INFORME
ETAPA 5
GENERACIÓN DE CÓDIGO
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.

Generación de código en cada nodo del AST

Nodos Sentencia:

Nodo asignación:
Se genera el código para el lado derecho de la asignación y luego para el lado izquierdo de la misma.

Nodo bloque:
Se invoca a cada sentencia del bloque a que genere su código. Al finalizar, se liberan n cantidad de lugares en la memoria, siendo n el total de variables locales declaradas en el bloque.

Nodo llamada:
Se genera el código del acceso en cuestión y luego, en caso de que el método tenga retorno y no sea utilizado, se genera una instrucción POP para descartar ese valor.

Nodo if:
Se genera el código de la condición asociada a la sentencia if, y:
- En caso de que el nodo no tenga una sentencia else, se genera la instrucción BF, donde, si la condición del if (que va a estar en el tope de la pila) resulta falsa, se saltará al final del if. Luego de lo mencionado, se genera el código de la sentencia asociada al if.
- En caso contrario, se genera la instrucción BF para evaluar el tope de la pila (valor obtenido al generar el código de la condición del if), donde, si esta resulta falsa, se realizará el salto al comienzo del else. Luego, se genera el código de la sentencia asociada al if, la instrucción JUMP que realizará el salto al final del if, y por último se genera el código de la sentencia else.

Nodo declaración de variable:
- Se genera el código de la expresión asociada a la asignación y luego, generando una instrucción STORE n, siendo n el offset asociado a la variable que está siendo declarada, se almacena el valor de la variable.
- Por último, se incrementa el total de variables locales del bloque asociado a la declaración de la variable.

Razuc Gonzalo

Nodo return:
- En caso de que el metodo asociado al nodo return deba almacenar el resultado se generara ese código y se utilizara la instrucción STORE n, siendo n la locación reservada para el retorno del método.
- Se eliminan las variables locales reservadas utilizando FMEM
- Se actualiza el frame pointer para que apunte al registro de activación del llamador, generando la instrucción STOREFP.
- Por último, se genera la instrucción RET n, donde n será el total de lugares en la pila a liberar (dependiendo del alcance del método) para actualizar el program counter.

- En caso de que el método no tenga un tipo de retorno, sólo se generarán las instrucciones STOREFP y RET n.

Nodo while:
- Se genera el código para una etiqueta para el inicio de la sentencia y para la condición.
- Se genera la instrucción BF label, donde se evaluará si el tope de la pila (valor obtenido de la condición de la sentencia) es falso. En caso de serlo, el program counter saltará al comienzo de label, siendo label la etiqueta en donde termina el while.
- Se genera el código de la sentencia asociada al nodo while.
- Se genera el código para ahora saltar al comienzo del while.
- Por último, se genera la etiqueta del fin de la sentencia while.

Nodos Expresión:

Nodo expresión binaria:
Se genera primero el código para el lado izquierdo de la expresión y luego para el lado derecho. Por último, se genera el código del operador de la expresión, donde, la instrucción que se generará dependerá del tipo de operador del que se trate.

Nodo expresión unaria:
Se genera el código del operando asociado a la expresión y luego, en caso de que el del operador de la expresión sea “!”, se generará la instrucción NOT. En caso de que el operador sea “-”, se generará la instrucción NEG.

Nodos Literales:

Nodo booleano: en caso de que el lexema del token asociado al nodo sea true, se generará la instrucción PUSH 1, en caso contrario, es decir, cuando el lexema del token sea false, se generará la instrucción PUSH 0.
Nodo caracter: se generará la instrucción PUSH lexeme, siendo lexeme el caracter asociado al nodo.

Nodo entero: se generará la instrucción PUSH lexeme, siendo lexeme el entero asociado al nodo.
Nodo nulo: se generará la instrucción PUSH 0.
Nodo String: los Strings son alojados en la sección de datos ".data" de la máquina virtual.
-Se generará una etiqueta, donde, mediante la directiva DW, se aloja el lexema asociado al token del nodo.
- Por último, en la sección de ".code", se generará la instrucción PUSH label, siendo label la etiqueta definida para el String.

Nodos Acceso:

Acceso Constructor:

- Se genera la instrucción RMEM 1 para reservar memoria para el resultado del malloc (referencia al nuevo CIR), se genera la instrucción para poner en el tope de la pila el tamaño del CIR (será el parámetro del malloc), se carga en el tope de la pila la etiqueta para llamar a la rutina malloc y, se genera la instrucción CALL para realizar la llamada.
- Se genera la instrucción DUP para no perder la referencia al CIR cuando se le quiera asociar la virtual table a través de la instrucción STOREREF.
- Se apila la dirección del comienzo de la virtual table que se le va a asociar al CIR creado generando la instrucción PUSH label, siendo label el nombre de la etiqueta de la virtual table.
- Se guarda la referencia a la virtual table en el CIR, generando la instrucción STOREREF 0, se genera el código de los parámetros y, por último, se apila, generando la instrucción PUSH, la dirección del comienzo de la generación de código del constructor de la clase en cuestión y luego se realiza la llamada generando CALL.

Acceso a método:

Si el acceso es a un método con alcance estático:
- Si el método tiene un tipo de retorno, se genera la instrucción RMEM 1, para reservar un lugar para el valor de retorno.
- Si el método accedido tiene parámetros, se genera el código de cada expresión de cada uno de sus parámetros.
- Se apila la dirección de la etiqueta label asociada al comienzo de la generación de código del método en cuestión, generando la instrucción PUSH label.
- Se realiza la llamada, generando la instrucción CALL.

