INFORME
ETAPA 3
ANALIZADOR SEMÁNTICO
COMPILADORES E INTÉRPRETES

Razuc Gonzalo

Errores que el analizador semántico puede detectar:

○ Nombre de clases concretas o interfaces repetidos.
○ Mismo nombre de método en una clase concreta o interfaz.
○ Una clase extiende a una clase no declarada.
○ Una clase implementa una interfaz no declarada.
○ Una interfaz extiende a una interfaz no declarada.
○ Nombre de atributo repetido en una clase, ya sea dentro de una misma clase o dentro de su línea de clases ancestras.
○ Herencia circular, ya sea entre interfaces o entre clases concretas. Ninguna clase concreta o interfaz podrá extenderse, dentro de su línea de ancestros, a sí misma.
○ Incorrecta redefinición de un método.
○ Mismo nombre de parámetro en el encabezado de un método.
○ Una clase concreta no sobrescribe todos los métodos de una interfaz que implementa.
○ Tipos no primitivos no declarados: el analizador detectará todo aquel tipo que no sea primitivo y que no esté declarado, ya sea el tipo de retorno de un método, el tipo de un atributo, o el tipo de un parámetro.
○ Método con alcance estático en una interfaz. El error se marcará en el nombre del método.
○ Ninguna clase concreta tiene un método estático denominado main y sin parámetros.
○ Más de un método main: una sola clase concreta podrá tener un método estático sin parámetros denominado “main” cuyo retorno sea de tipo void. Si se encuentra más de una clase con el método mencionado, es decir, sin parámetros, de tipo void y estático, el analizador lo detectará como un error semántico. Aclaración: si una clase hereda un método main, no se detectará como un error
○ Redefiniciones incorrectas de métodos read, printBln, etc, siempre y cuando la clase extienda la clase que contiene el método en cuestión.

Razuc Gonzalo

Decisiones de diseño y clases utilizadas

Tipos primitivos y no primitivos: para diferenciarlos, se hizo uso de una clase abstracta denominada “Tipo” que contiene el token que el analizador sintáctico reconoció en una determinada declaración, y un método “isPrimitive” que, según el tipo concreto del que se trate, retornará verdadero o falso.

La clase “TipoPrimitivo” extiende a Tipo y se utiliza para representar los tipos void, int, char y boolean. Por otro lado, la clase “TipoReferencia” extiende a Tipo y, como su nombre lo indica, es utilizada para representar los tipos no primitivos.
En caso que estemos hablando de un atributo, TipoPrimitivo sólo podrá representar tipos int, char y boolean.

Con respecto a la implementación de la clase “Method”, el alcance estático de un método se representa mediante un atributo de tipo String cuyo valor es “static” en caso de que el alcance de un método sea estático. En caso contrario, el valor del atributo será un String vacío.

