INFORME
ETAPA 1
ANALIZADOR LÉXICO
COMPILADORES E INTÉRPRETES

Razuc, Gonzalo

Tokens reconocidos por el Analizador Léxico y Expresiones Regulares.

Siendo (X) el conjunto de todos los caracteres posibles reconocidos por MiniJava, exceptuando el carácter de salto de línea.
Siendo (Y) el conjunto de todos los caracteres posibles reconocidos por MiniJava, inclusive el /n.

Conjuntos útiles:
C1 = **X - { \ }**
C2 = **X - { \ , ‘ }**
C3 = **X - { \ , “ }**

Identificador de Clase: **[A..Z] ([a..z] | [A..Z] | [0..9] | _)***
Identificador de Métodos y Variables: **[a..z] ([a..z] | [A..Z] | [0..9] | _)***
Comentario multilínea: **/* (Y)* */**
Comentario simple: **// (X)* /n**
Literal String: **“(C3 | \X)*”**
Literal carácter: **‘(C2 | \X)’**
EOF: **EOF**
Paréntesis abre: **(**
Paréntesis cierra: **)**
Corchete abre: **{**
Corchete cierra: **}**
Punto y coma: **;**
Punto: **.**
Coma: **,**
Operador mayor: **>**
Operador menor: **<**
Operador negación: **!**
Operador igual igual: **==**
Operador mayor o igual: **>=**
Operador menor o igual: **<=**
Operador distinto: **!=**
Operador más: **+**
Operador menos: **-**
Operador asterisco: *****
Operador barra: **/**
Operador conjunción lógica: **&&**
Operador disyunción lógica: **||**
Operador división: **%**
Operador igual: **=**
Operador más-igual: **+=**
Operador menos-igual: **+=**
Literal entero: **[0..9]^n** donde n es un entero que varía entre 0 y 9

Razuc, Gonzalo

Palabras clave: **{class, interface, extends, implements, public, static, void, boolean, char, int, if, else, while, return, var, this, new, null, true, false}**

Errores léxicos que el Analizador Léxico puede detectar

Literales String inválidos: salto de línea o fin de archivo antes de cerrar un String.
Literales carácter inválidos: caracteres demasiado largos ('cx'), caracteres inválidos (#), caracteres sin cierre por barra invertida('\') o directamente sin cierre ('a).
Símbolos inválidos: #, [,], etc.
Literales enteros inválidos: Enteros que contengan más de nueve dígitos.
Comentarios multi-línea sin cierre.

Decisiones de diseño y explicación de clases

LexicalAnalyzer:
Recibe como argumentos un FileHandler (utilizado para manipular el archivo) y un mapeo con todas las palabras clave.
El mapeo será utilizado para chequear si un Token corresponde a un identificador o una palabra clave del lenguaje.

FileHandler:
Se encarga de leer el archivo y recuperar del mismo un carácter o cadena de caracteres.
El método nextCharacter(), lee el próximo carácter del archivo y devuelve el entero asociado al mismo.
Decisión de Diseño
En caso de leer un enter (\t), se lee el siguiente carácter, es decir, el analizador léxico **nunca** recibirá un carácter \t.

LexicalException:
Se encarga de generar un String con el detalle del error léxico que se produzca.
Esta clase es utilizada para el logro "Columnas".

Main:
Recibe como parámetro el archivo que se quiere analizar, inicializa el Mapeo y añade al mismo todas las palabras clave.
Además, inicializa el FileHandler y LexicalAnalyzer.
Esta clase le pedirá Tokens al archivo hasta que encuentre un End Of File (EOF), en caso de no encontrarse, se lanzará una excepción.