Si el acceso es a un método con alcance dinámico:
- Se genera la instrucción LOAD 3 para cargar el this.
- Si el método tiene un tipo de retorno, se genera la instrucción RMEM 1, para reservar un lugar para el valor de retorno, y, la instrucción SWAP que, intercambia el tope de la pila con el tope -1, para que el this quede en el tope.
- Análogamente al acceso de un método con alcance estático, si el método en cuestión tiene parámetros, se genera el código de cada expresión de cada uno de sus parámetros, y, luego de cada generación de cada parámetro, se intercambia el tope de la pila con el tope -1, generando la instrucción SWAP, para que el this quede en el tope de la pila.
- Se duplica el this, generando la instrucción DUP, ya que, al cargar la virtual table con la instrucción LOADREF 0, se pierde la referencia al this.
- Se carga la virtual table, generando la instrucción LOADREF 0.
- Se carga la dirección del método en cuestión, generando la instrucción LOADREF n, siendo n el offset asociado al método.
- Se realiza la llamada, generando la instrucción CALL.
- Si el acceso tiene un encadenado, se genera el código del encadenado.

Nodo acceso this:
Se apila la referencia al this, generando la instrucción LOAD 3 y, si el nodo tiene un encadenado, se genera el código para este.

Nodo acceso variable:
Se genera el código dependiendo si la variable es una variable local/parámetro o un atributo.

Si la variable es estática:
- Se genera la instrucción PUSH con el label asociada a la misma.
- En caso de no ser un lado izquierdo o tener encadenado se genera la instrucción LOADREF 0 para cargar la virtual table.
- En caso contrario se genera la instrucción SWAP para que el this quede en el tope de la pila y luego STORE REF 0 para cargar la virtual table.

Si la variable es un atributo (variable de instancia):
- Se apila la referencia al objeto (CIR) de donde se buscará la variable, generando la instrucción LOAD 3.
- En caso de que el atributo tenga un encadenado o no sea el lado izquierdo de una asignación, se apila el valor del atributo, generando la instrucción LOADREF n, siendo n el offset asociado al atributo en cuestión dentro del CIR.
- En caso contrario, es decir, si el atributo se accedió como un lado derecho de una asignación, o bien, este tiene un encadenado, se

genera la instrucción SWAP para bajar la referencia al this, y, por último, se almacena el valor del tope de la pila en el atributo, generando la instrucción STOREREF n, siendo n el offset asociado al atributo dentro del CIR.

Si la variable es local o es un parámetro:
- En caso de que la variable/parámetro no tenga un encadenado o no sea el lado izquierdo de una asignación, se apila el valor de la variable/parámetro generando la instrucción LOAD n, siendo n el offset asociado a la variable/parámetro.
- En caso contrario, se almacena el valor del tope de la pila en la variable, generando la instrucción STORE n, siendo n el offset asociado a la variable/parámetro.
- Por último, si la variable/parámetro o atributo tienen un encadenado, se genera el código para este.

Nodo expresión parentizada:
Se genera el código de la expresión y, en caso de tener un encadenado, se genera el código de este.

Nodos encadenado

Nodo variable encadenada:
- Si la variable no es el lado izquierdo de una asignación, o tiene un encadenado, se apila el valor de la variable en la pila, generando la instrucción LOADREF n, siendo n el offset asociado a la variable dentro CIR.
- En caso contrario, se genera la instrucción SWAP para intercambiar el tope de la pila con el tope -1 y así tener el this en el tope para poder almacenar el valor en el objeto en cuestión. Luego, se almacena el valor del tope de la pila en la variable, generando la instrucción LOADREF n, siendo n el offset asociado de la variable en el CIR, desapilando el valor y la referencia al objeto.
- Por último, en caso de que la variable tenga un encadenado, se genera su código.

Nodo llamada encadenada:
Si el método llamado tiene alcance estático:
- Se descarta la referencia en el tope de la pila ya que no se necesita al tratarse de una llamada a un método estático, generando la instrucción POP.
- En caso de que el método en cuestión tenga un tipo de retorno, se genera la instrucción RMEM 1 para reservar lugar de retorno.
- Se generan, análogamente a la generación de código de los nodos de acceso a método estático, los parámetros del método encadenado.

