INFORME
ETAPA 2
ANALIZADOR SINTACTICO
COMPILADORES E INTÉRPRETES

Razuc Gonzalo

Modificación de la gramática

En primer paso, identifico las producciones de la gramática que contengan recursión a izquierda.

<ExpresionCompuesta> ::=
<ExpresionCompuesta> <OperadorBinario> <ExpresionBasica>
<ExpresionCompuesta> ::= <ExpresionBasica>

Lo soluciono aplicando factorización simple:
<ExpresionCompuesta> ::= <ExpresionBasica> <ExpresionCompuestaPrima>
<ExpresionCompuestaPrima> ::=
<OperadorBinario> <ExpresionBasica> <ExpresionCompuestaPrima> | ε

Ahora, procedo a identificar las producciones de la gramatica que contengan un prefijo izquierdo comun y puedan ser solucionadas aplicando factorizacion a izquierda de manera directa

<ListaArgsFormales> ::= <ArgFormal>
<ListaArgsFormales> ::= <ArgFormal> , <ListaArgsFormales>

<If> ::= if (<Expresion>) <Sentencia>
<If> ::= if (<Expresion>) <Sentencia> else <Sentencia>

<Expresion> ::= <ExpresionCompuesta> = <Expresion>
<Expresion> ::= <ExpresionCompuesta>

<ListaExps> ::= <Expresion>
<ListaExps> ::= <Expresion> , <ListaExps>

Las soluciono aplicando factorizacion a izquierda directa, quedando de la siguiente manera:

<ListaArgsFormales> ::= <ArgFormal> <ArgFormalPrima>
<ArgFormalPrima>::= , <ListaArgsFormales> | ε

<If> ::= if (<Expresion>) <Sentencia> <ElseOpcional>
<ElseOpcional> :== else<Sentencia> | ε

<Expresion> ::= <ExpresionCompuesta> <ExpresionPrima>
<ExpresionPrima> ::= = <Expresion> | ε

<ListaExps> ::= <Expresion> <ListaExpresionesPrimas>
<ListaExpresionesPrimas> ::= , <ListaExps> | ε

Razuc Gonzalo

Procedo a identificar las producciones de la gramática que contengan un prefijo izquierdo común y deben ser solucionadas aplicando factorización a izquierda profunda

<Sentencia> ::= ;
<Sentencia> ::= <Asignacion> ;
<Sentencia> ::= <Llamada> ;
<Sentencia> ::= <VarLocal> ;
<Sentencia> ::= <Return> ;
<Sentencia> ::= <If>
<Sentencia> ::= <While>
<Sentencia> ::= <Bloque>

<Primario> ::= <AccesoThis>
<Primario> ::= <AccesoVar>
<Primario> ::= <AccesoConstructor>
<Primario> ::= <AccesoMetodo>
<Primario> ::= <AccesoMetodoEstatico>
<Primario> ::= <ExpresionParentizada>

<EncadenadoOpcional> ::= <VarEncadenada> | <MetodoEncadenado> | ε

En <Sentencia> los primeros de <Asignacion> y <Llamada> son los mismos, por ende, su intersección no es vacía.
Al aplicar factorización a izquierda profunda, la gramatica queda así.

<Sentencia> ::= ;
<Sentencia> ::= <Expresion> ;
<Sentencia> ::= <VarLocal> ;
<Sentencia> ::= <Return> ;
<Sentencia> ::= <If>
<Sentencia> ::= <While>
<Sentencia> ::= <Bloque>

Procedo con la siguiente producción, en este caso, <Primario>.
<AccesoMetodo> y <AccesoVar> tienen los mismos primeros, por ende, su intersección no es vacía.
Al aplicar factorización a izquierda profunda, la gramática queda así.

<Primario> ::= idMetVar <PrimarioOpcional>
<Primario> ::= <AccesoThis>
<Primario> ::= <AccesoConstructor>
<Primario> ::= <AccesoMetodoEstatico>
<Primario> ::= <ExpresionParentizada>
<PrimarioOpcional> :== <ArgsActuales> | ε

En <EncadenadoOpcional> la intersección entre los primeros de <VarEncadenada> y <MetodoEncadenado> no es vacía. Por ende, se soluciona de la siguiente manera.