Razuc, Gonzalo

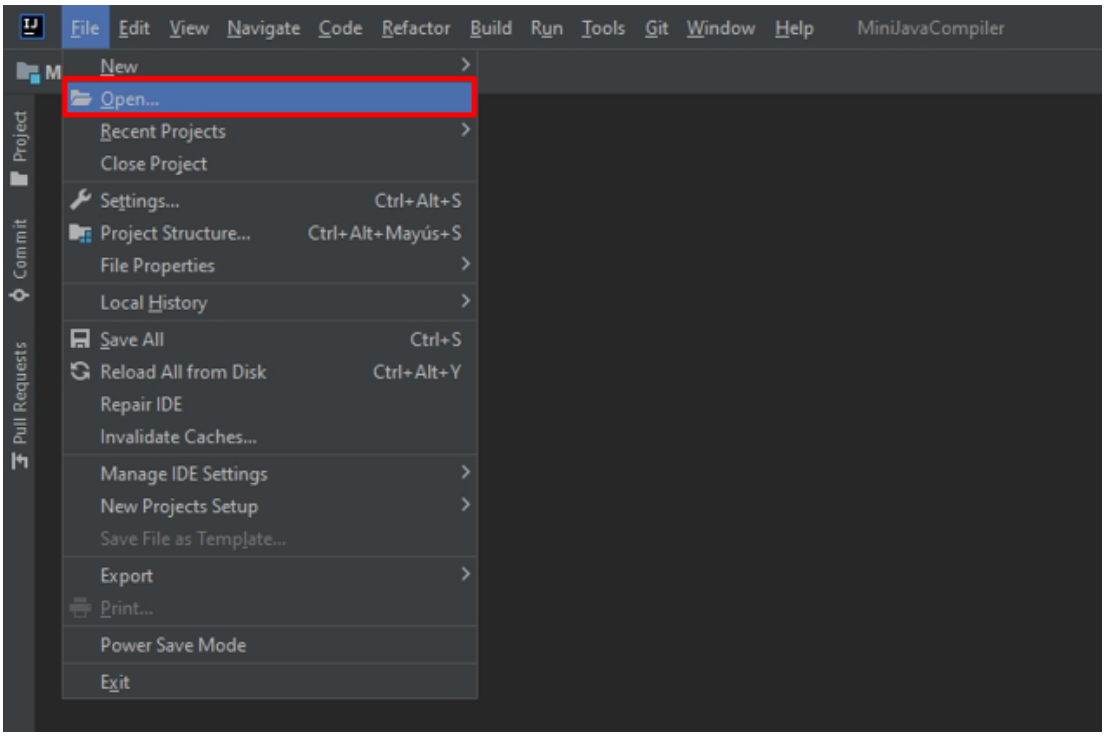
Decisión de Diseño
En algunos casos no quedaba claro que nombre colocarle a ciertos Tokens, ya que por ejemplo en el caso de un String, en las pautas aparece como "**litString**" y en el PDF llamado "***Aspectos Lexicos MiniJava 2023***" proporcionado por la cátedra aparece como "**stringLiteral**".
La decisión que se tomó es de basarse exactamente en este último archivo, por lo cual, en este caso el String quedaría determinado como "**stringLiteral**".

Logros que se intentaron alcanzar

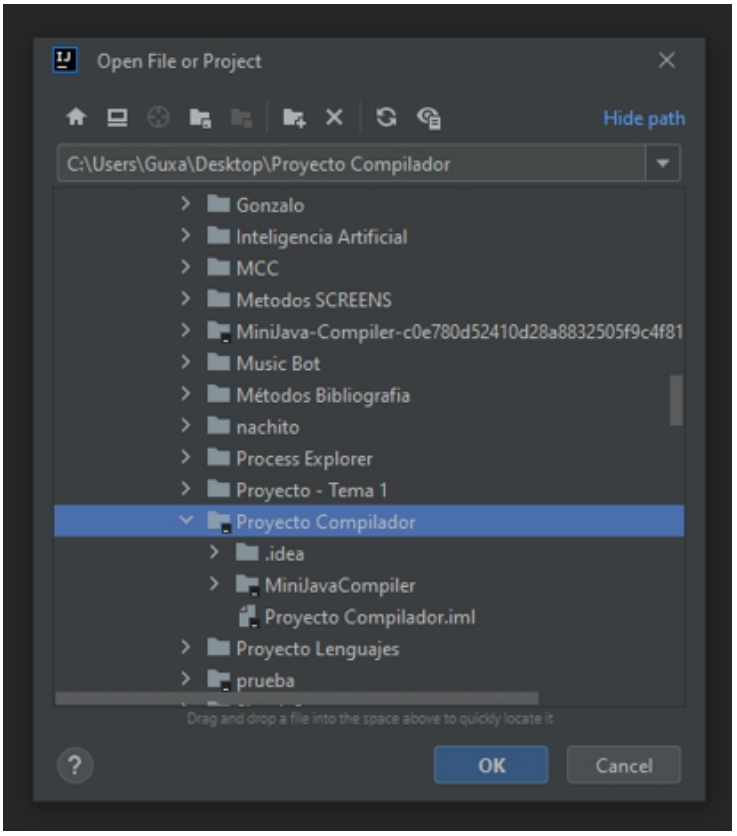
Entrega anticipada léxica
Imbatibilidad léxica
Reporte de error elegante
Columnas