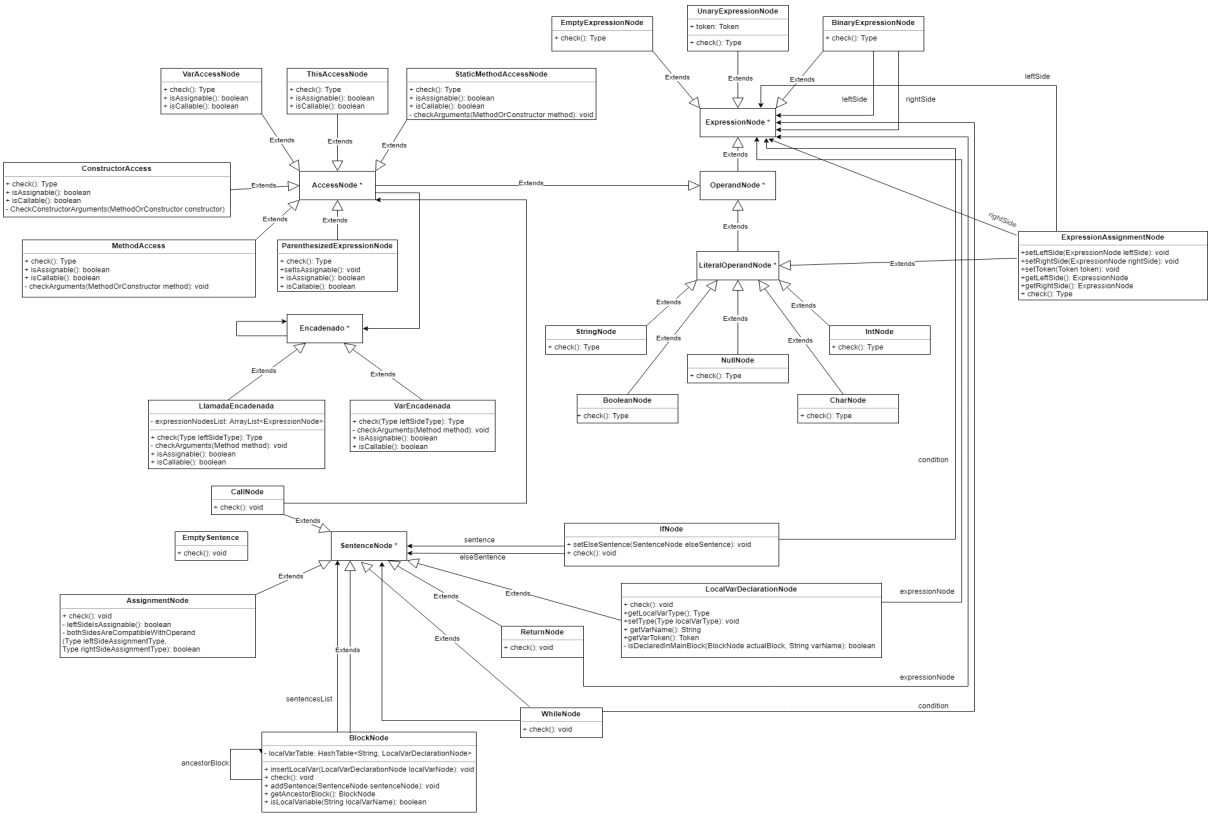
- Se genera la instrucción PUSH label, siendo label la etiqueta asociada al método.
- Se genera la instrucción CALL para ir a la dirección en donde comienza la generación de código del método.

Si el método llamado no tiene alcance estático:
- Si el método tiene un tipo de retorno, se genera la instrucción RMEM 1, para reservar un lugar para el valor de retorno, y, la instrucción SWAP que, intercambia el tope de la pila con el tope -1, para que el this quede en el tope.
- Se generan, análogamente a la generación de código de los nodos de acceso a método dinámico, los parámetros del método encadenado.

- Se duplica el this, generando la instrucción DUP, ya que, al cargar la virtual table con la instrucción LOADREF 0, se pierde la referencia al this.
- Se carga la virtual table, generando la instrucción LOADREF 0.
- Se carga la dirección del método en cuestión, generando LOADREF n, siendo n el offset asociado al método.
- Se realiza la llamada, generando la instrucción CALL.

Generación de código de métodos y constructores
- Se genera una instrucción para establecer la etiqueta de la unidad.
- Se guarda el enlace dinámico del registro de activación del llamador, generando la instrucción LOADFP.
- Se carga en el tope de la pila el valor del stack pointer, de manera que se apile el lugar en donde comienza el registro de activación de la unidad llamada, generando la instrucción LOADSP.
- Se actualiza el frame pointer con el valor del tope de la pila generando la instrucción STOREFP.
- En el caso de ser un Metodo, se genera el offset de sus parametros.
- Se genera el código de su bloque principal, tanto para el constructor como para el metodo.
- Se actualiza el frame pointer con el valor del tope de la pila generando la instrucción STOREFP.
- Se retorna de la unidad liberando n lugares en la pila, generando la instrucción RET n.

