# 实验目的

通过⾃⼰编写、调试⼀个语法分析程序，对token序列进行语法分析。对于输入的每一个token进行查表分析，进而获得每一步的推导产生式。

# 内容描述

本程序是进行一个对token序列的语法分析。在程序里对产生式进行消除左递归，求非终结符的first和follow，生成LL（1）文法的预测分析表。生成预测分析表后，对输入的token序列进行查表分析。

# 思路方法

# 本人使用第三种实现方法：Generating programs based on your own YACC。

# 首先读入产生式文件，获得产生式；

# 查询产生式中是否存在直接左递归，使产生式改变成不含直接左递归的；

# 将无直接左递归和无二义性的产生式生成PPT（预测分析表）。中间穿插了各个非终结符的first和follow求算。

# 读取输入的token序列，进行查表分析。我这里构造的使LL（1）文法的预测分析表。故只需要一个推导过程中的符号栈即可。中间token序列分析出错会有错误提示。

# 5.将查表过程中使用到的推导式写入到结果文件中；如果是错误的token序列，将错误信息写入结果文件中。（控制台也会打印相关信息）。

# 假设

假设输入的产生式都是无二义性的。

# 相关FA

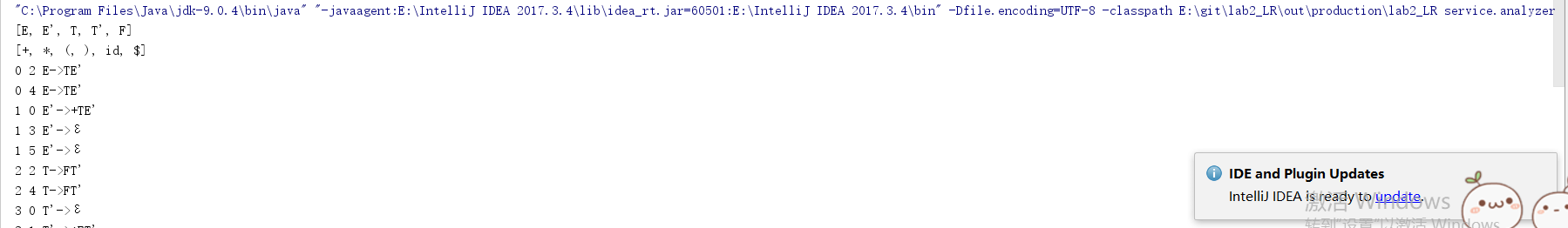
预测分析表：

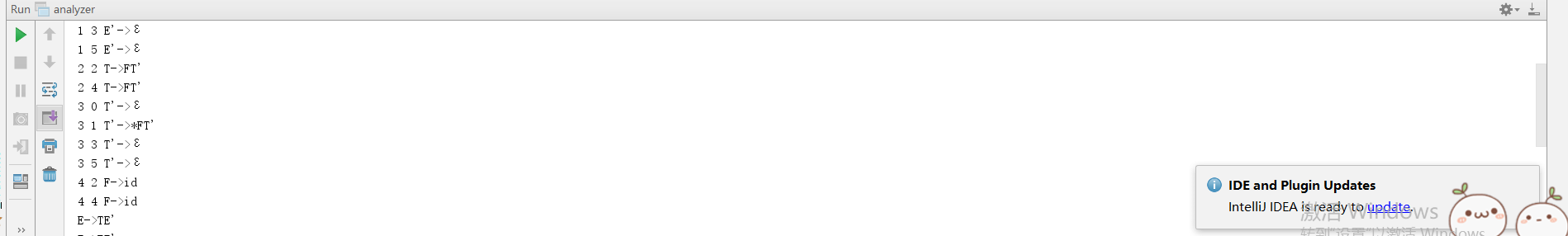
|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | + | \* | （ | ） | id | $ |
| E |  |  | E ->TE’ |  | E ->TE’ |  |
| E’ | E'->+TE |  |  | E'->ε |  | E'->ε |
| T |  |  | T->FT' |  | T->FT' |  |
| T’ | T'->ε | T'->\*FT' |  | T'->ε |  | T'->ε |
| F |  |  | F->id |  | F->id |  |