Compilación del Proyecto

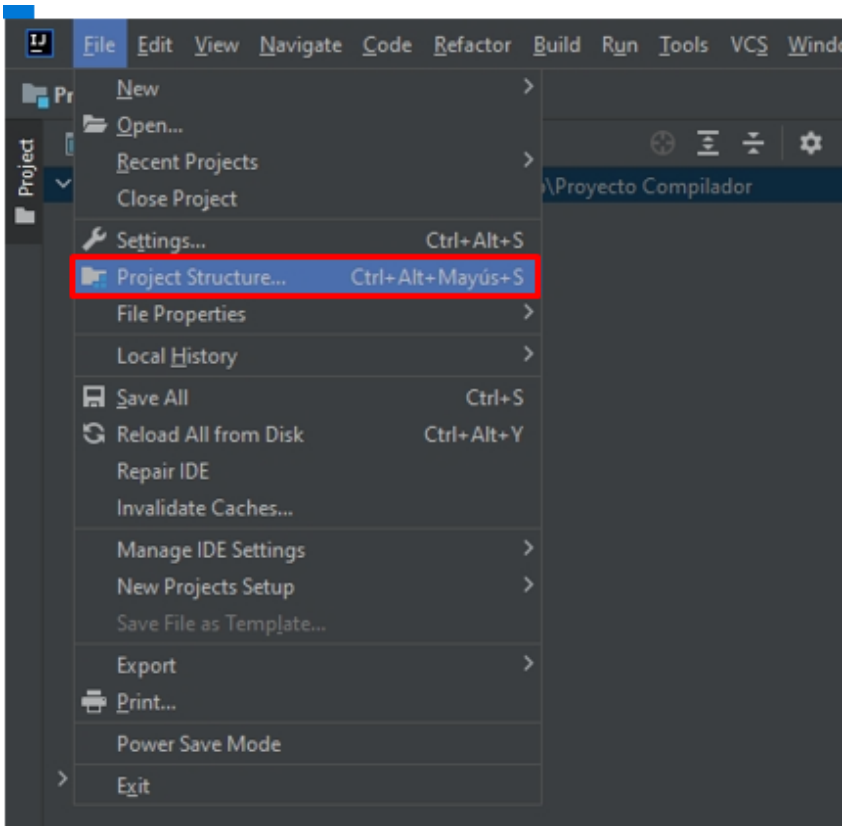
Abrir Intellij Idea, dirigirse a "Open..."