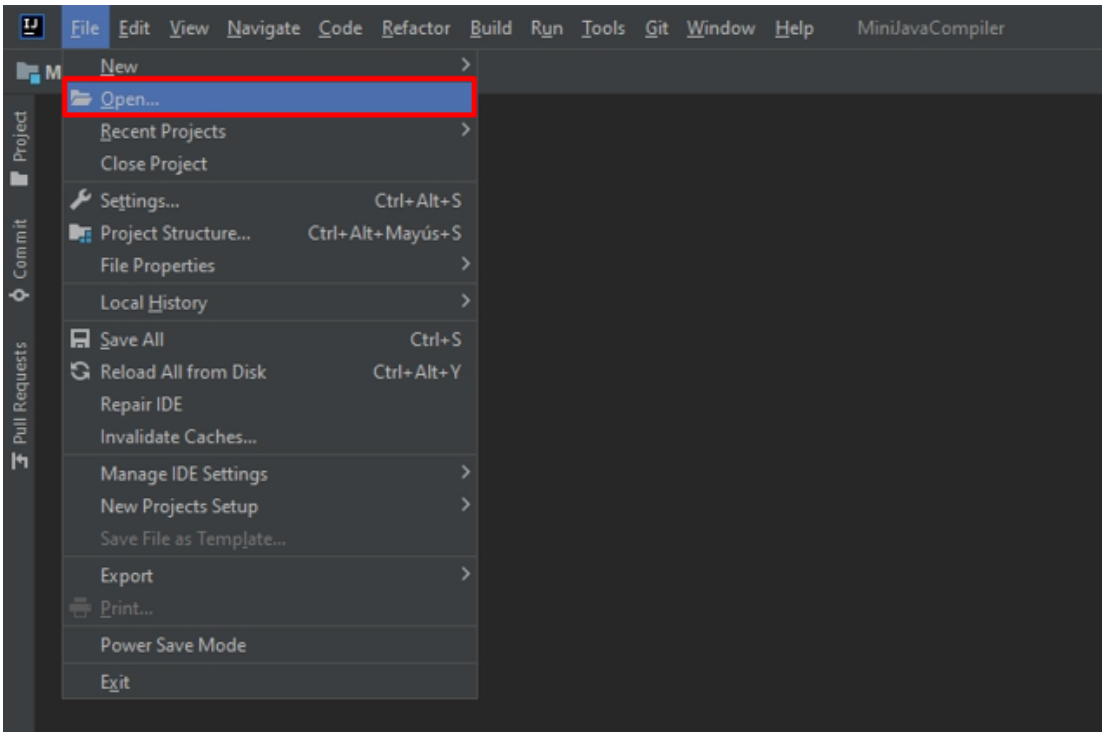
Diagrama de los AST



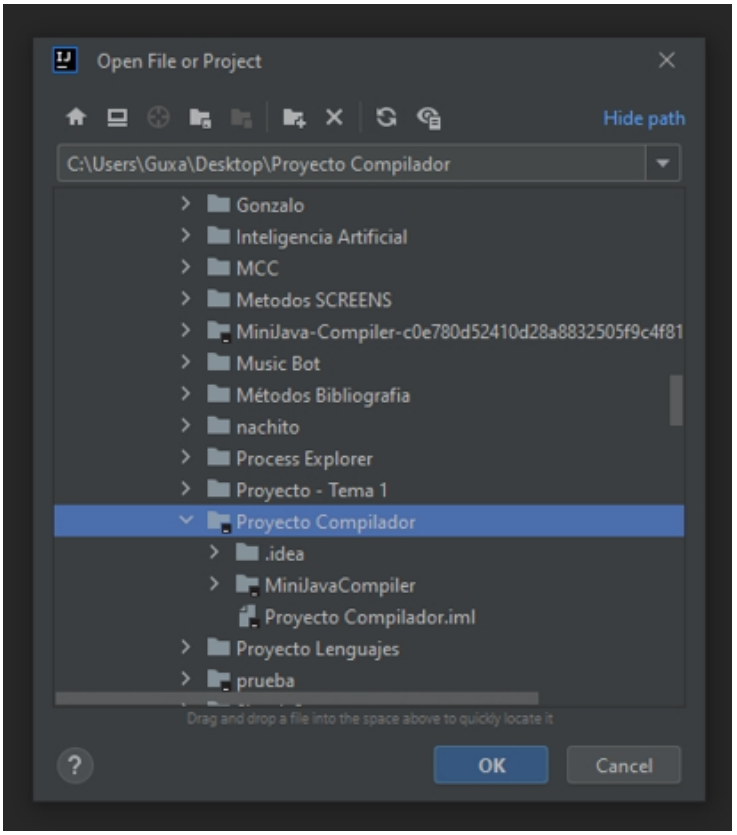
Razuc Gonzalo

Compilación del Proyecto

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

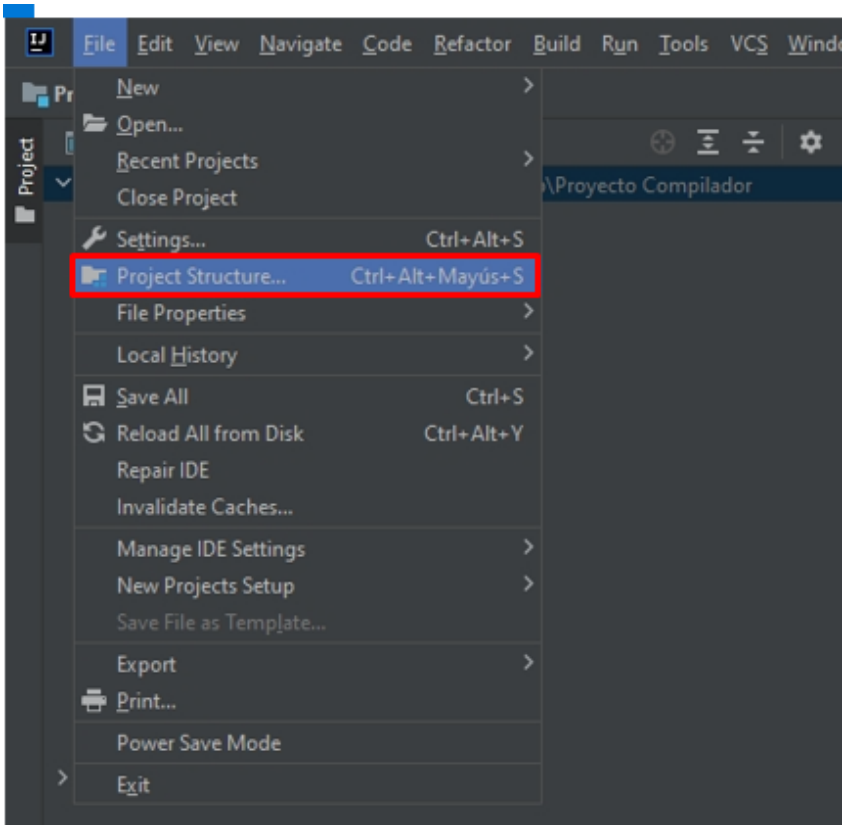


