

**LPROG**

**2020/21**

**Exemplo de  
utilização do  
ANTLR no IntelliJ**

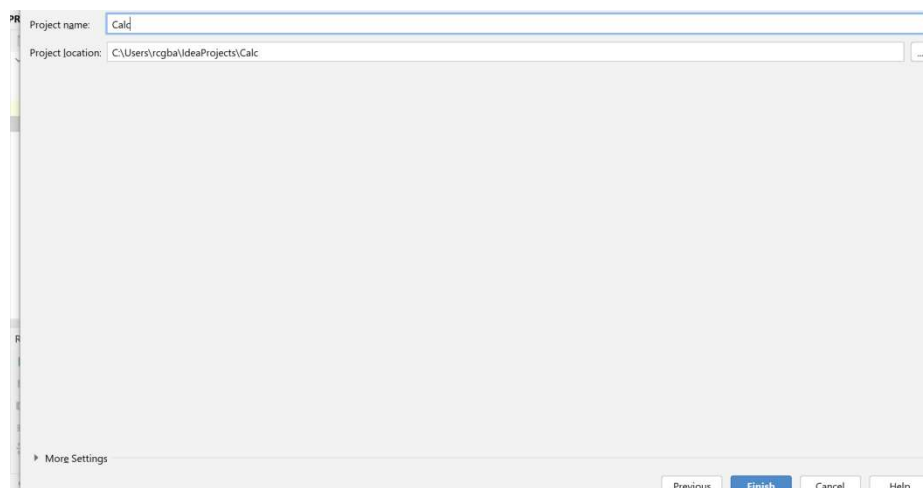
## Índice

1. Criar um projeto com o IntelliJ.....	3
2. Instalar o Plugin ANTLR .....	3
3. Descarregar a biblioteca ANTLR e acrescentar ao projeto .....	3
4. Criar/editar uma gramática no directório source (src/Calc) do projeto.....	4
5. Testar a gramática usando a opção Test Rule .....	5
6. Gerar os ficheiros Java .....	6
7. Criar outros ficheiros java para completar o projeto.....	8
8. Execução do programa completo .....	10

## 1. Criar um projeto com o IntelliJ

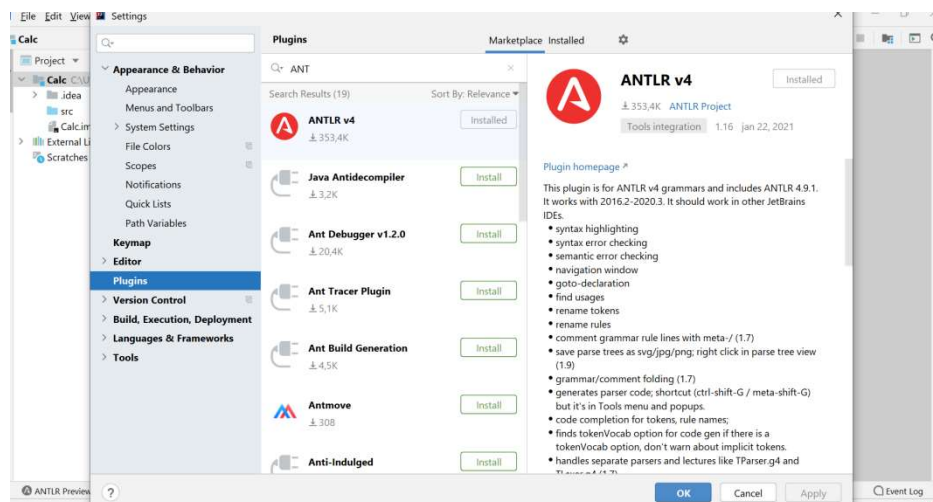
Iniciar o IntelliJ.

Criar um projeto “Java Application” com nome “Calc”.