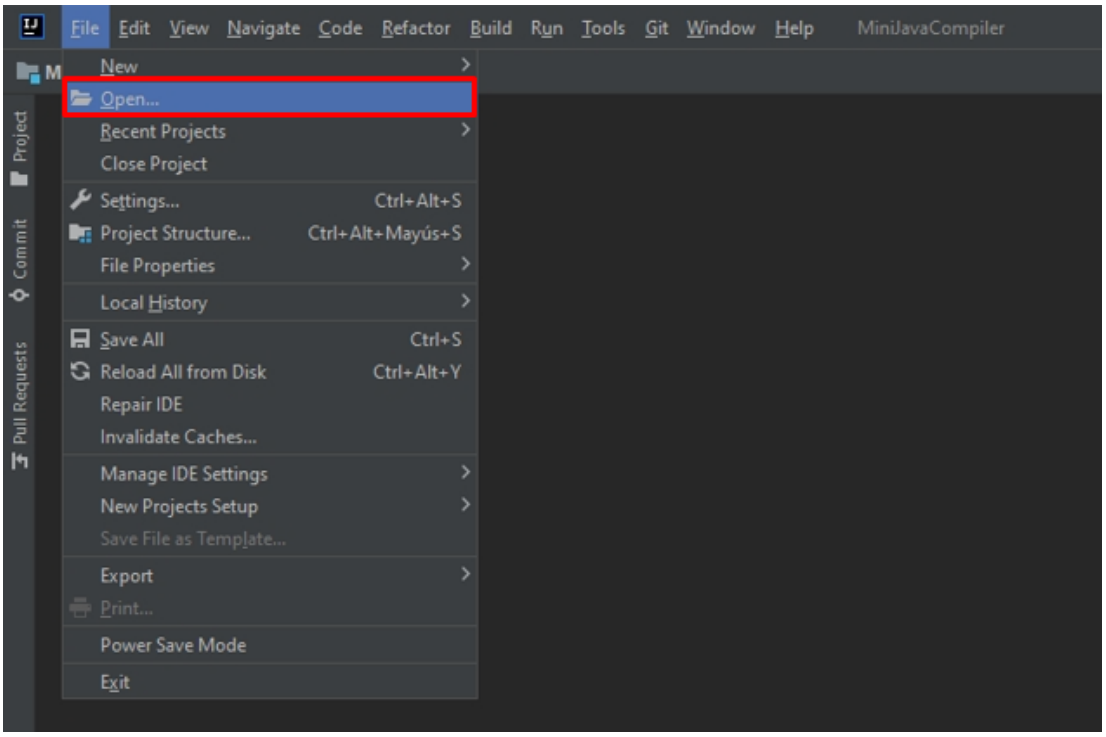
Para la implementación de las clases concretas y las interfaces, se hizo uso de una clase abstracta denominada “Clase”. Esta, contiene aquellos atributos en común entre los dos diferentes tipos de clases. También, tiene tres métodos abstractos: “insertMethod(Method methodToInsert)”, “consolidate()” y “checkDeclarations()”. El hecho de que los métodos hayan sido declarados abstractos es porque su implementación dependerá del tipo de clase con el que se esté tratando, ya que, por ejemplo, al insertar un método en una interfaz, se verifica que el alcance no sea estático. En cambio, al insertar un método en una clase concreta, este chequeo no se realiza.

Para la implementación del Constructor, se creó una clase llamada “Constructor” la cual funciona de manera similar a la clase Metodo.