Elegir la carpeta base del proyecto

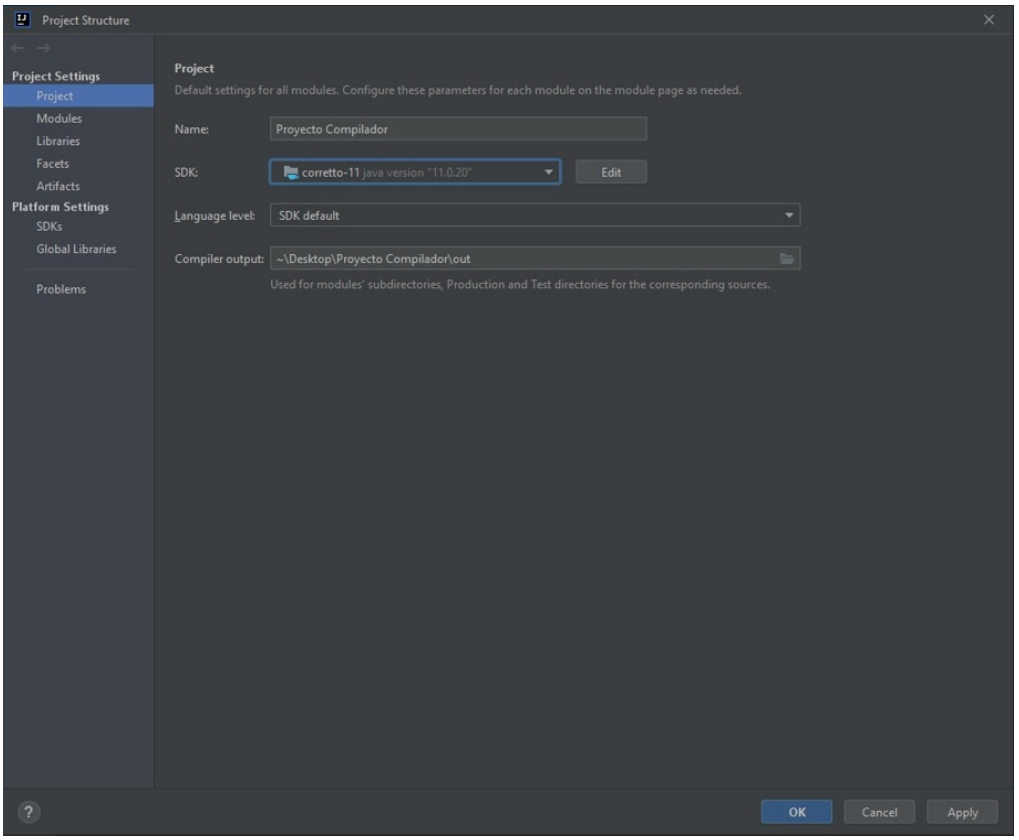


Razuc Gonzalo

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

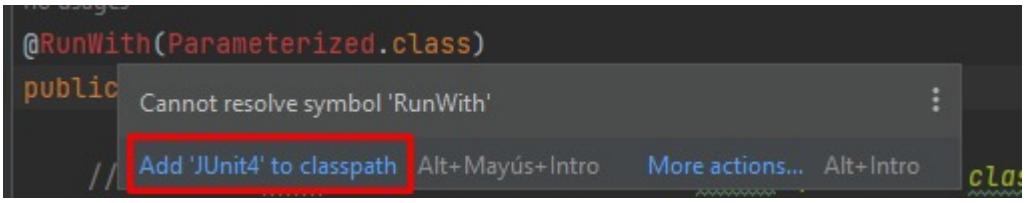


Debería verse de la siguiente manera

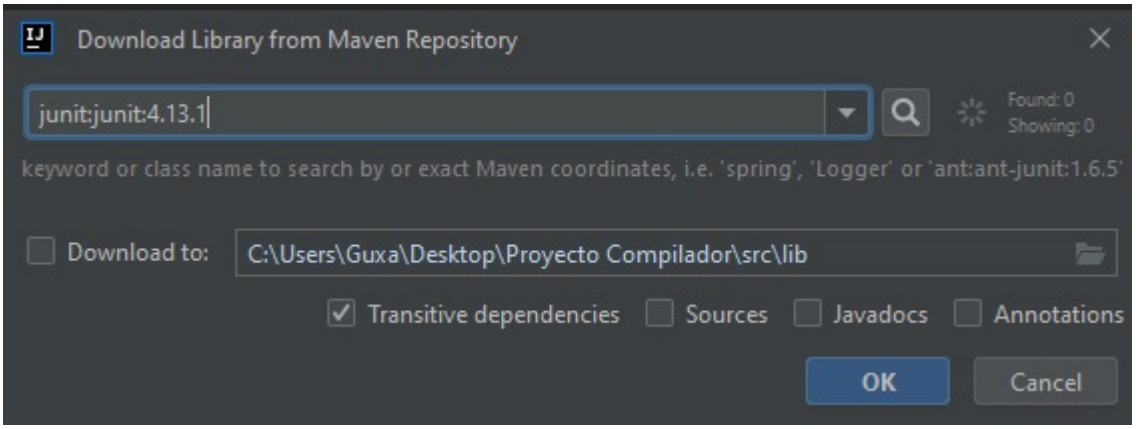