程序中分析产生式后，得到的预测分析表打印如下：

前两行分别是对应非终结符和终结符集合；

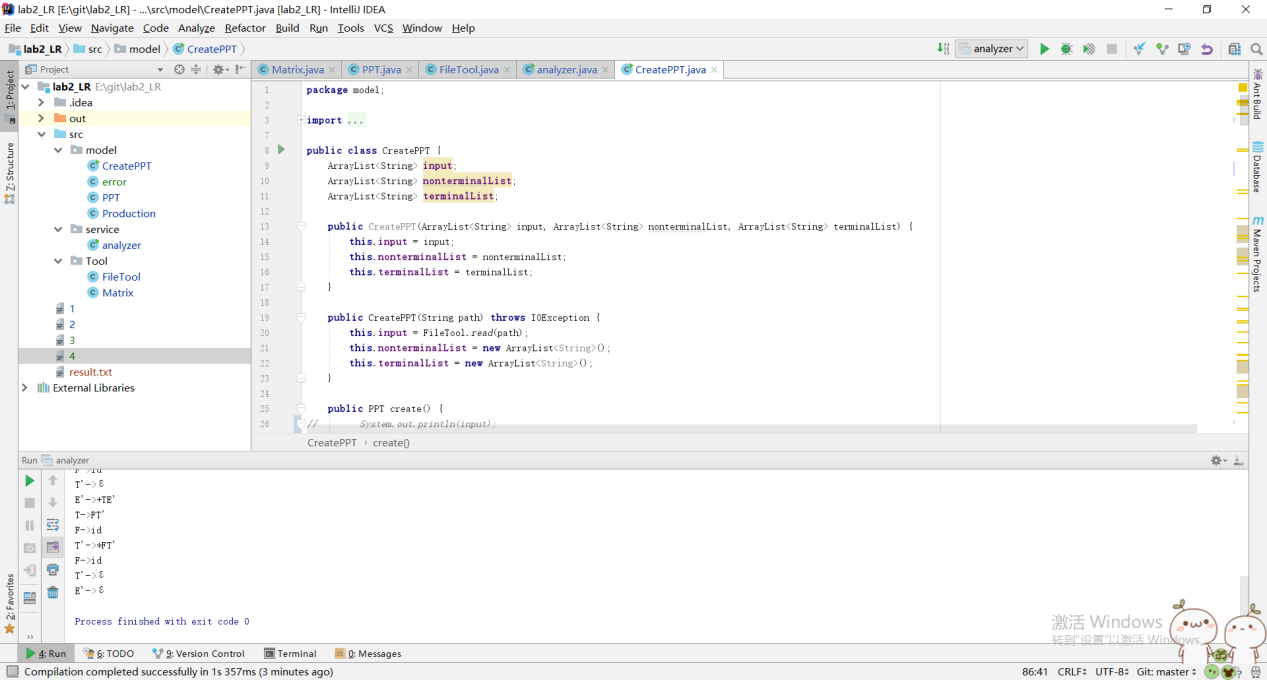
后面的每行，前面两个数字对应产生式对应在预测分析表矩阵中的位置。与上面的表格完全对应。