<EncadenadoOpcional> ::= . idMetVar<EncadenadoOpcionalPrima> | ε
<EncadenadoOpcionalPrima> ::= <EncadenadoOpcional> | <ArgsActuales> <EncadenadoOpcional>

La gramática LL(1) resultante de aplicar las modificaciones mencionadas fue la siguiente:

<Inicial> ::= <ListaClases>
<ListaClases> ::= <Clase> <ListaClases> | ε
<Clase> ::= <ClaseConcreta> | <Interface>
<ClaseConcreta> ::= class idClase <HerenciaOpcional> { <ListaMiembros> }
<Interface> ::= interface idClase <ExtiendeOpcional> { <ListaEncabezados> }
<HerenciaOpcional> ::= <HeredaDe> | <ImplementaA> | ε
<HeredaDe> ::= extends idClase
<ImplementaA> ::= implements idClase
<ExtiendeOpcional> ::= extends idClase | ε
<ListaMiembros> ::= <Miembro> <ListaMiembros> | ε
<ListaEncabezados> ::= <EncabezadoMetodo> <ListaEncabezados> | ε
<Miembro> ::= <EstaticoOpcional> <TipoMiembro> idMetVar <AtributoOMetodo> | <Constructor>
<AtributoOMetodo> :: = <ArgsFormales> <Bloque> | ;
<EncabezadoMetodo> ::= <EstaticoOpcional> <TipoMiembro> idMetVar <ArgsFormales> ;
<Constructor> ::= public idClase <ArgsFormales> <Bloque>
<TipoMiembro> ::= <Tipo> | void
<Tipo> ::= <TipoPrimitivo> | idClase
<TipoPrimitivo> ::= boolean | char | int
<EstaticoOpcional> ::= static | ε
<ArgsFormales> ::= (<ListaArgsFormalesOpcional>)
<ListaArgsFormalesOpcional> ::= <ListaArgsFormales> | ε
<ListaArgsFormales> ::= <ArgFormal> <ArgFormalPrima>
<ArgFormalPrima>::= , <ListaArgsFormales> | ε
<ArgFormal> ::= <Tipo> idMetVar
<Bloque> ::= { <ListaSentencias> }
<ListaSentencias> ::= <Sentencia> <ListaSentencias> | ε
<Sentencia> ::= ; | <Expresion> ; | <VarLocal> ; | <Return> ; | <If> | <While > | <Bloque>
<VarLocal> ::= var idMetVar = <ExpresionCompuesta>
<Return> ::= return <ExpresionOpcional>
<ExpresionOpcional> ::= <Expresion> | ε
<If> ::= if (<Expresion>) <Sentencia> <ElseOpcional>
<ElseOpcional> :== else <Sentencia> | ε

Razuc Gonzalo

<While> ::= while (<Expresion>) <Sentencia>
<Expresion> ::= <ExpresionCompuesta> <ExpresionPrima>
<ExpresionPrima> ::= = <Expresion> | ε
<ExpresionCompuesta> ::= <ExpresionBasica> <ExpresionCompuestaPrima>
<ExpresionCompuestaPrima> ::= <OperadorBinario> <ExpresionBasica>
<ExpresionCompuestaPrima> | ε
<OperadorBinario> ::= || | && | == | != | < | > | <= | >= | + | - | * | / | %
<ExpresionBasica> ::= <OperadorUnario> <Operando>
<ExpresionBasica> ::= <Operando>
<OperadorUnario> ::= + | - | !
<Operando> ::= <Literal>
<Operando> ::= <Acceso>
<Literal> ::= null | true | false | intLiteral | charLiteral | stringLiteral
<Acceso> ::= <Primario> <EncadenadoOpcional>
<Primario> ::= idMetVar <PrimarioOpcional> | <AccesoThis> |
<AccesoConstructor> | <AccesoMetodoEstatico> | <ExpresionParentizada>
<PrimarioOpcional> :== <ArgsActuales> | ε
<AccesoThis> ::= this
<AccesoConstructor> ::= new idClase <ArgsActuales>
<ExpresionParentizada> ::= (<Expresion>)
<AccesoMetodoEstatico> ::= idClase . idMetVar <ArgsActuales>
<ArgsActuales> ::= (<ListaExpsOpcional>)
<ListaExpsOpcional> ::= <ListaExps> | ε
<ListaExps> ::= <Expresion> <ListaExpresionesPrimas>
<ListaExpresionesPrimas> ::= , <ListaExps> | ε
<EncadenadoOpcional> ::= . idMetVar<EncadenadoOpcionalPrima> | ε
<EncadenadoOpcionalPrima> ::= <EncadenadoOpcional> | <ArgsActuales>
<EncadenadoOpcional>