Razuc Gonzalo

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

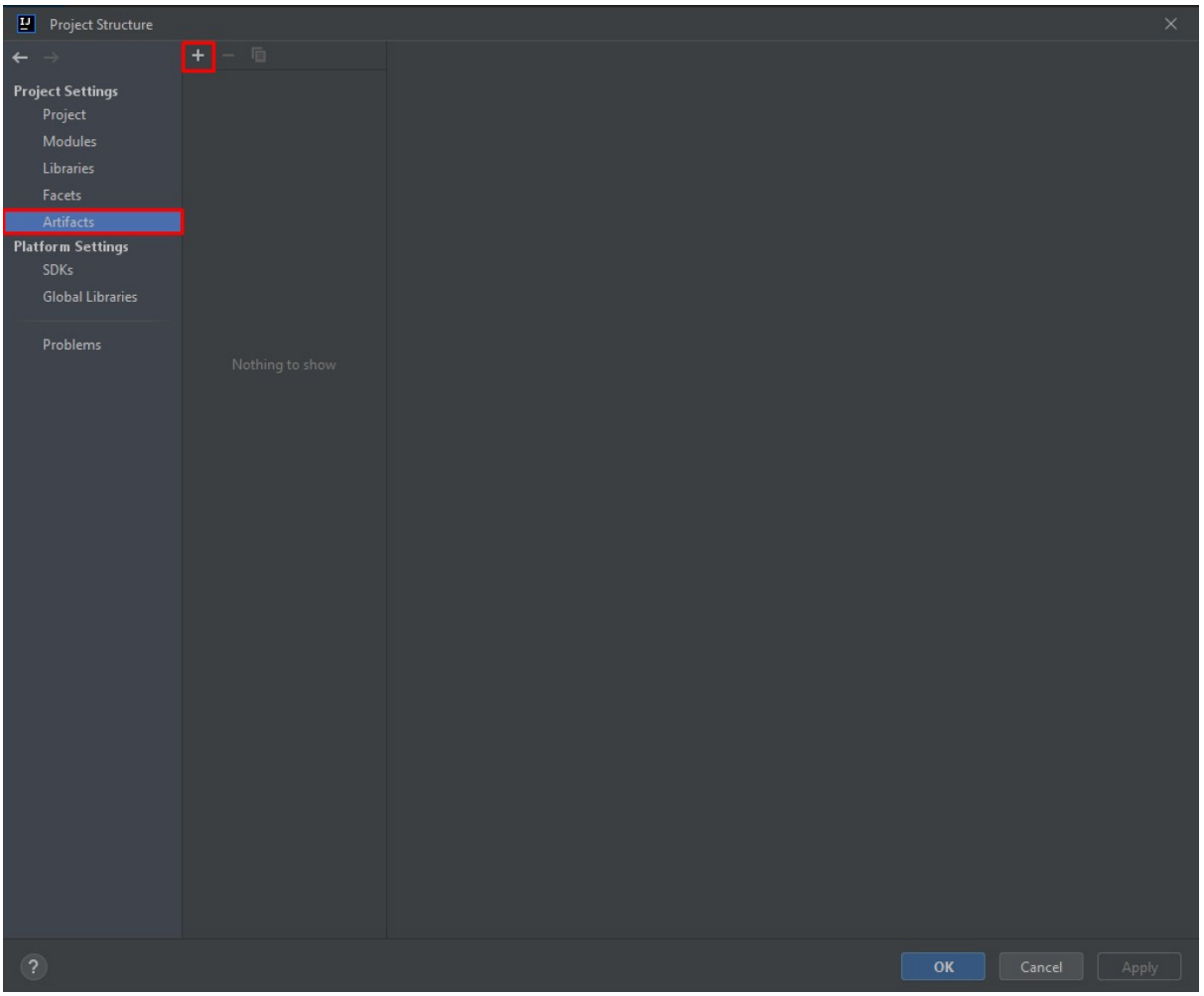


Y apretar “OK”

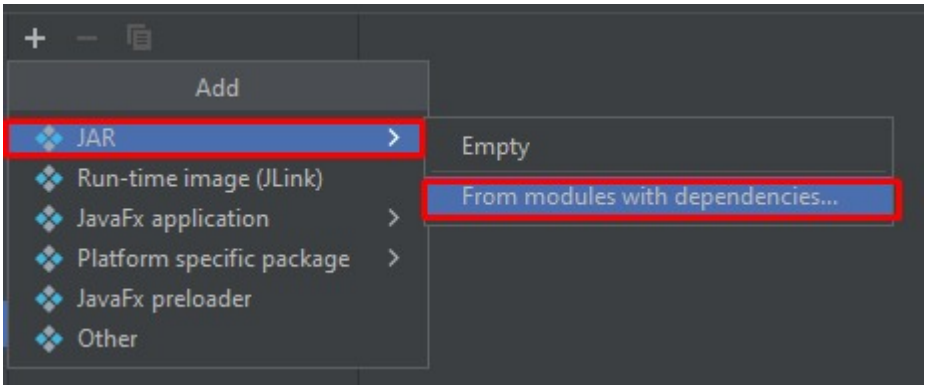


Generación del Proyecto

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.