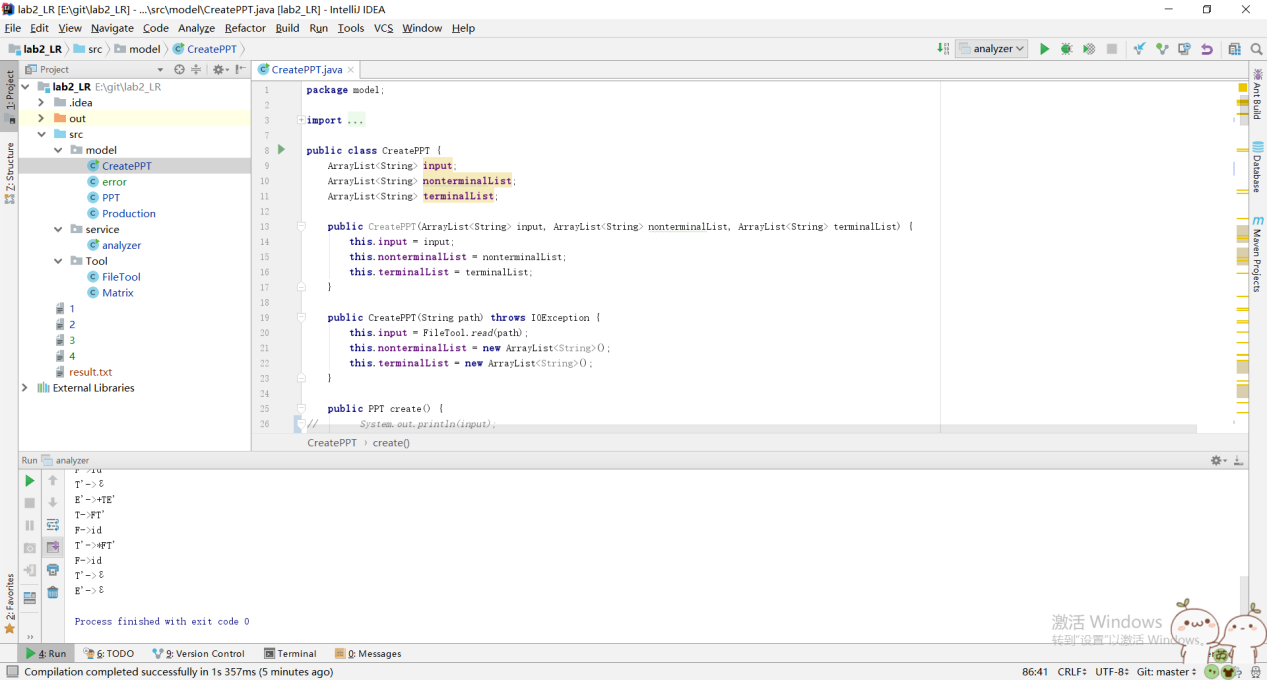


# 重要数据结构描述

总共有7个类。



CreatePPT类：



这个类的主要功能是根据输入的产生式来生成预测分析表。