Elegir la carpeta base del proyecto

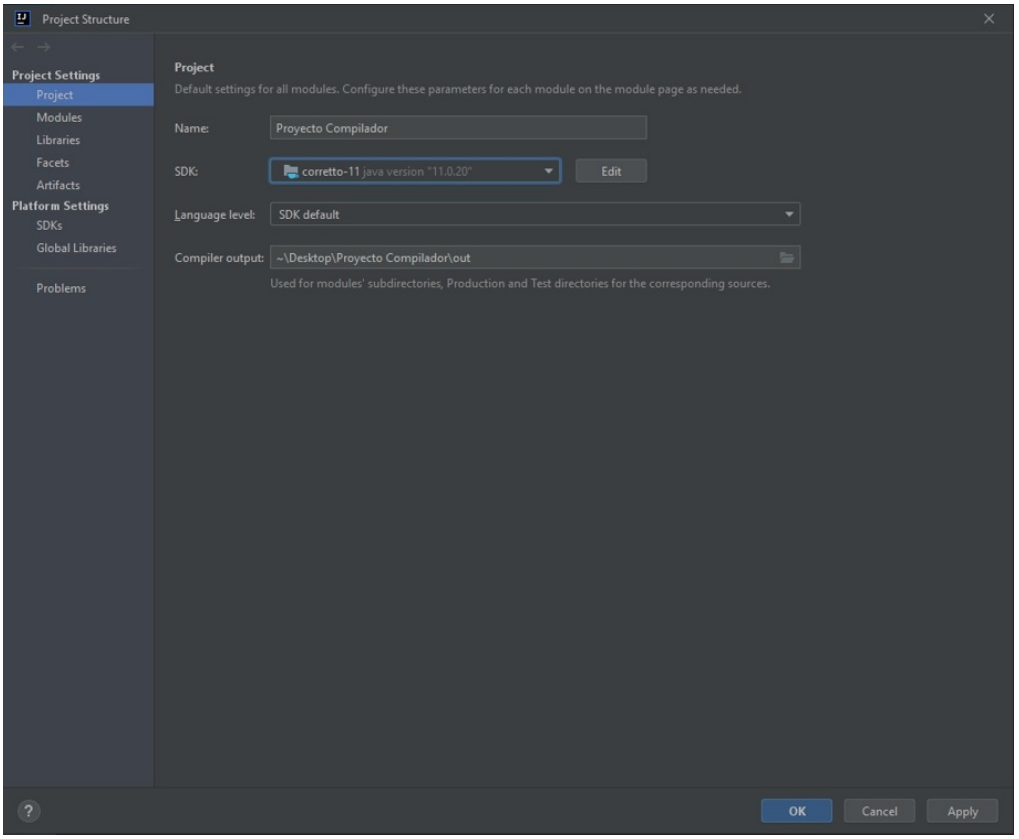


Una vez abierto el Proyecto, verificar que el proyecto se encuentre configurado correctamente, entrando a “Project Structure...”