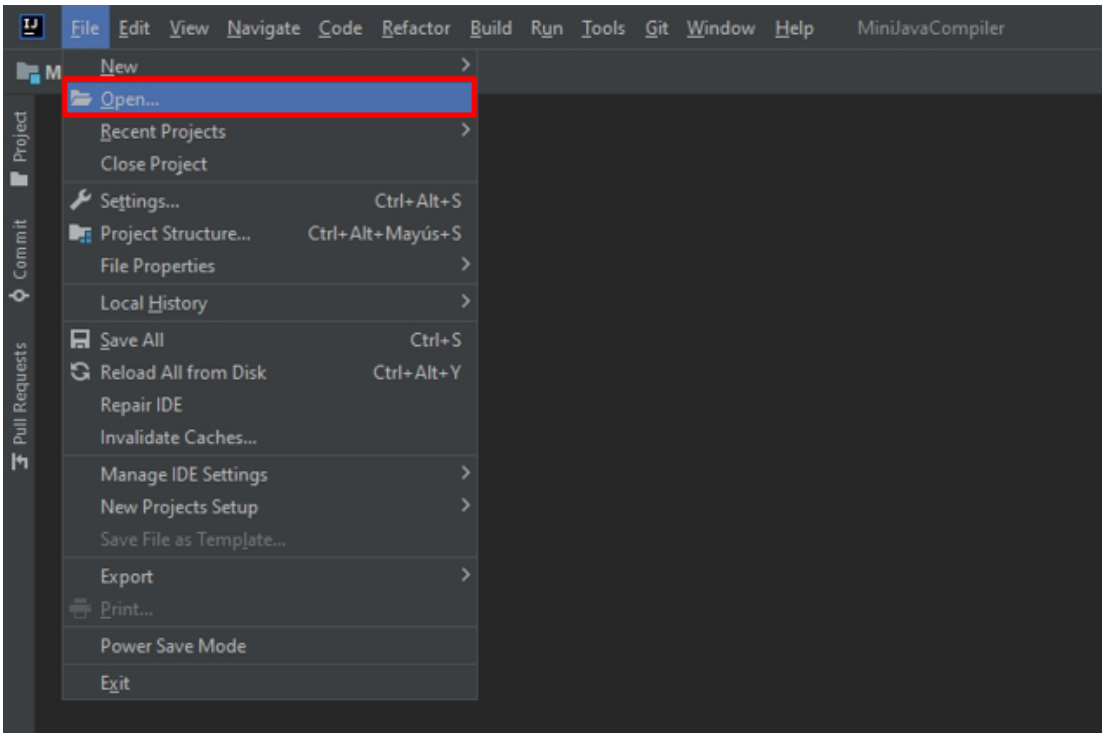
Logros que se intentaron alcanzar

Entrega anticipada sintáctica
Imbatibilidad sintáctica
Multi-detección de Errores Léxicos

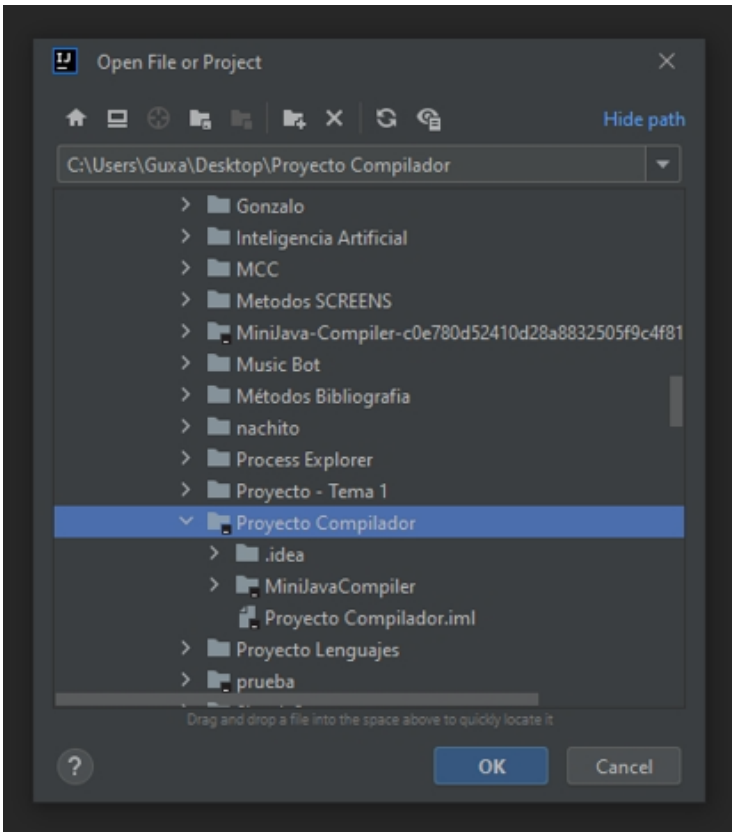
Cabe destacar que se entregaron dos zips, uno llamado "*LexicalAnalyzer.zip*" y el otro "*SyntacticAnalyzer.zip*". Para probar la multi detección de errores léxicos, se debe descomprimir "*LexicalAnalyzer.zip*" y en el proceso de compilación del proyecto y generación del jar (explicados en la siguiente página) **se debe utilizar el Main y los archivos de ese zip/proyecto**.
Luego, para probar el sintáctico, se debe descomprimir el zip "*SyntacticAnalyzer*" y realizar exactamente los mismos pasos de compilación y generación, **pero utilizando el Main y los archivos de ese zip/proyecto.**