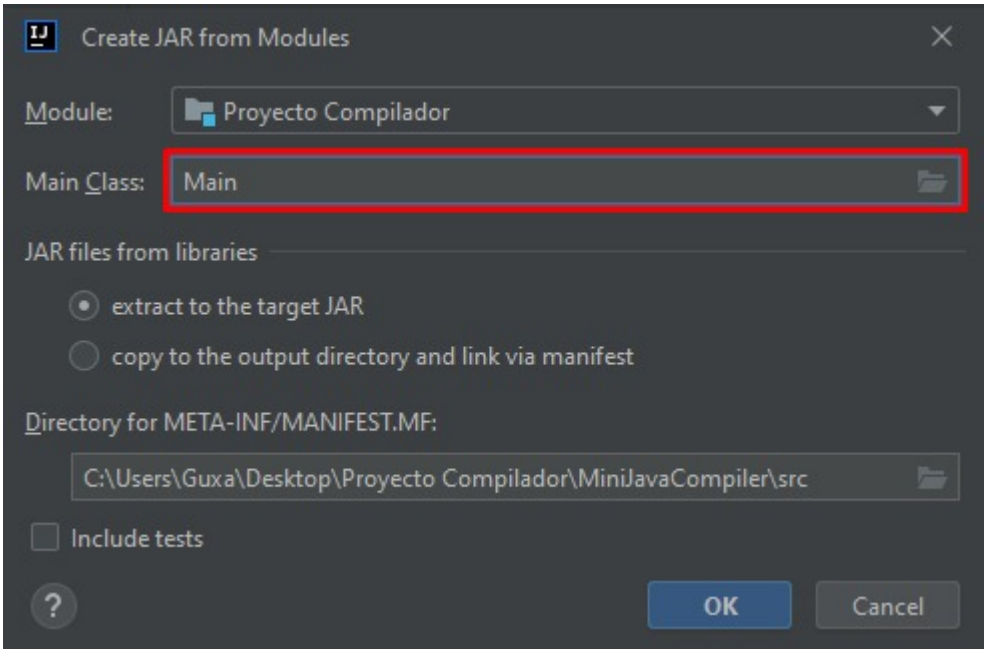


Una vez apretado el “+” debemos hacer lo siguiente

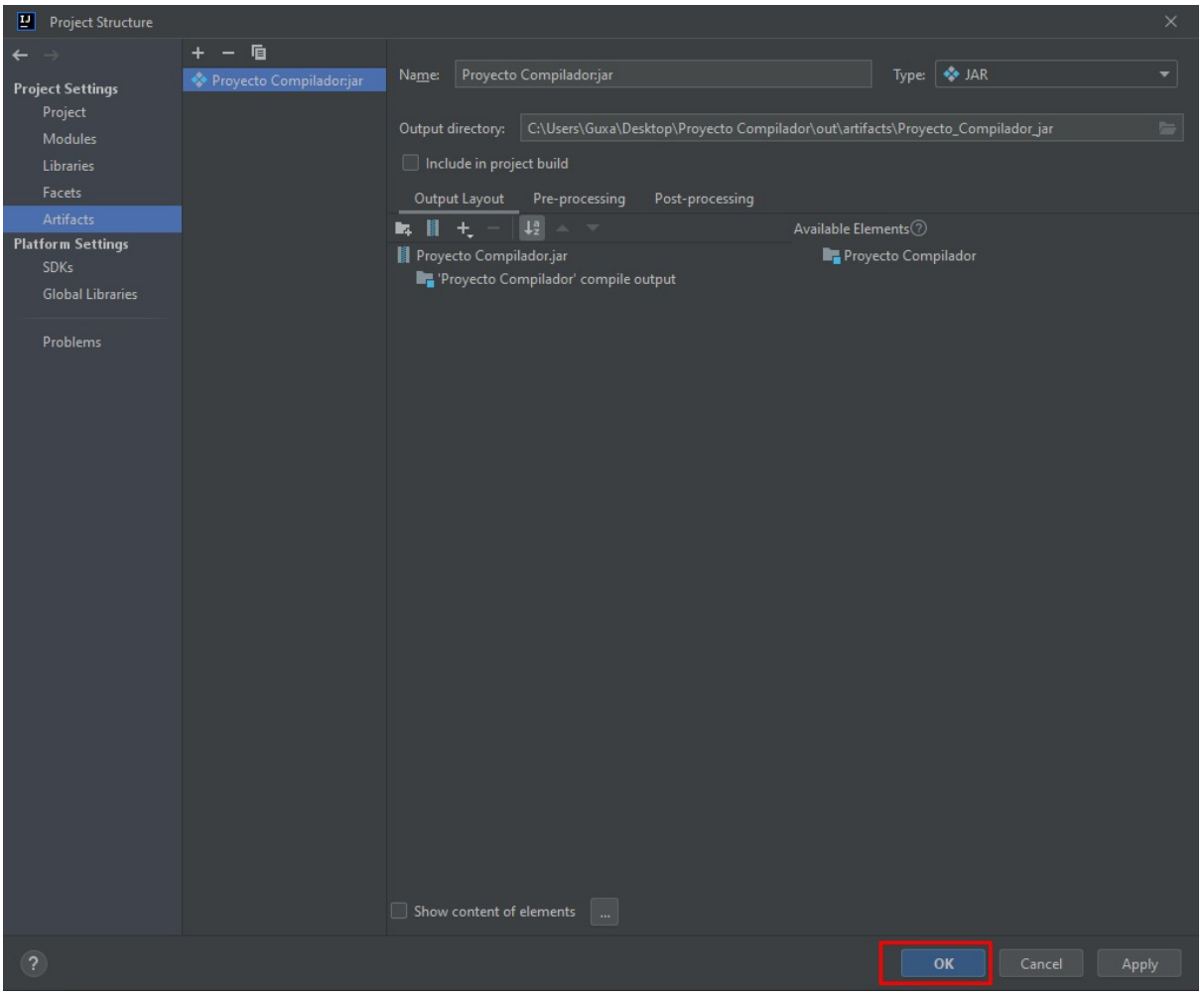


Seleccionar la clase “Main”, y apretar “OK”.