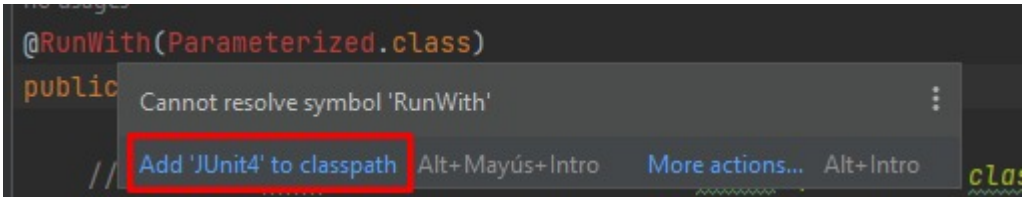


Razuc, Gonzalo

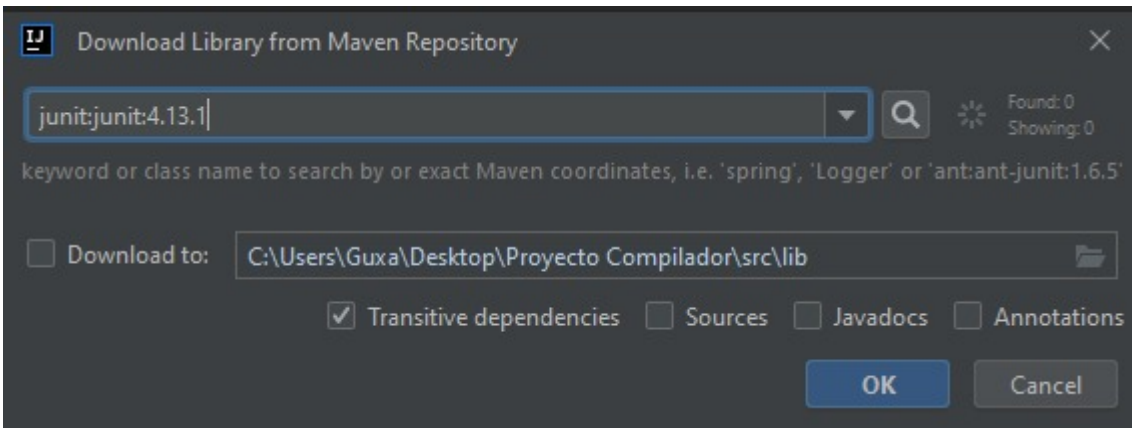
Debería verse de la siguiente manera



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



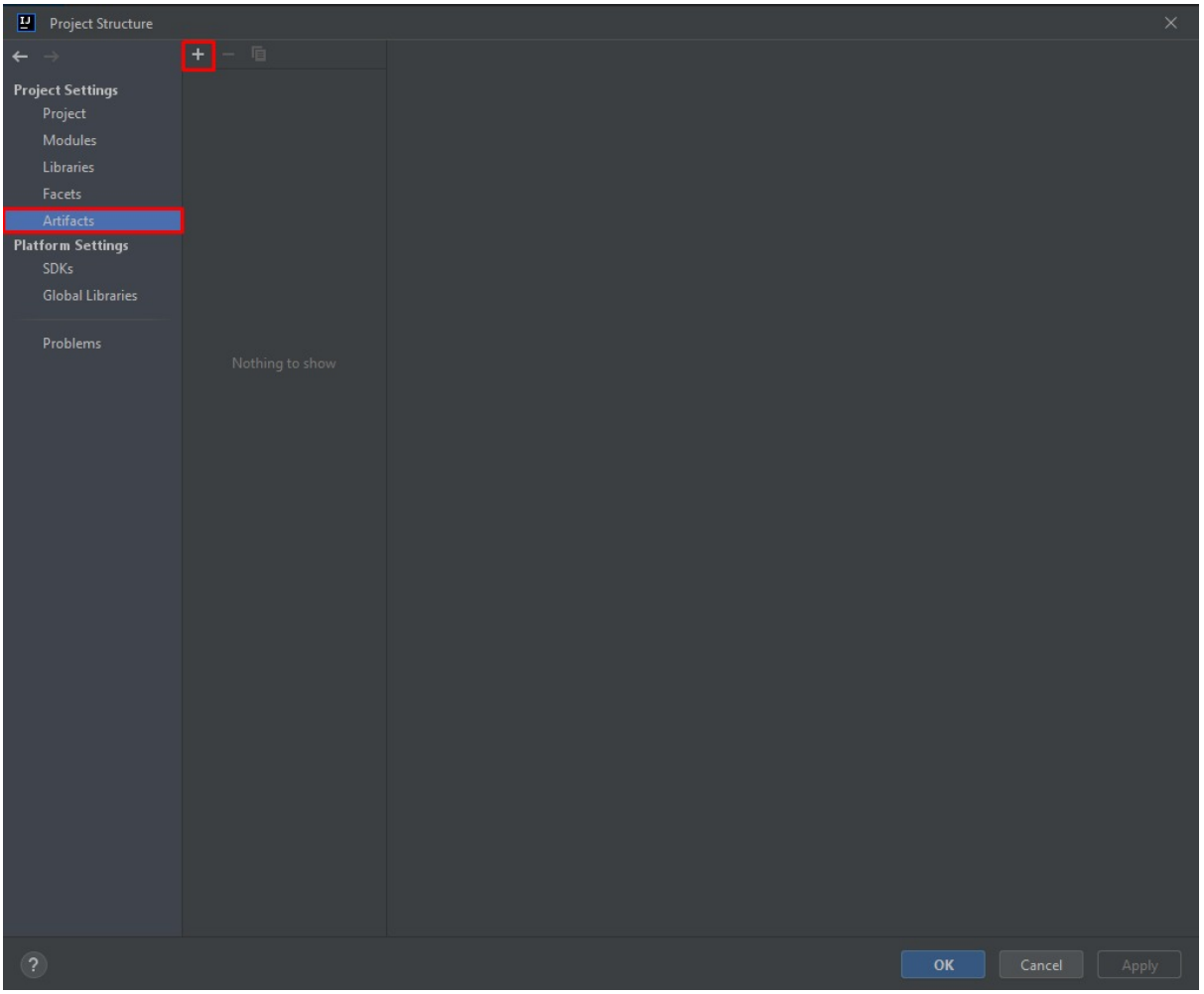
Y apretar “OK”