Compilación del Proyecto

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

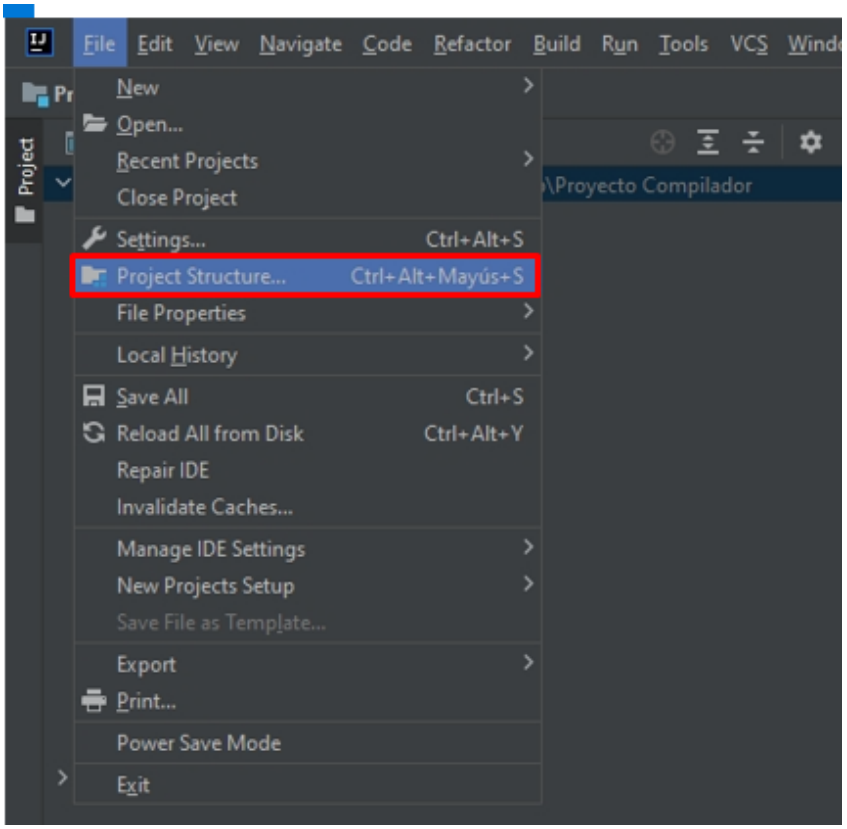


Elegir la carpeta base del proyecto

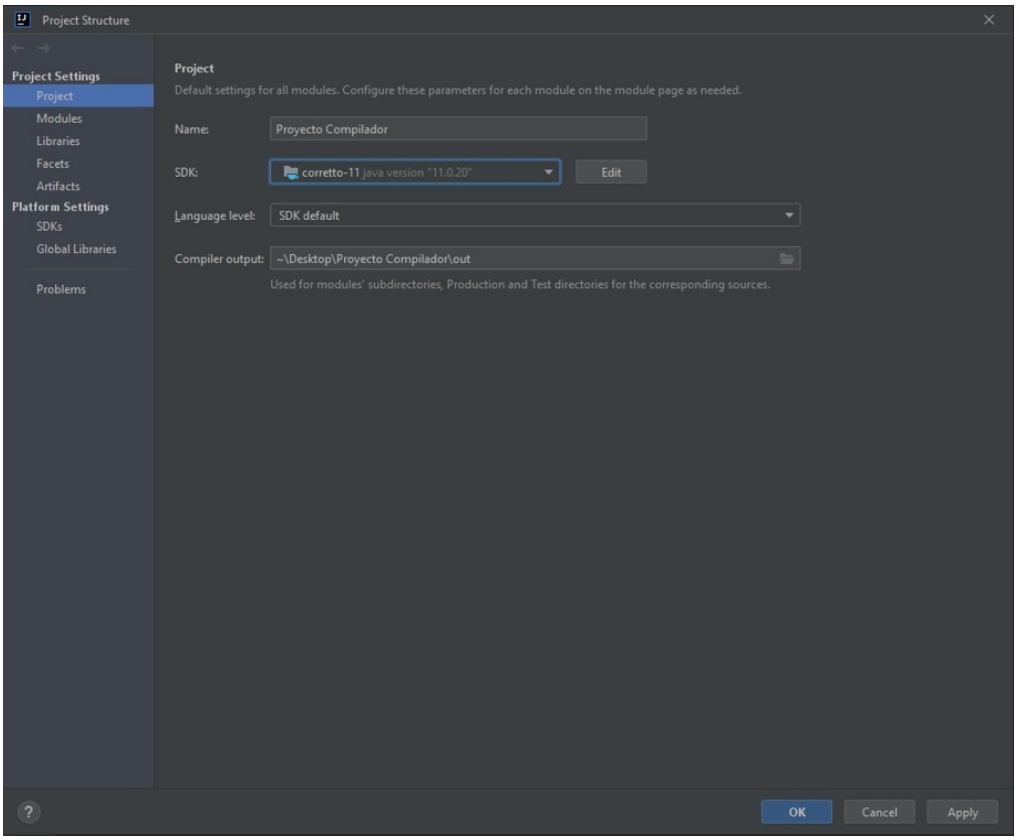


Razuc Gonzalo