## 2. Instalar o Plugin ANTLR

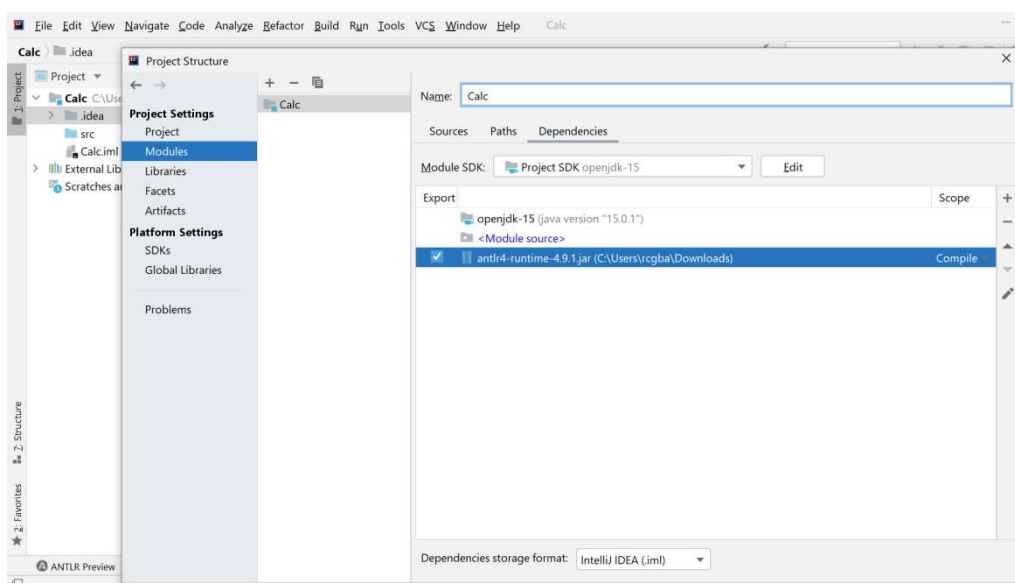
File -> Settings -> Plugins



## 3. Descarregar a biblioteca ANTLR e acrescentar ao projeto

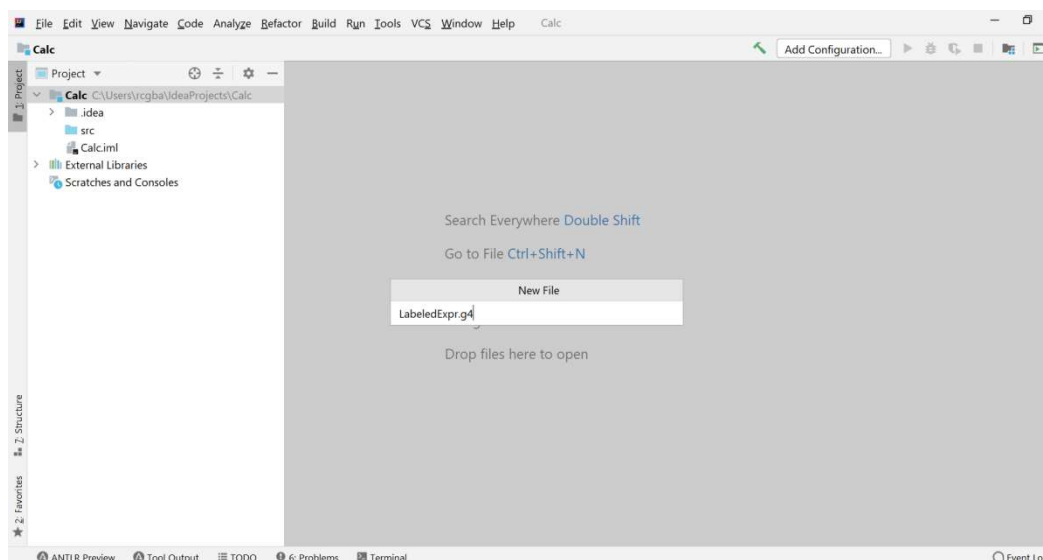
File -> Project Structure -> Modules -> Dependencies -> +

Carregar o ficheiro antlr4-runtime-4.9.1.jar, ativar a caixa de seleção.



#### 4. Criar/editar uma gramática no directório source (src/Calc) do projeto

New -> File



Adicionar o texto ao ficheiro **LabeledExpr.g4** e gravar.