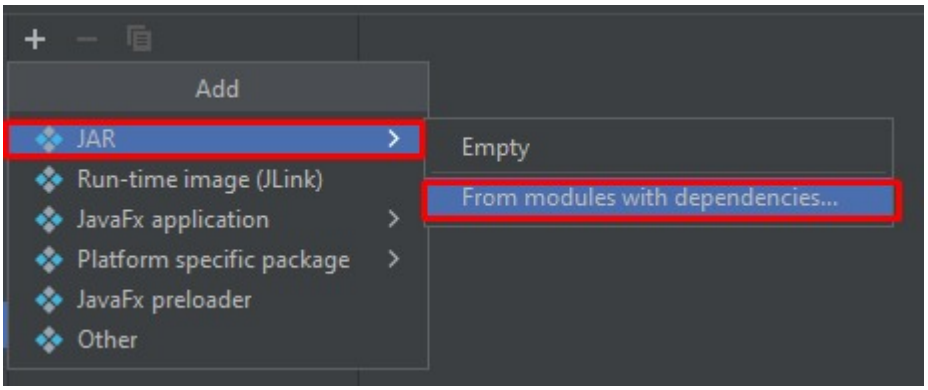
Razuc, Gonzalo

Generación del Java

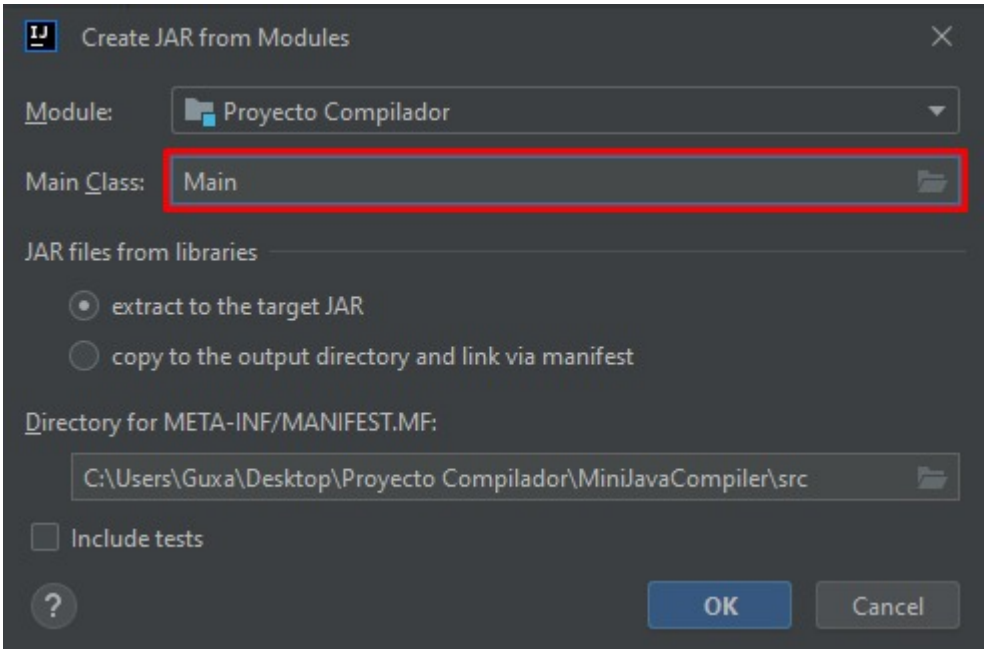
Para generar el archivo .jar, primero debe dirigirse a “Project Structure...” como indicado anteriormente
Una vez allí debemos dirigirnos a “Artifacts” y seleccionar el signo “+” como está indicado en la siguiente foto.