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

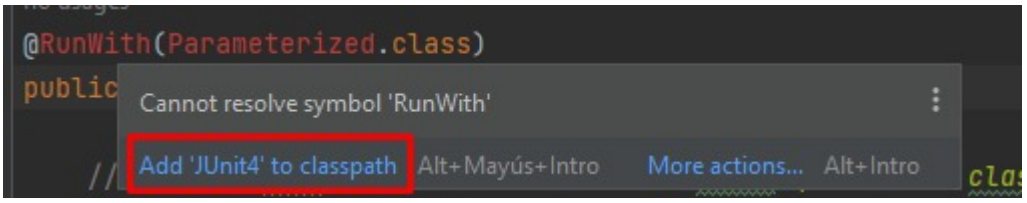


Debería verse de la siguiente manera

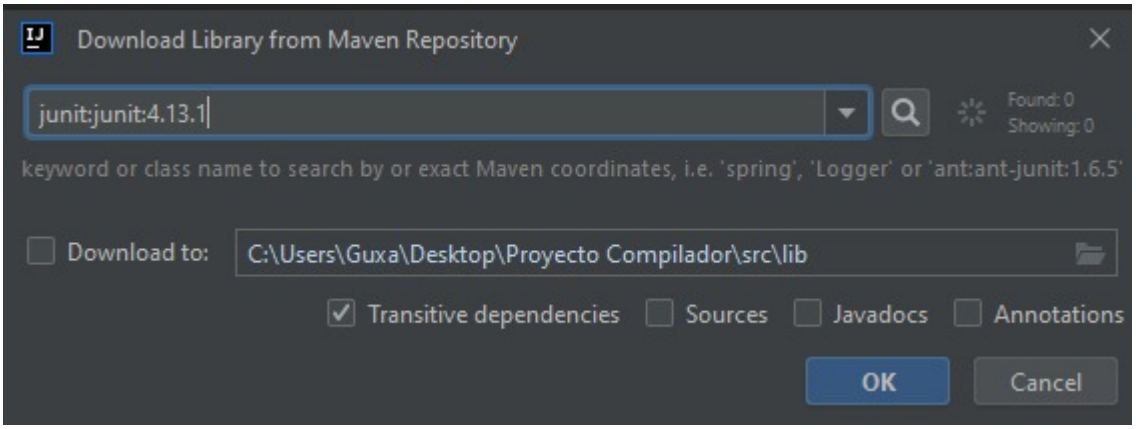


Razuc Gonzalo