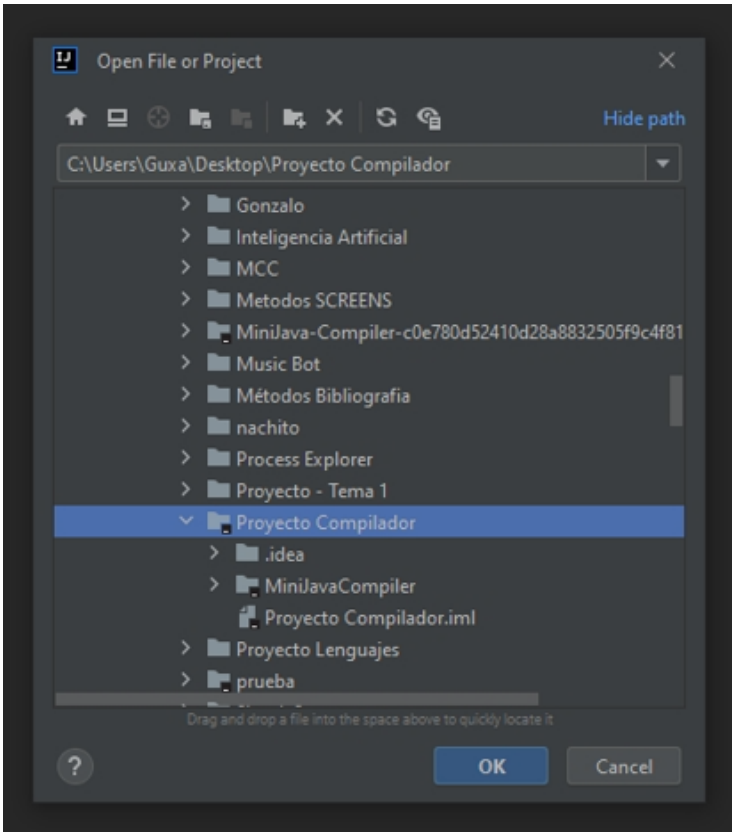
Razuc Gonzalo

Compilación del Proyecto

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

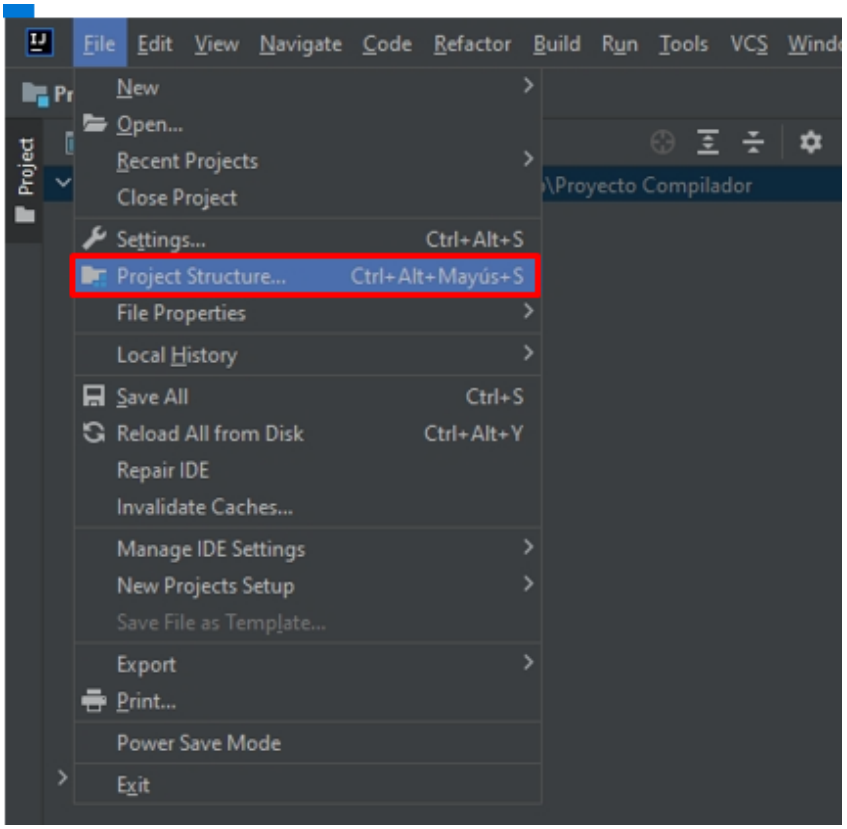


Elegir la carpeta base del proyecto