---

grammar LabeledExpr;

prog: stat+ ;

stat: expr NEWLINE # printExpr  
 | ID '=' expr NEWLINE # assign  
 | NEWLINE # blank

```

;
expr: expr op=('*' | '/') expr # MulDiv
    | expr op=('+' | '-') expr # AddSub
    | INT # int
    | ID # id
    | '(' expr ')' # parens
;
NEWLINE : [\r\n]+ ;
INT:[0-9]+;
ID:[a-z]+;
MUL : '*'; // assigns token name to '*' used above in grammar
DIV : '/';
ADD : '+';
SUB : '-';
ATR : '=';
LPR : '(';
RPR : ')';

```

---

Criar um ficheiro chamado “teste.txt” na pasta base do projeto IntelliJ com o seguinte texto:

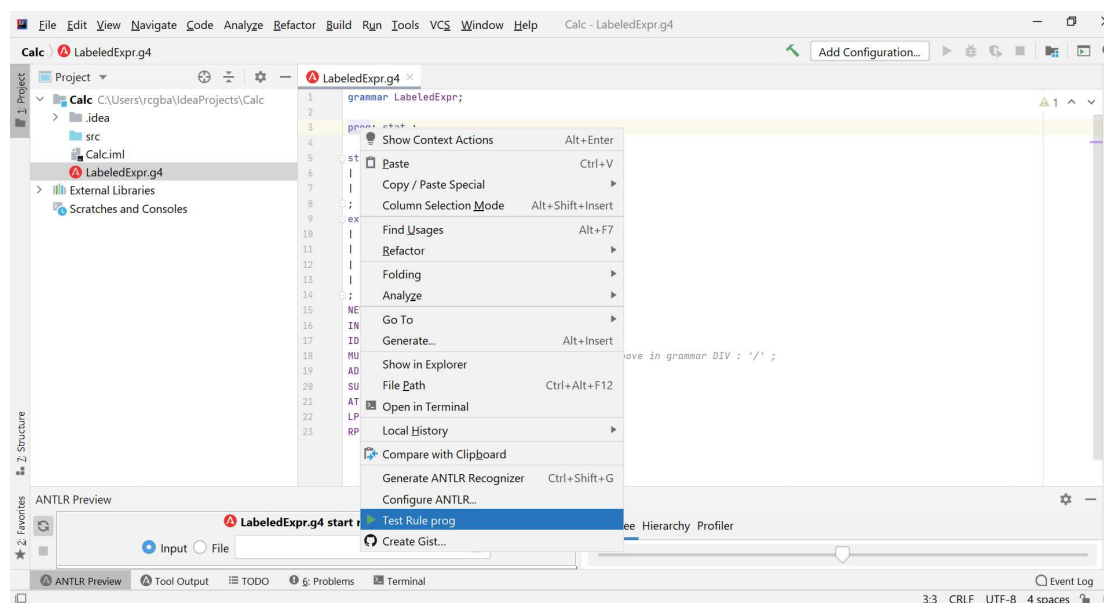
```

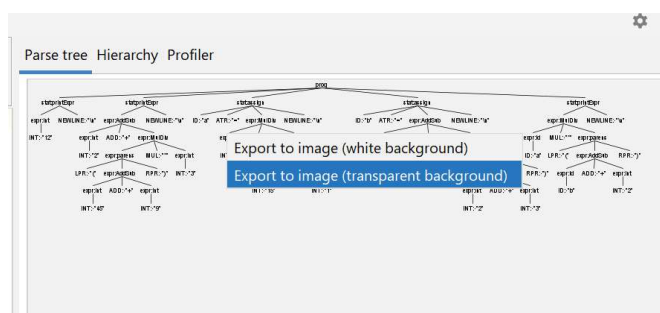
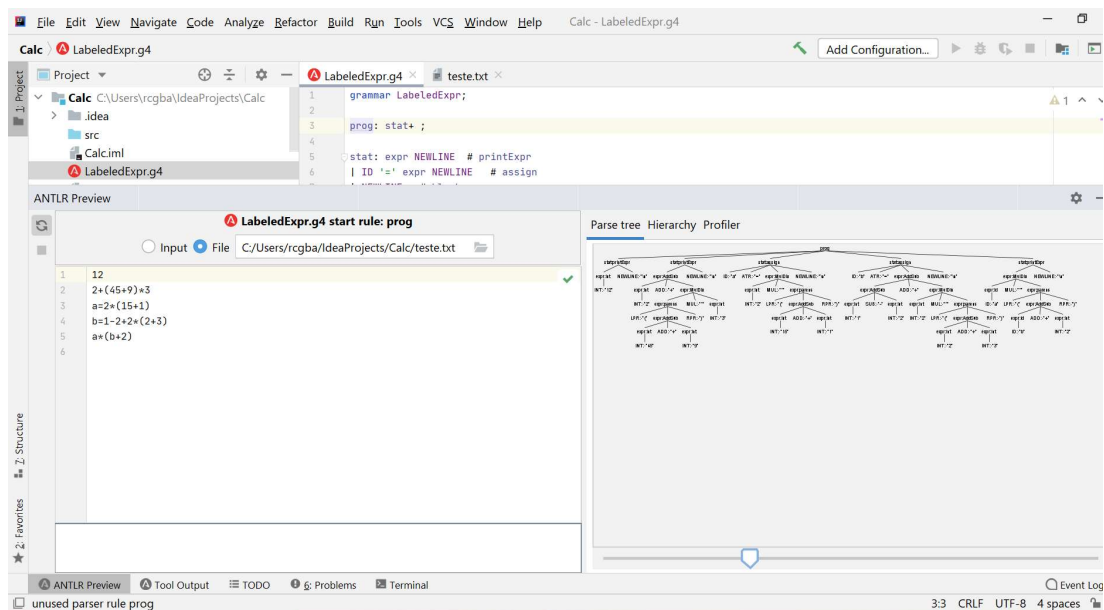
12
2+ (45+9) *3
a=2* (15+1)
b=12+2* (2+3)
a* (b+2)