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

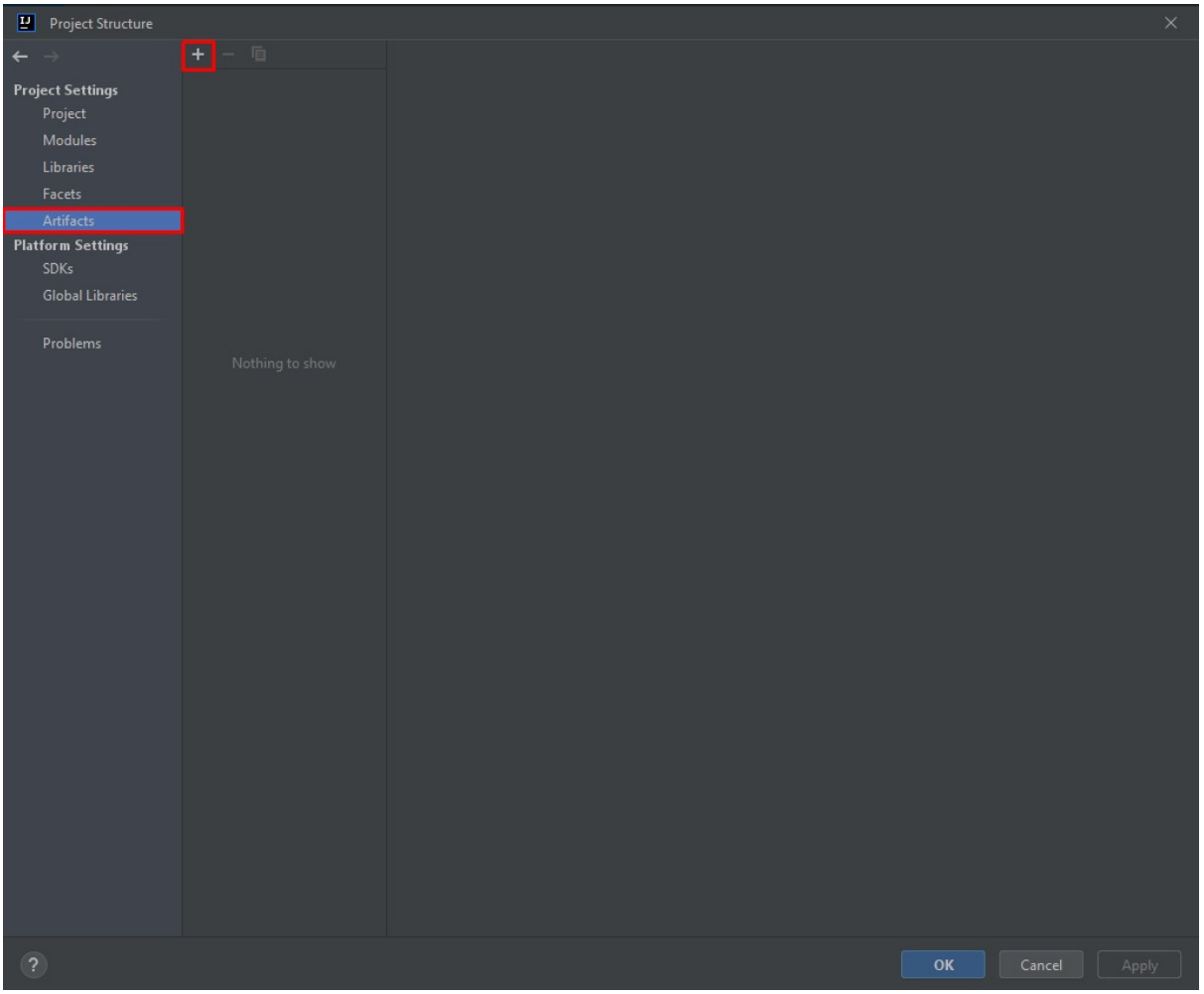


Y apretar “OK”

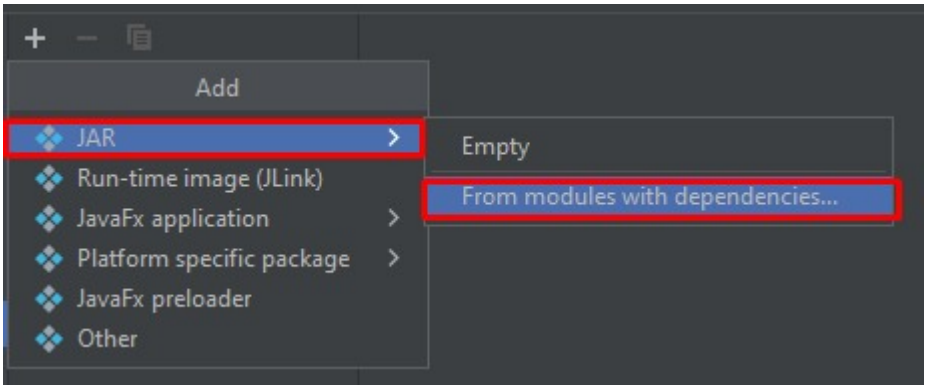


Generación del Jar

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.