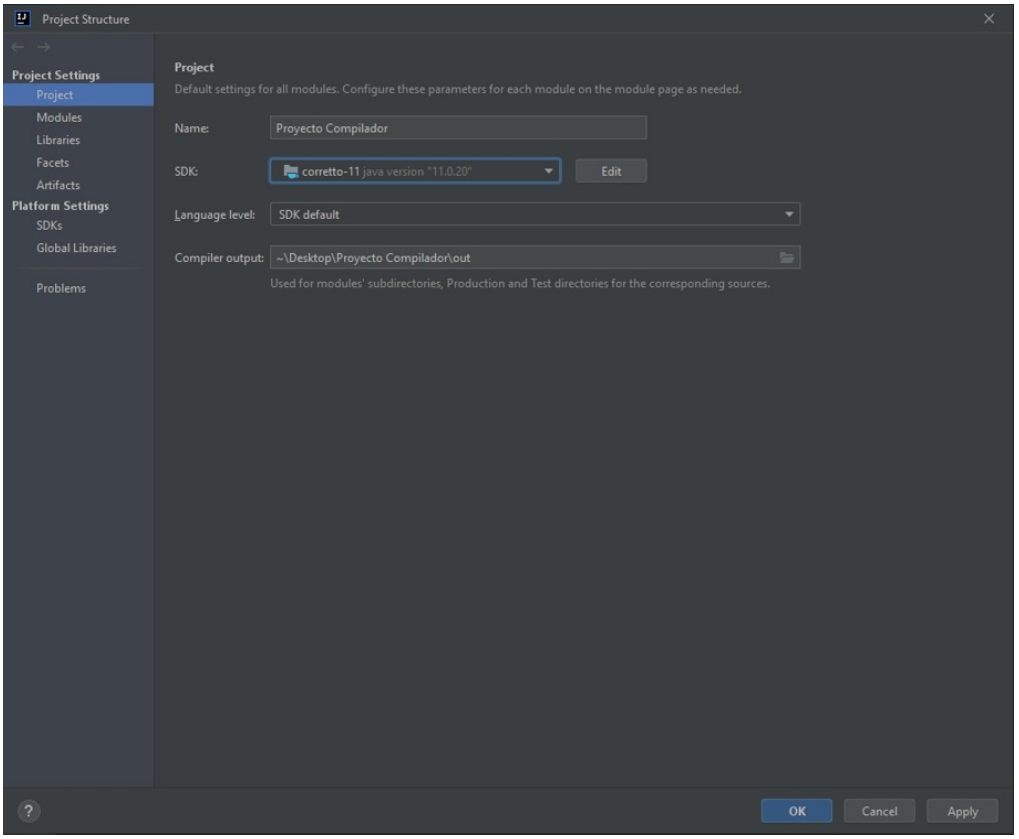


Razuc Gonzalo

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

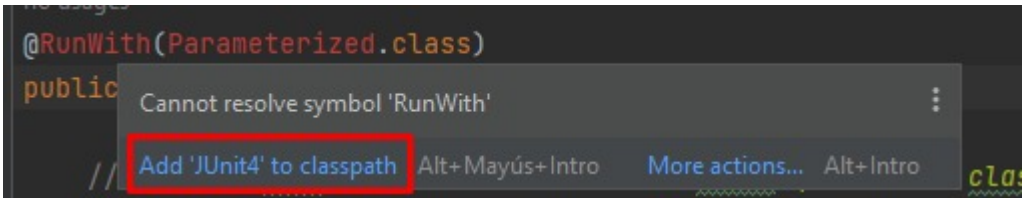


Debería verse de la siguiente manera

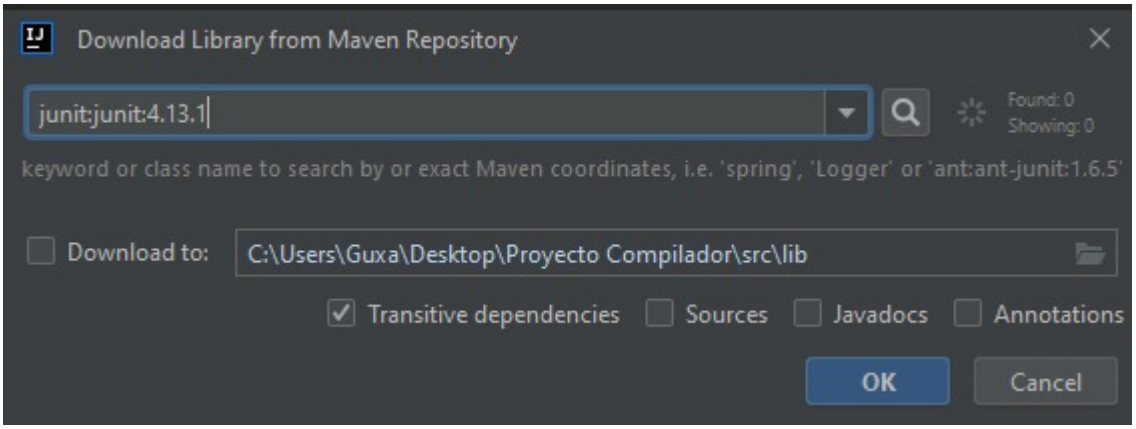


Razuc Gonzalo