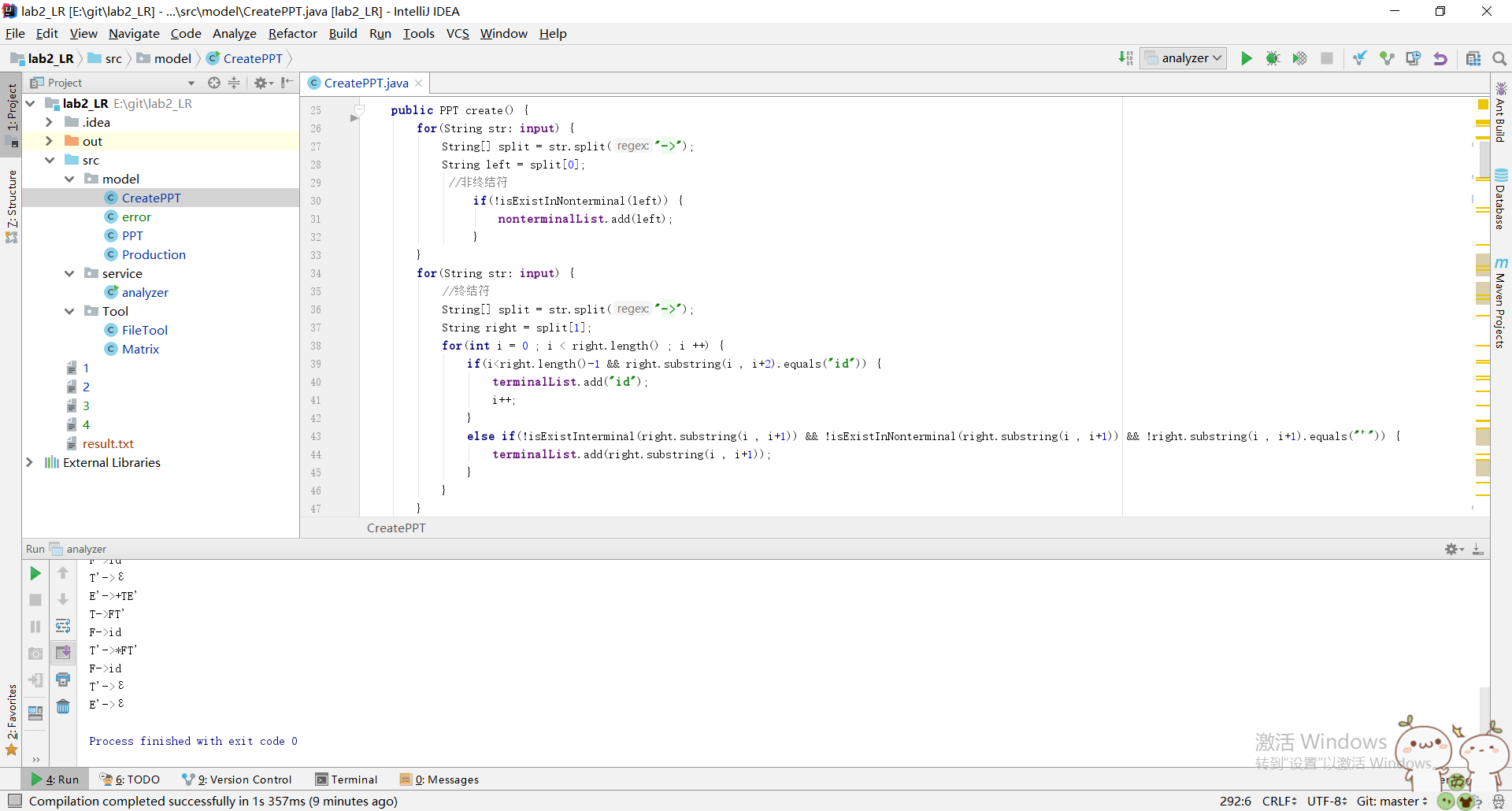
类属性：input是输入的产生式集合；

nonterminalList是非终结符元素的集合。

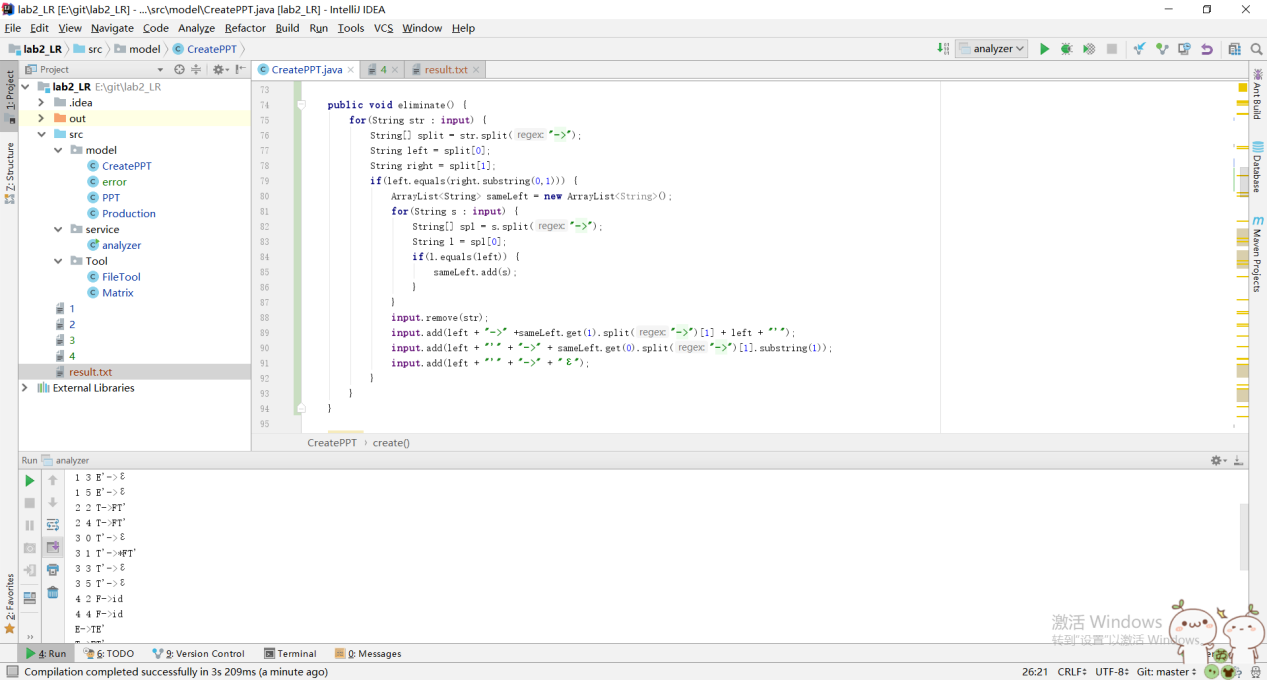
terminalList是终结符元素的集合。