```

---

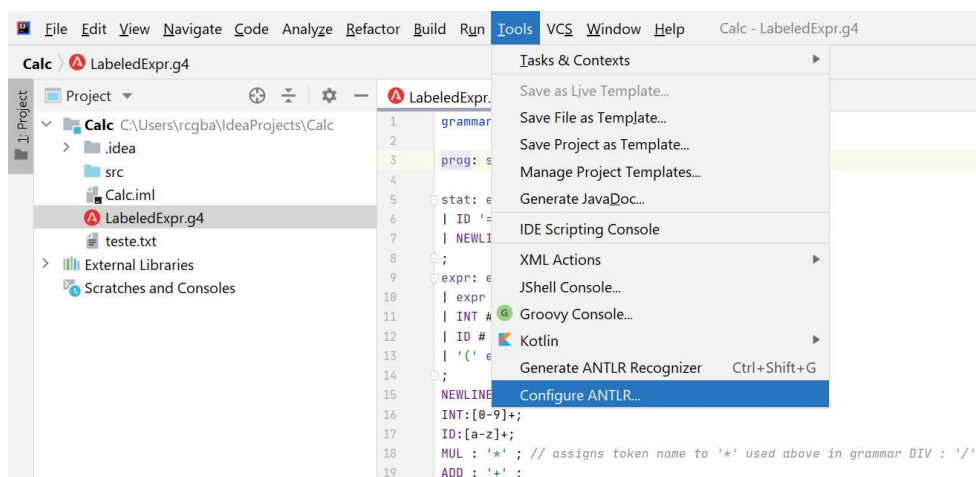
## 5. Testar a gramática usando a opção Test Rule