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

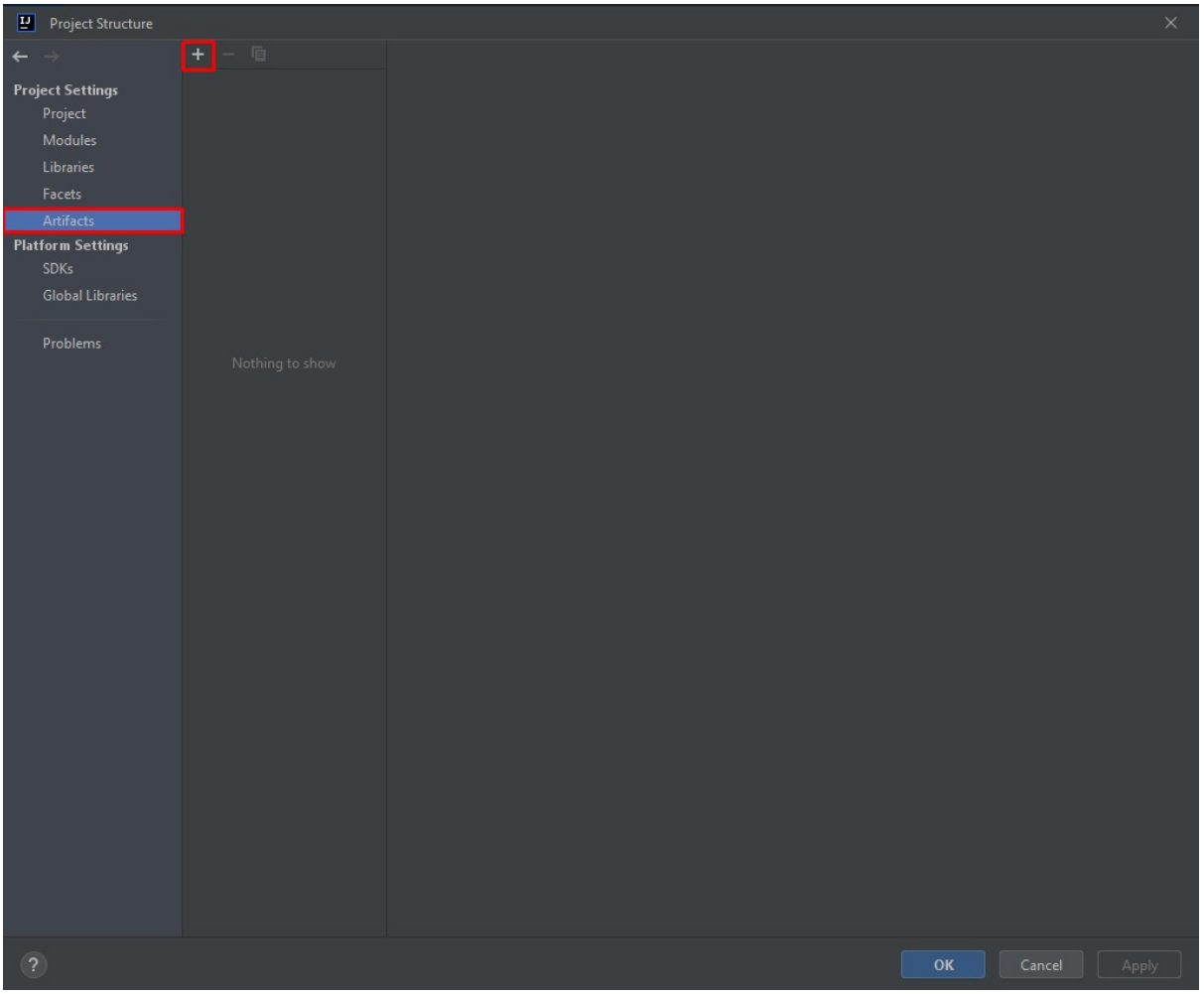


Y apretar “OK”

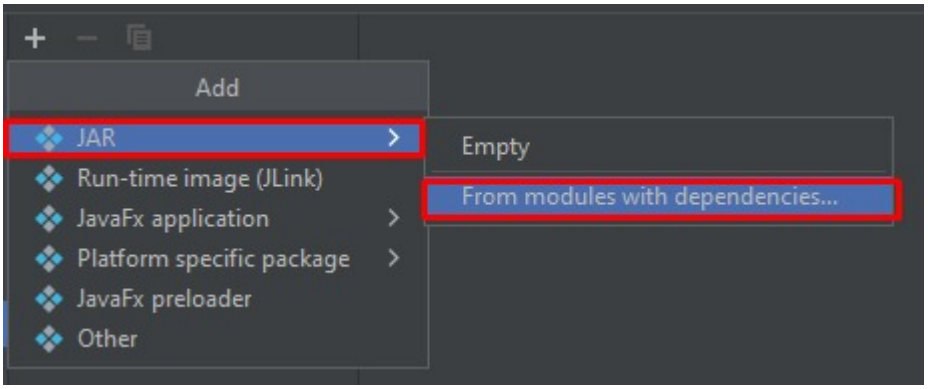


Generación del Jar

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.