主要方法说明：

create方法：是产生ppt预测分析表。



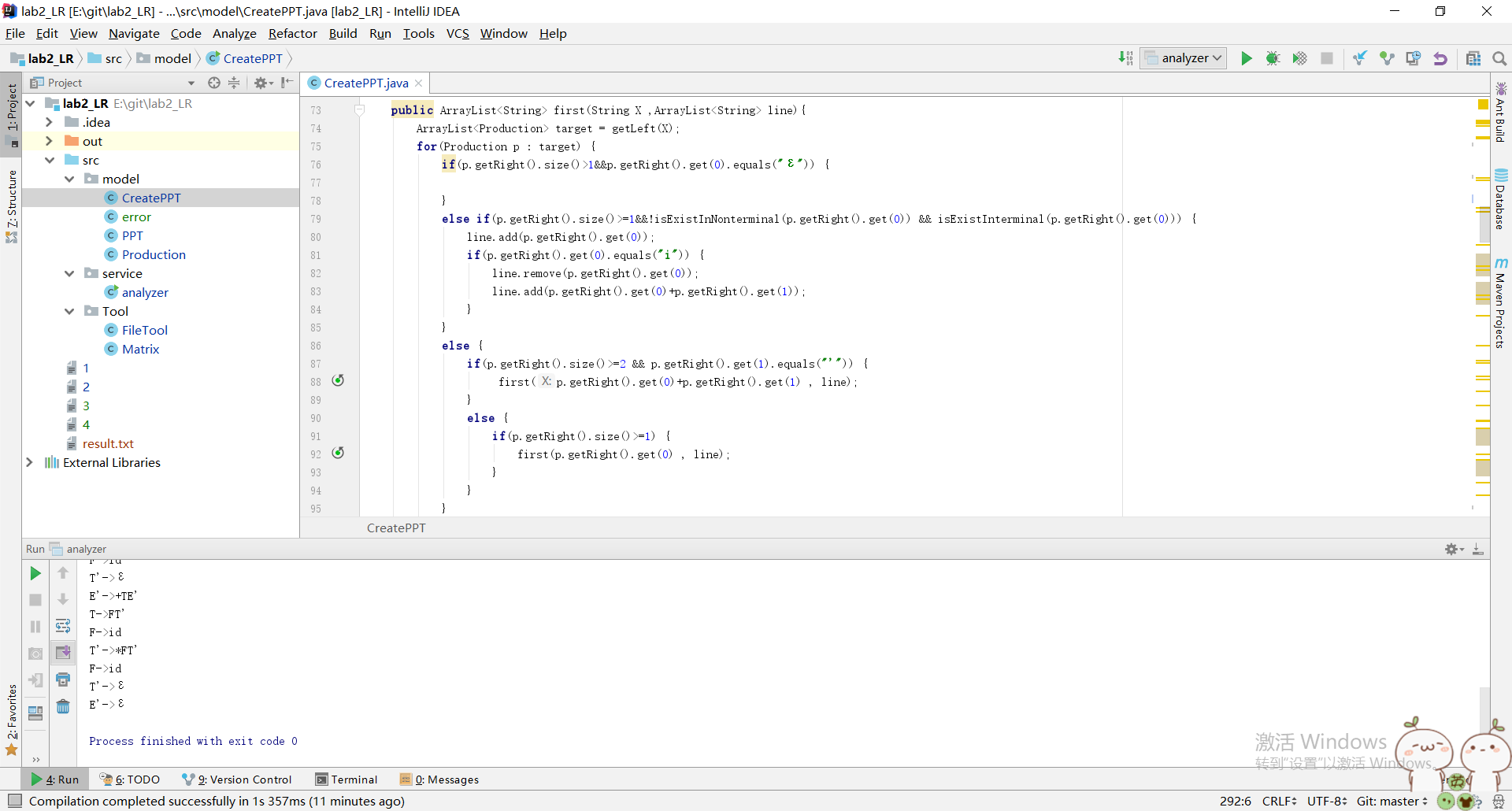
预处理eliminate方法：消除左递归。将含有直接左递归的产生式拆解成不含直接左递归。