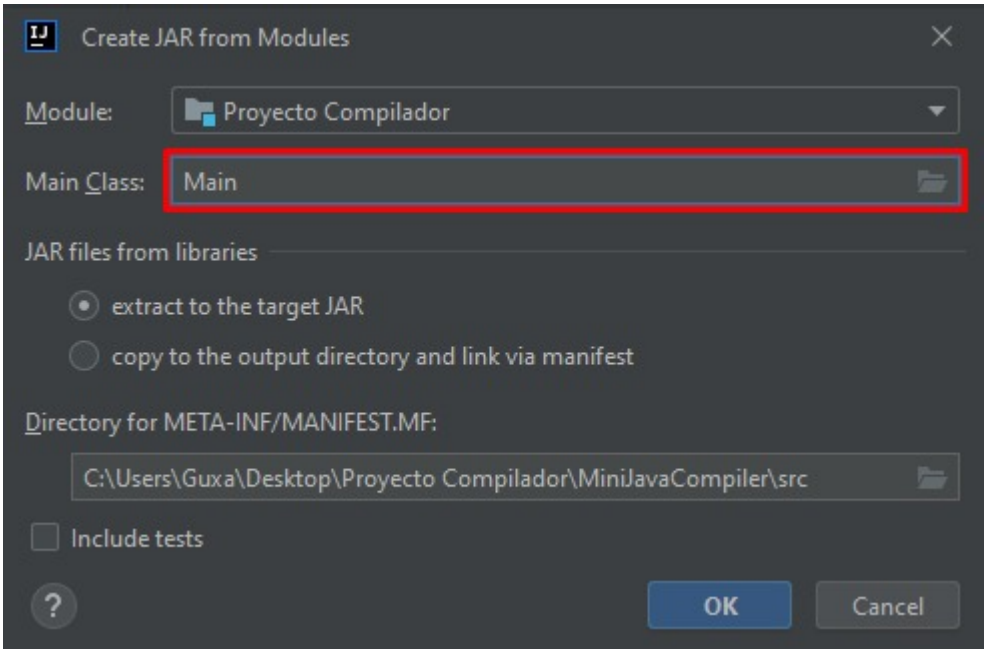


Una vez apretado el “+” debemos hacer lo siguiente

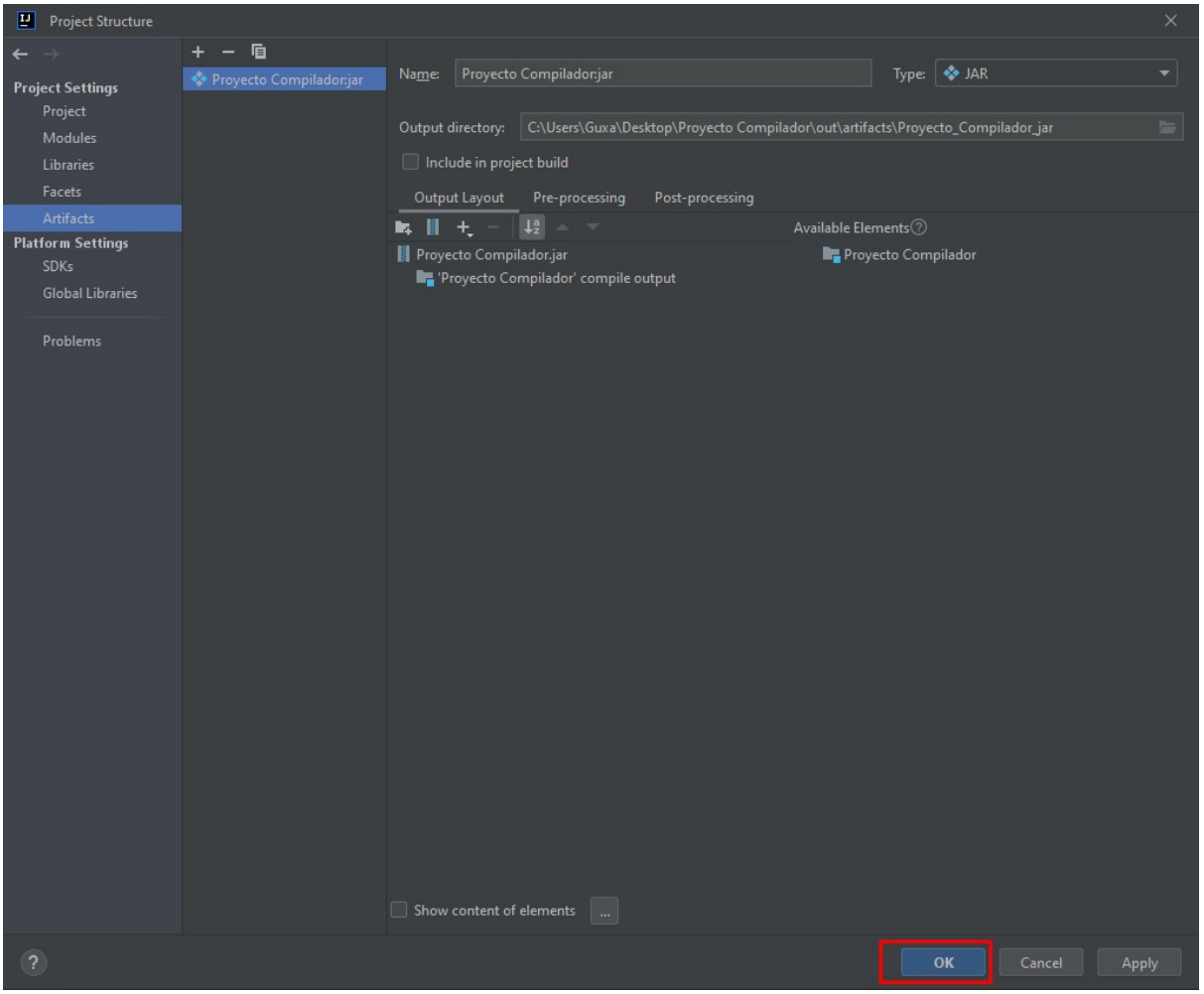


Razuc Gonzalo

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

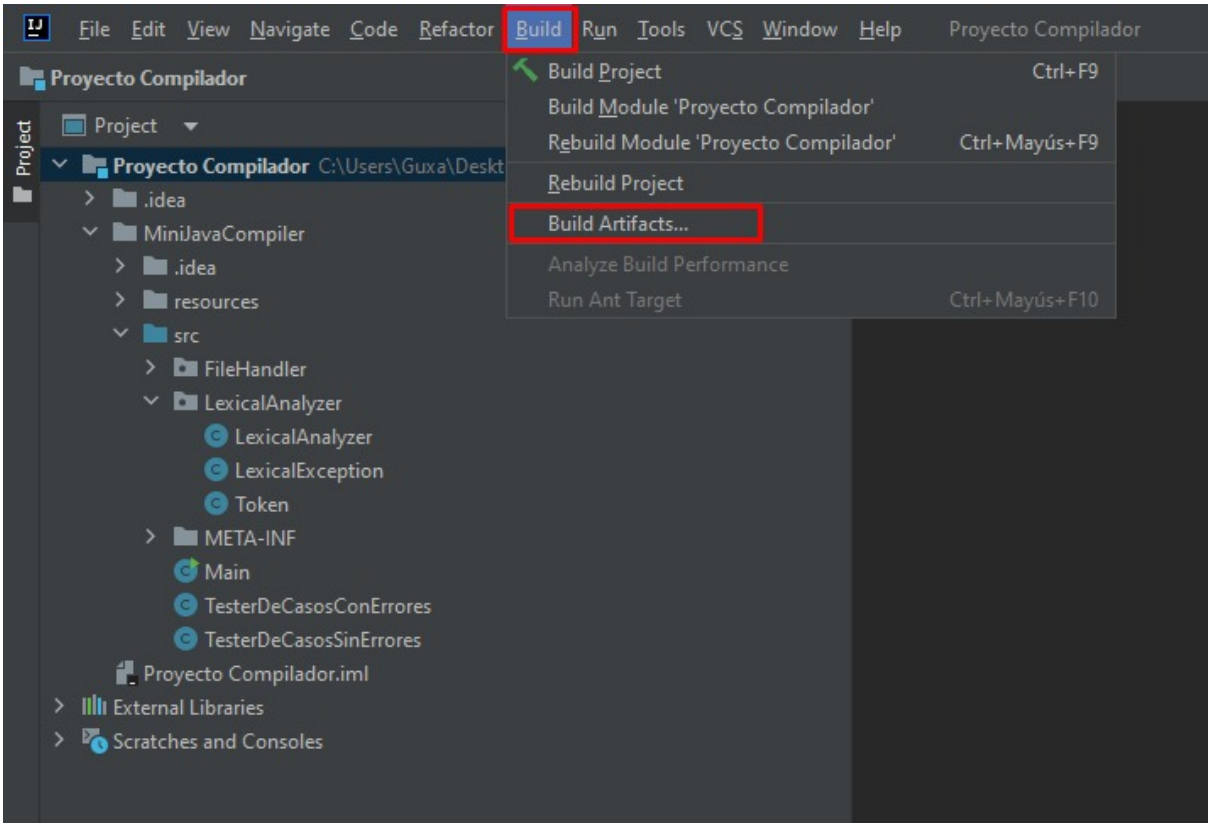