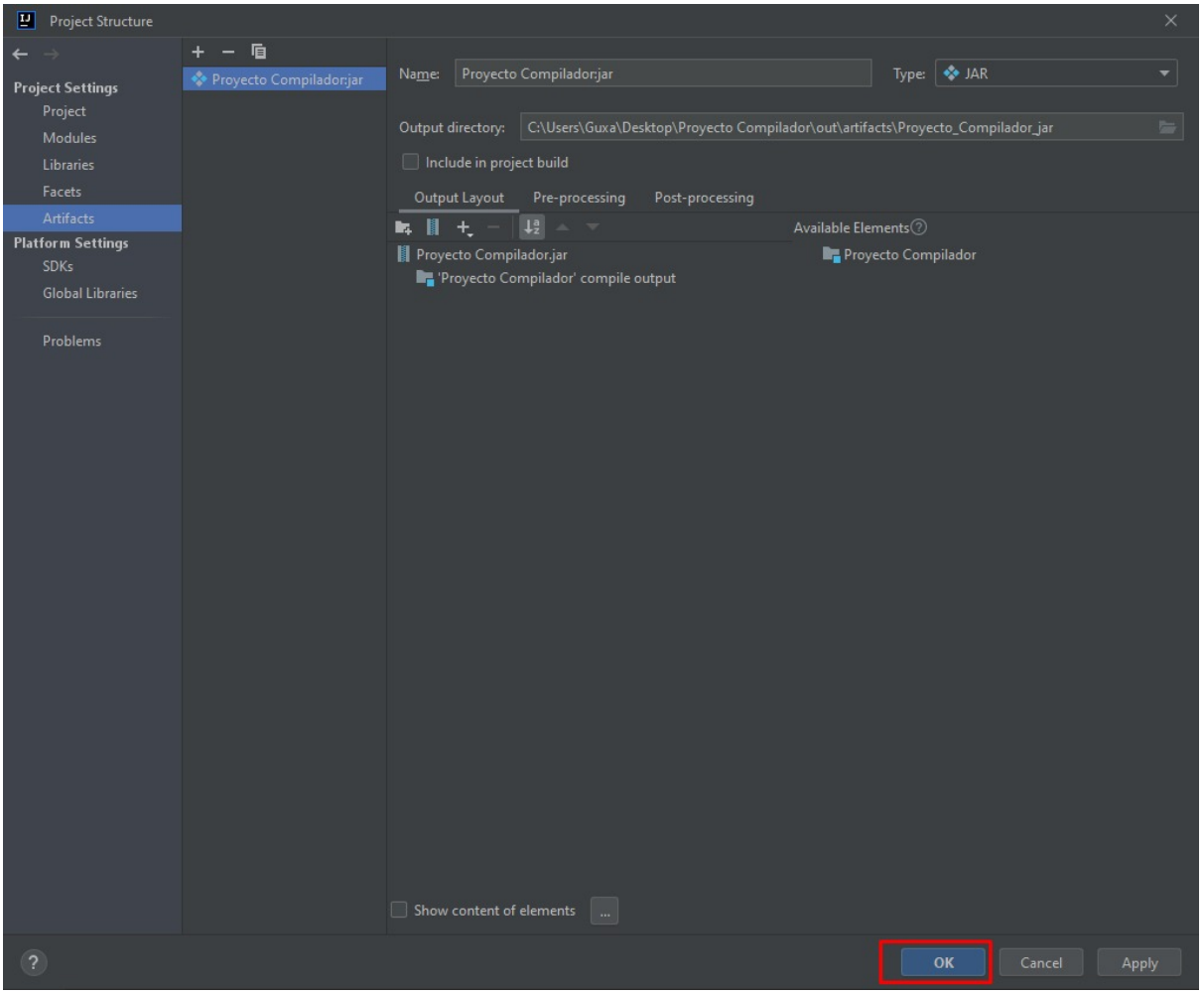
Una vez apretado el “+” debemos hacer lo siguiente