first方法：计算各个非终结符的first值，为生成ppt作准备；

采用了递归。

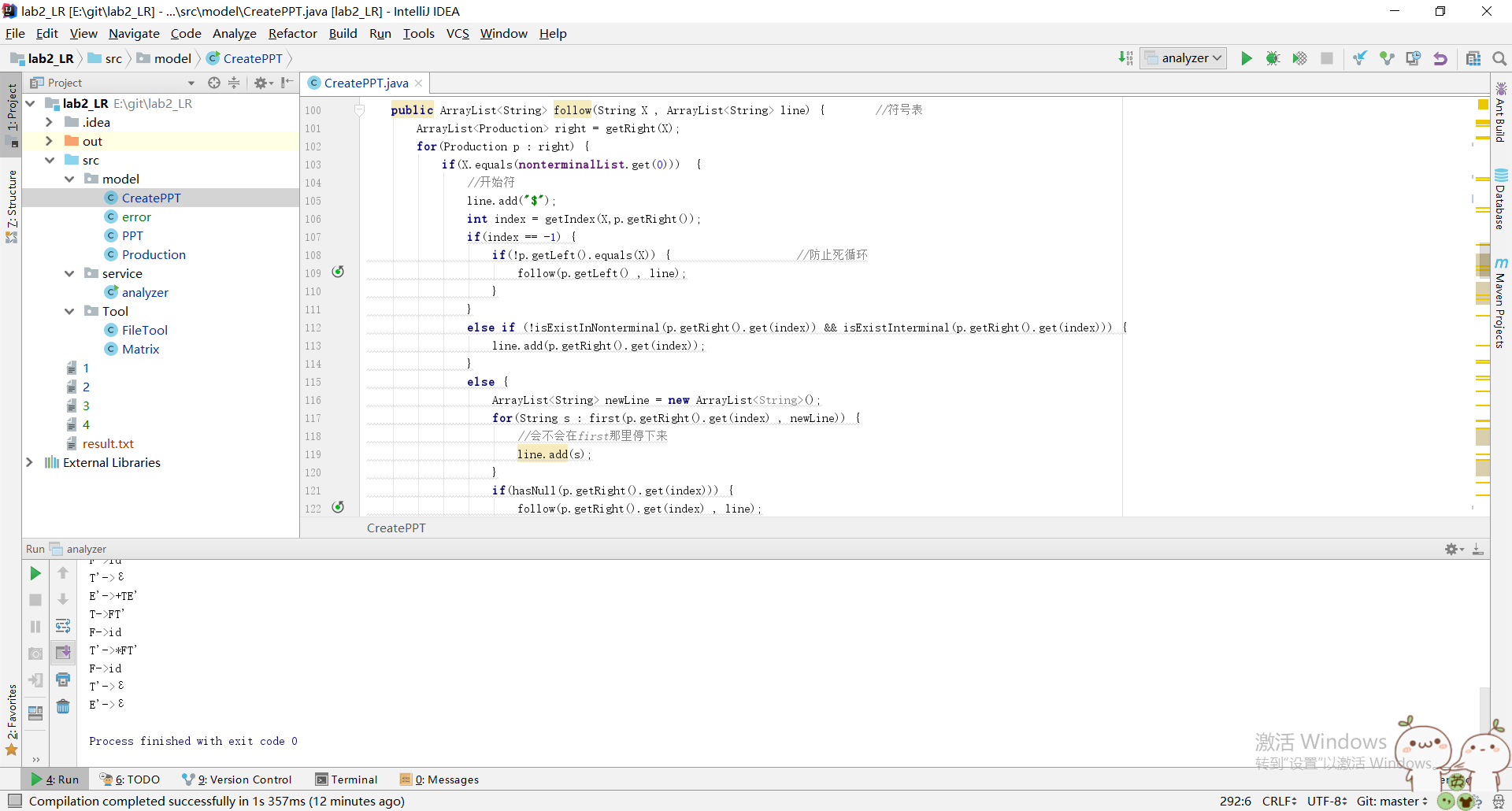
参数是非终结符和存取值的list。



follow方法：计算各个非终结符的follow值，为生成ppt作准备；