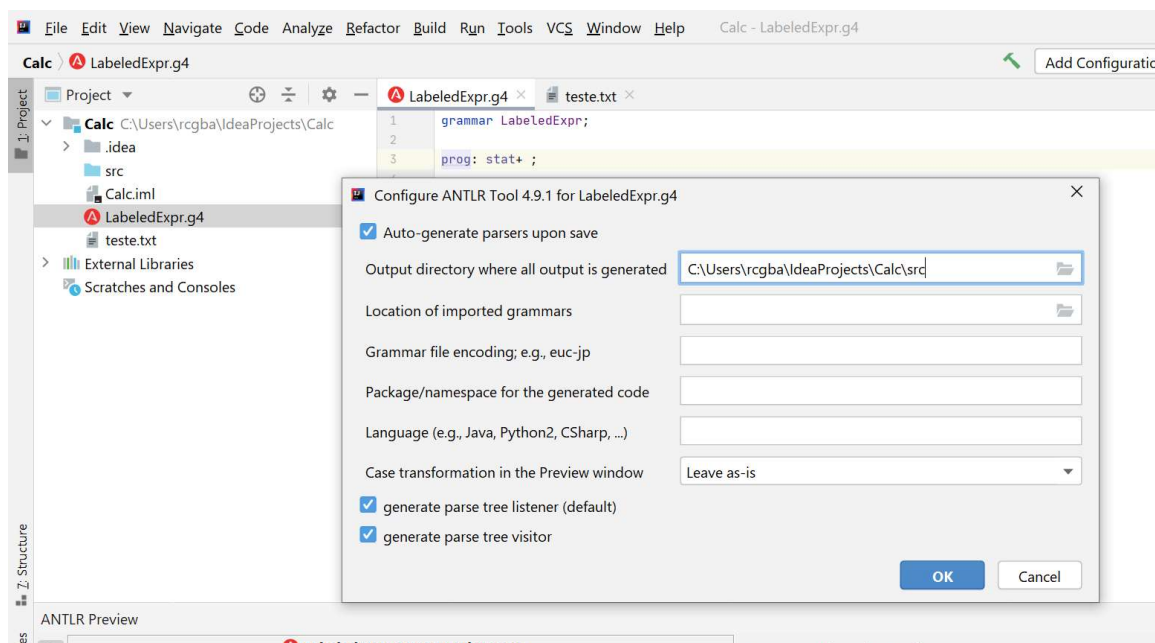




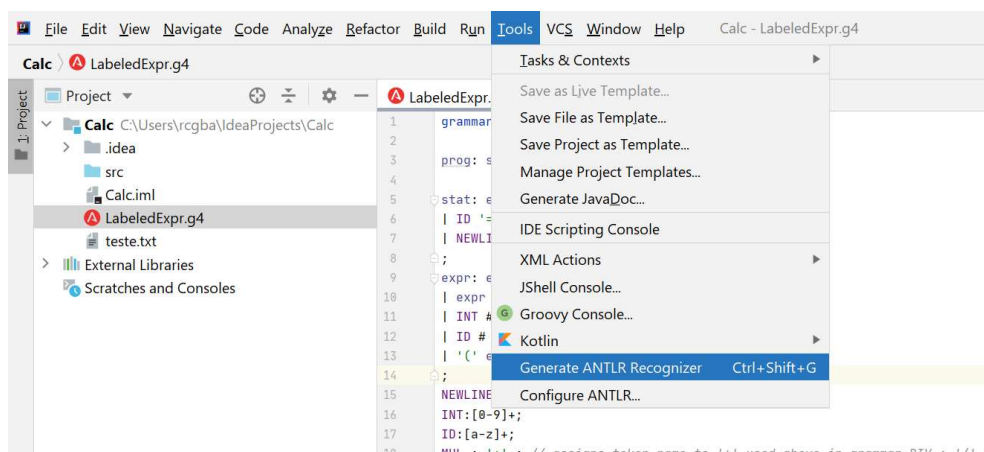
## 6. Gerar os ficheiros Java

Alterar o caminho para a pasta src: Tools -> Configure ANTLR

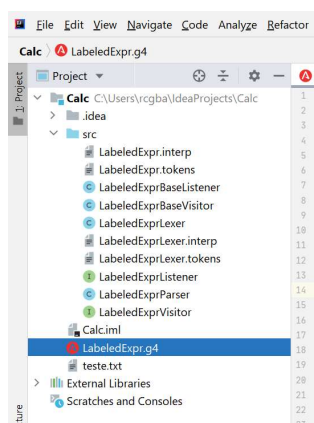




## Gerar o código: Tools -> Generate ANTLR Recognizer



## Ficheiros gerados



## 7. Criar outros ficheiros java para completar o projeto.

Criar ficheiro “Calc.java” e colocar o seguinte código :

---

```
import java.io.*;
import org.antlr.v4.runtime.*;
import org.antlr.v4.runtime.tree.*;

public class Calc {

    public static void main(String[] args) throws IOException {
        FileInputStream fis = new FileInputStream(new File("teste.txt"));
        LabeledExprLexer lexer = new LabeledExprLexer(new ANTLRInputStream(fis));
        CommonTokenStream tokens = new CommonTokenStream(lexer);
        LabeledExprParser parser = new LabeledExprParser(tokens);
        ParseTree tree = parser.prog(); // parse
        EvalVisitor eval = new EvalVisitor();
        eval.visit(tree);
    }
}
```