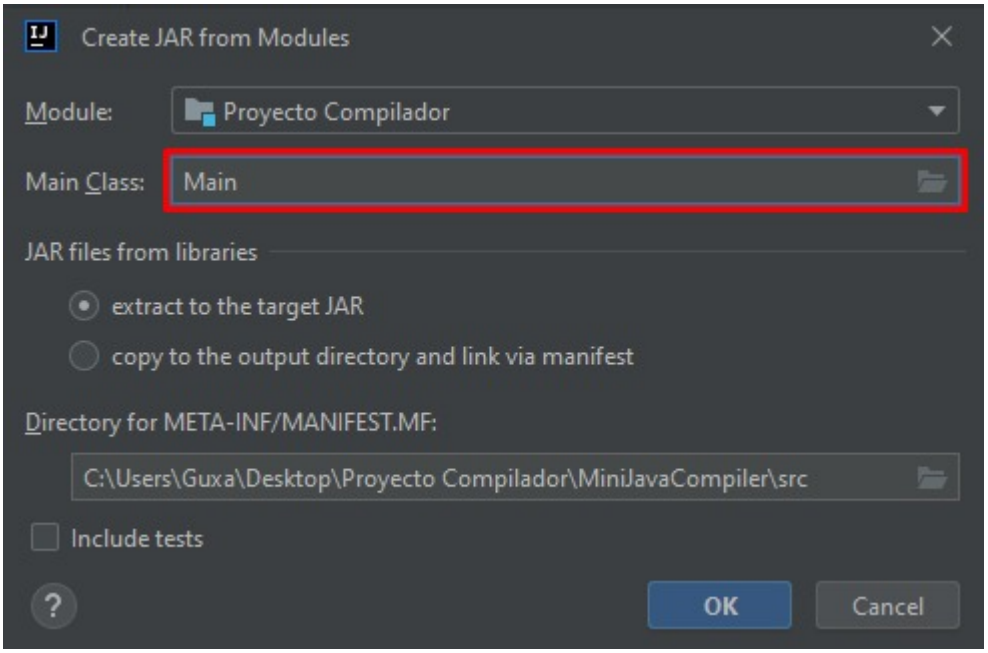


Una vez apretado el “+” debemos hacer lo siguiente

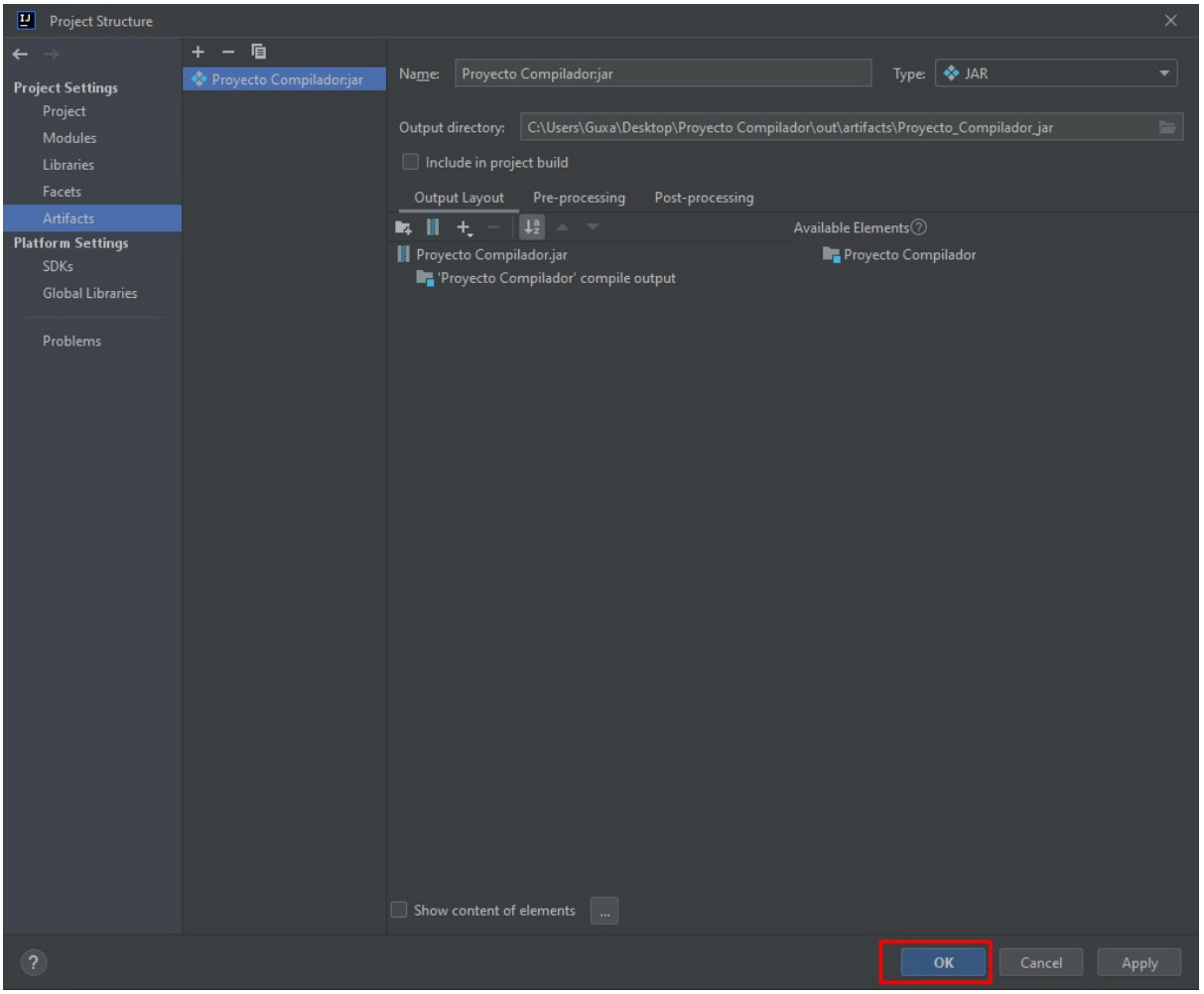


Razuc Gonzalo

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

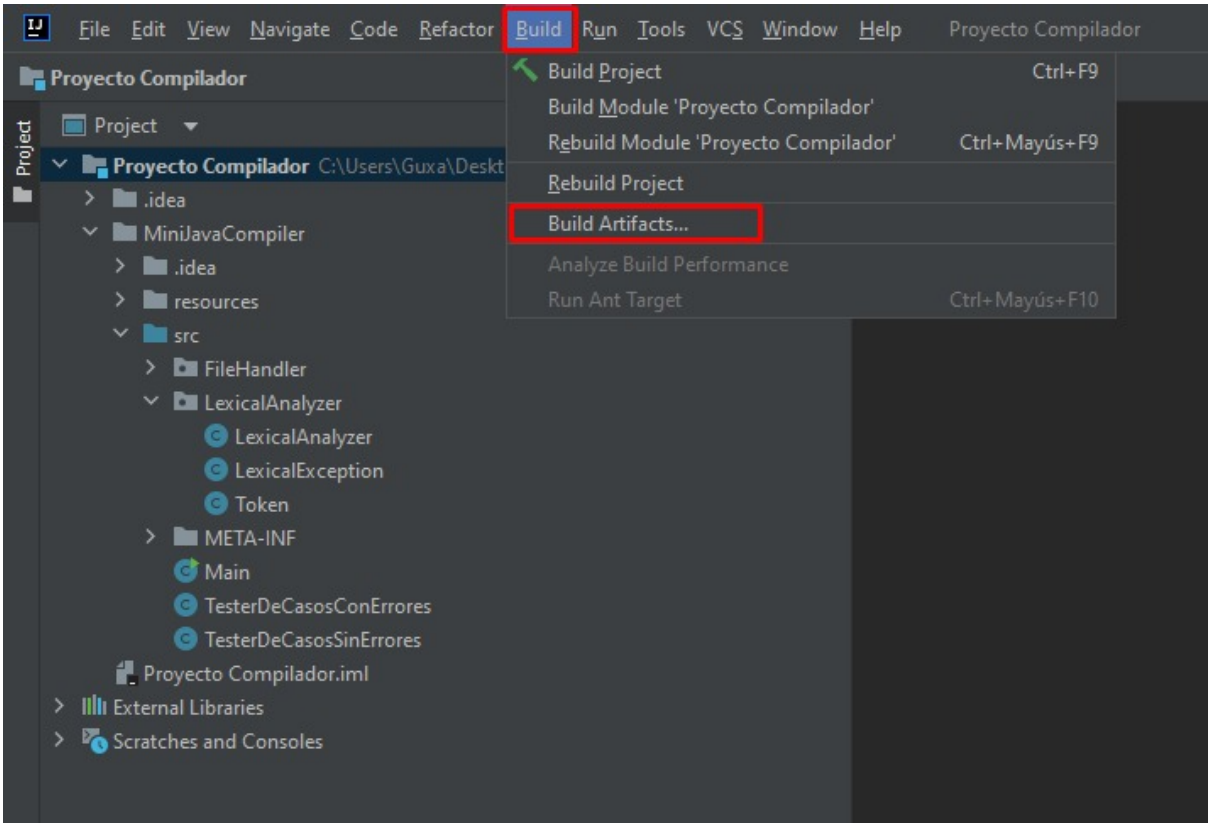


Una vez hecho esto, la pantalla debería quedar así, en la cual debemos apretar OK nuevamente.