采用了递归以及调用了first函数。

参数是非终结符和存取值的list。

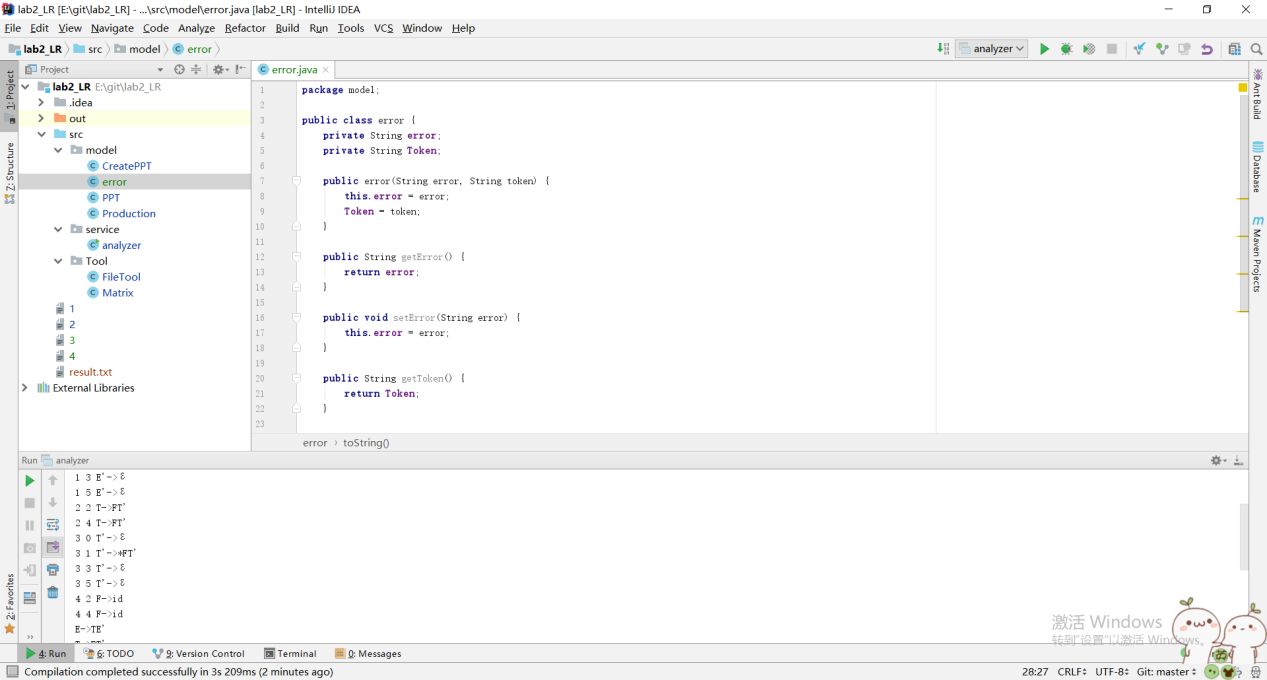


其他两百行左右的代码是为了辅助主要方法而写的辅助方法。

