correctamente, encontraremos el .jar creado en la carpeta base del Proyecto




***Razuc Gonzalo***

## Cómo leer archivos utilizando el .jar

Una vez estemos en el directorio donde se encuentra el .jar, debemos ejecutar el siguiente comando por consola

```
java -jar Compilador.jar programa1.java
```

*Donde:*
**Compilador.jar** debe ser modificado al nombre del .jar creado.
**programa1.java** debe ser modificado al archivo que se desea analizar léxicamente.

## Logros que se intentan alcanzar en esta etapa:

- Imbatibilidad Semántica II