Una vez hecho esto, la pantalla debería quedar así, en la cual debemos apretar OK nuevamente.

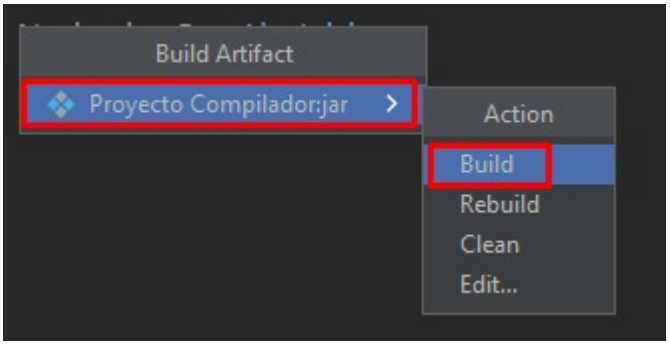


Razuc Gonzalo

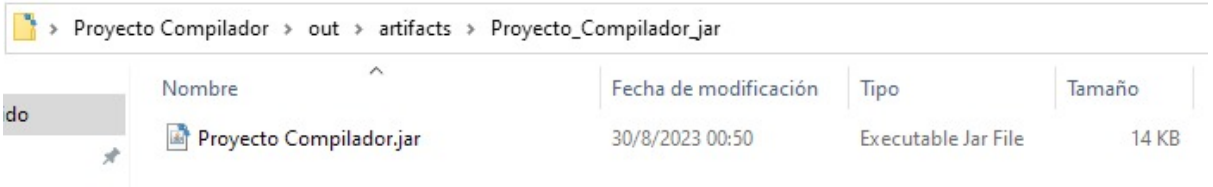
Luego, debemos ir a “Build” y seleccionar “Build Artifacts...”



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.

Razuc Gonzalo