Razuc Gonzalo

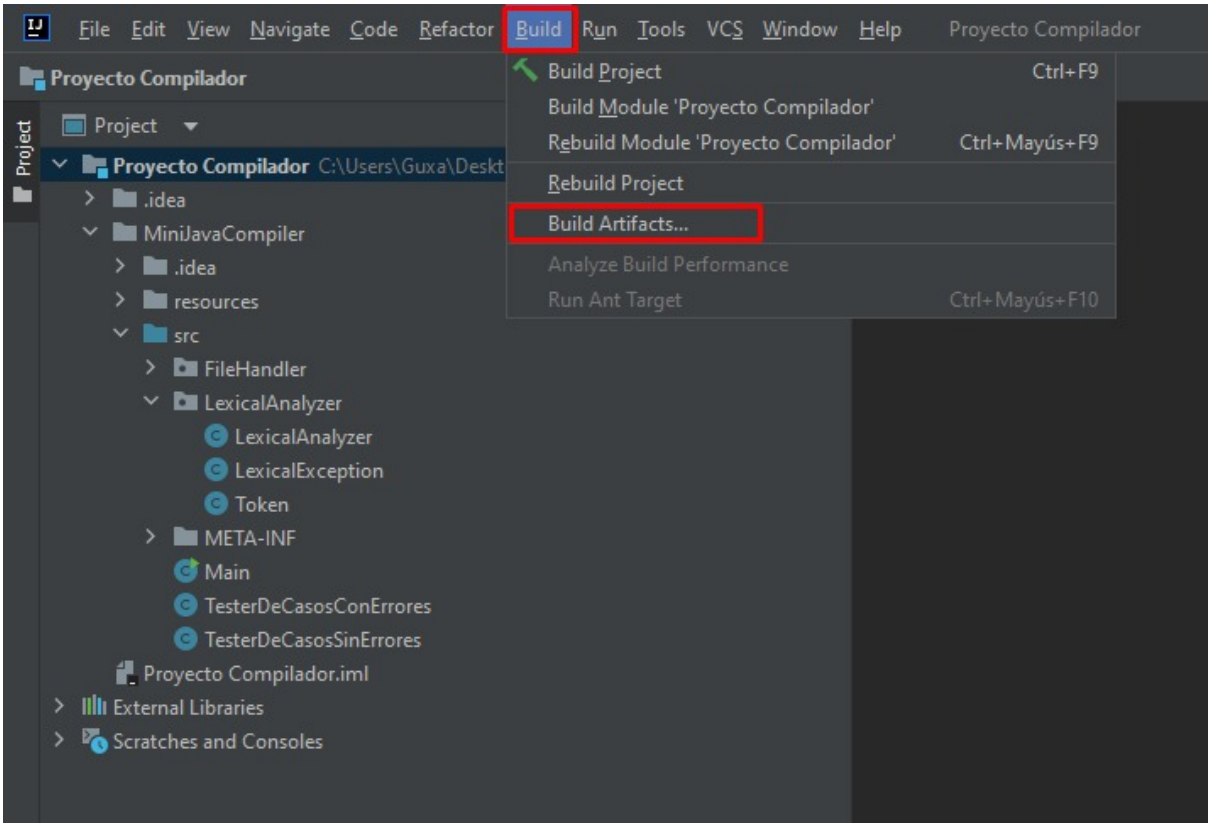


Una vez hecho esto, la pantalla debería quedar así, en la cual debemos apretar OK nuevamente.

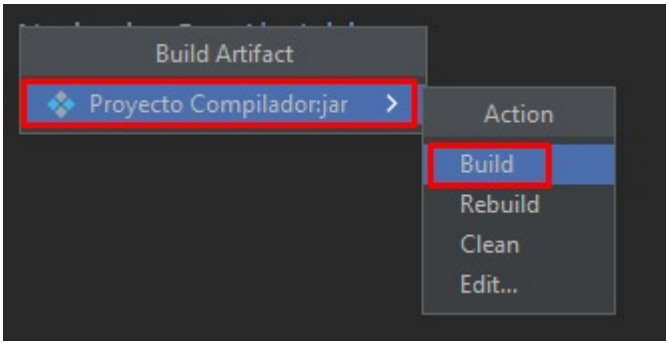


Luego, debemos ir a “Build” y seleccionar “Build Artifacts...”

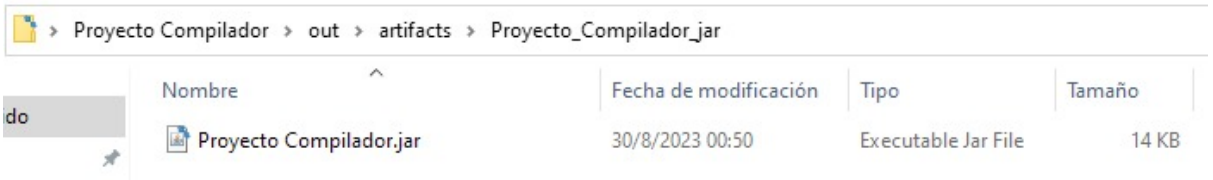
Razuc Gonzalo



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 (Generación de Código)

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

```
java -jar Compilador.jar programa1.java salida.txt
```

Donde:
Compilador.jar debe ser modificado al nombre del .jar creado.
programa1.java debe ser modificado al archivo que se desea analizar.
salida.txt será el output (*se puede utilizar cualquier extensión*)

Logros que se intentan alcanzar en esta etapa:

- Imbatibilidad generación
- Variables Locales Clásicas Sintáctico

Cabe destacar que se entregaron dos zips, uno llamado "*Compilador Etapa 2 - Variables Locales.zip*" y el otro "*MiniJavaCompiler.zip*".
Para probar el compilador con generación de código, se debe descomprimir el zip "*MiniJavaCompiler.zip*" y **realizar los pasos descritos anteriormente**.

En cambio, para probar las Variables Locales Clásicas, se debe descomprimir el zip "*Compilador Etapa 2 - Variables Locales.zip*" y en el proceso de compilación del proyecto y generación del jar **se debe utilizar el Main y los archivos de ese zip/proyecto.**
Además, para leer archivos utilizando el jar, se debe hacer de la siguiente manera:

Cómo leer archivos utilizando el .jar (Variables Locales Clásicas)

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.

Como verán, se utiliza exactamente igual que en las 4 etapas anteriores.