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

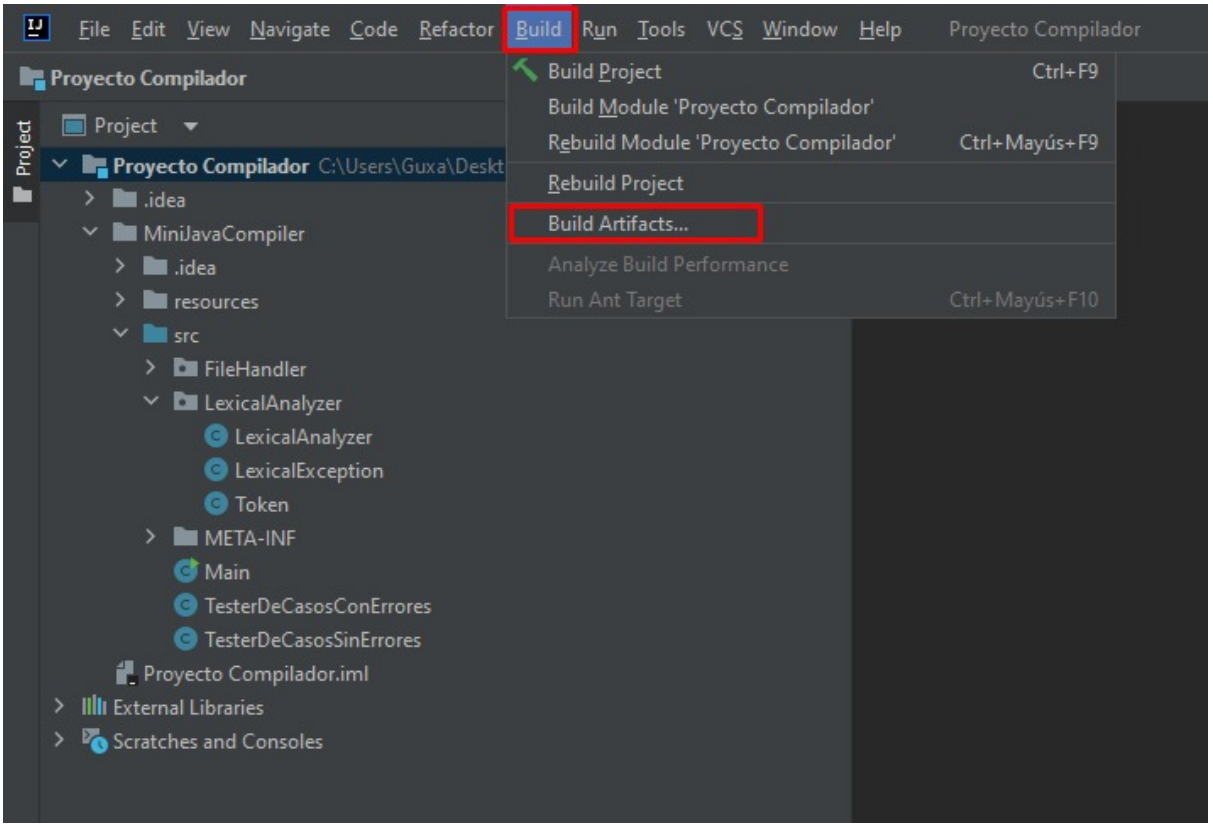


Una vez hecho esto, la pantalla debería quedar así, en la cual debemos apretar OK nuevamente.

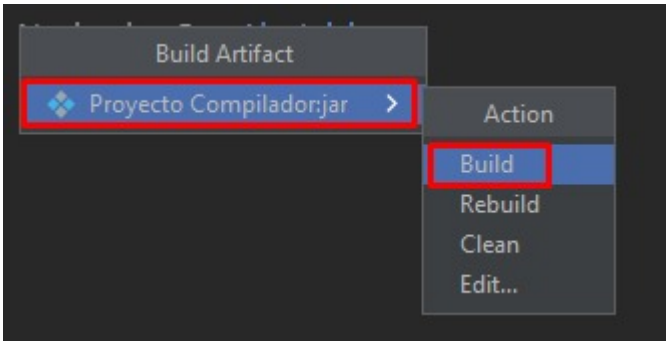


Razuc, Gonzalo

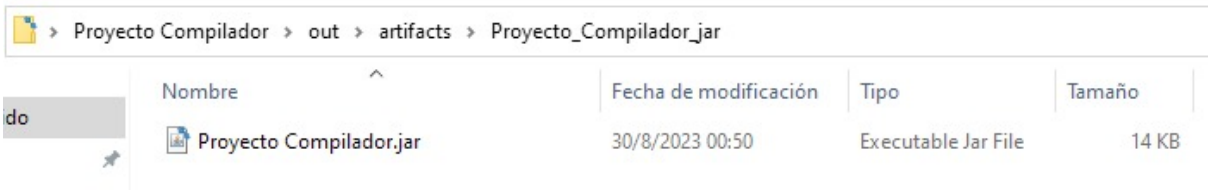
Luego, debemos ir a “Build” y seleccionar “Build Artifacts...”



Una vez hecho esto, realizar los siguientes pasos



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



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.