---

Criar ficheiro “EvalVisitor.java” e substituir o conteúdo por:

---

```
import java.util.HashMap;
import java.util.Map;

public class EvalVisitor extends LabeledExprBaseVisitor<Integer> {

    /** "memory" for our calculator; variable/value pairs go here */
    Map<String, Integer> memory = new HashMap<>();

    /**
     * ID '=' expr NEWLINE
     */
    @Override
    public Integer visitAssign(LabeledExprParser.AssignContext ctx) {
        String id = ctx.ID().getText(); // id is left-hand side of '='
        int value = visit(ctx.expr()); // compute value of expression on right
        memory.put(id, value); // store it in our memory
        System.out.println(id+'='+value); // print the result
        return value;
    }

    /** expr NEWLINE */
    @Override
    public Integer visitPrintExpr(LabeledExprParser.PrintExprContext ctx) {
        Integer value = visit(ctx.expr()); // evaluate the expr child
    }
}
```



```

        System.out.println(value); // print the result
        return 0; // return dummy value
    }

    /** INT */
    @Override
    public Integer visitInt(LabeledExprParser.IntContext ctx) {
        return Integer.valueOf(ctx.INT().getText());
    }

    /**
     * ID
     */
    @Override
    public Integer visitId(LabeledExprParser.IdContext ctx) {
        String id = ctx.ID().getText();
        if (memory.containsKey(id)) {
            return memory.get(id);
        }
        return 0;
    }

    /** expr op=('*' | '/') expr */
    @Override
    public Integer visitMulDiv(LabeledExprParser.MulDivContext ctx) {
        int left = visit(ctx.expr(0)); // get value of left subexpression
        int right = visit(ctx.expr(1)); // get value of right subexpression
        if (ctx.op.getType() == LabeledExprParser.MUL) {
            return left * right;
        }
        return left / right; // must be DIV
    }

    /** expr op=('+' | '-') expr */
    @Override
    public Integer visitAddSub(LabeledExprParser.AddSubContext ctx) {
        int left = visit(ctx.expr(0)); // get value of left subexpression
        int right = visit(ctx.expr(1)); // get value of right subexpression
        if (ctx.op.getType() == LabeledExprParser.ADD) {
            return left + right;
        }
        return left - right; // must be SUB
    }

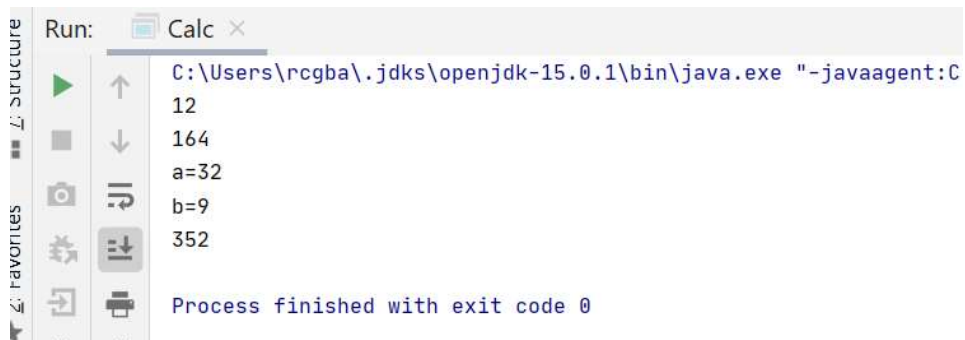
    /**
     * '(' expr ')'
     */
    @Override
    public Integer visitParens(LabeledExprParser.ParensContext ctx) {
        return visit(ctx.expr()); // return child expr's value
    }
}

```

---

## 8. Execução do programa completo

Da execução do programa deve resultar:



```
Run: Calc x
C:\Users\rcgba\.jdk\openjdk-15.0.1\bin\java.exe "-javaagent:C
12
164
a=32
b=9
352

Process finished with exit code 0
```