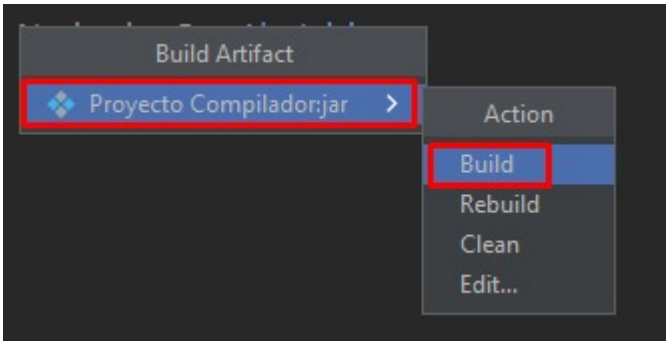


Razuc Gonzalo

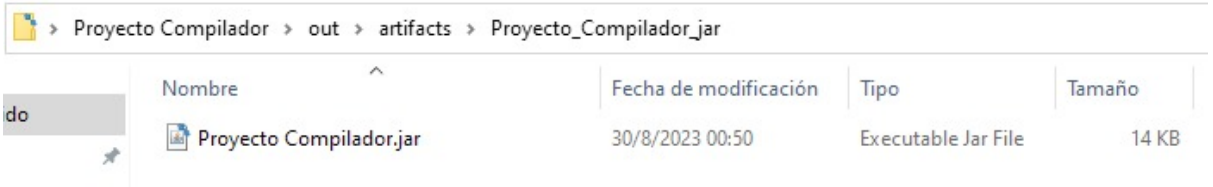
Luego, debemos ir a “Build” y seleccionar “Build Artifacts...”



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 I
- Entrega anticipada Semántica I
- Multi Detección Errores Semánticos en Declaraciones: en lugar de lanzar una excepción al encontrar un error semántico, el mensaje de error y el token se guardan en una lista. Al finalizar la tabla de símbolos de consolidar tanto interfaces como clases concretas, si la lista mencionada contiene al menos un error, se lanzará una excepción semántica que el módulo principal capturará y se imprimirán todos los errores que dicha lista contiene.

Razuc Gonzalo