error类：

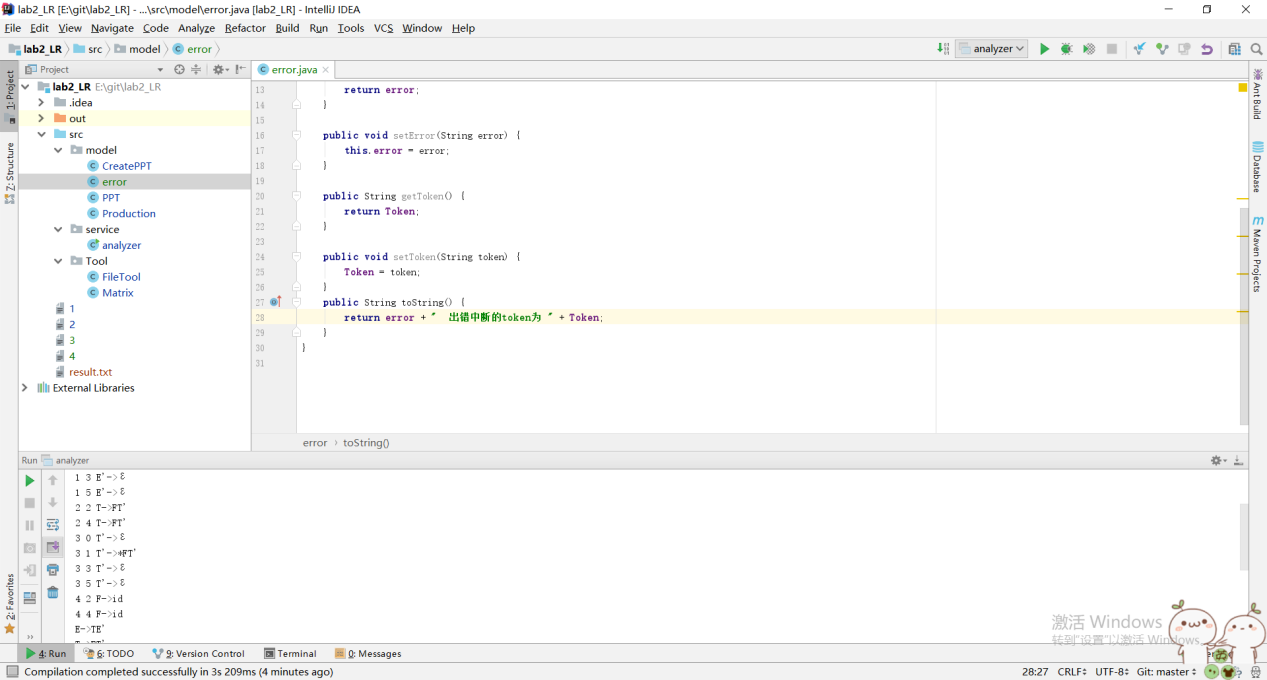
类属性：error 报错提示

Token 引起语法分析中断的token



主要方法分析：

重写toString的方法，将error转为字符串。

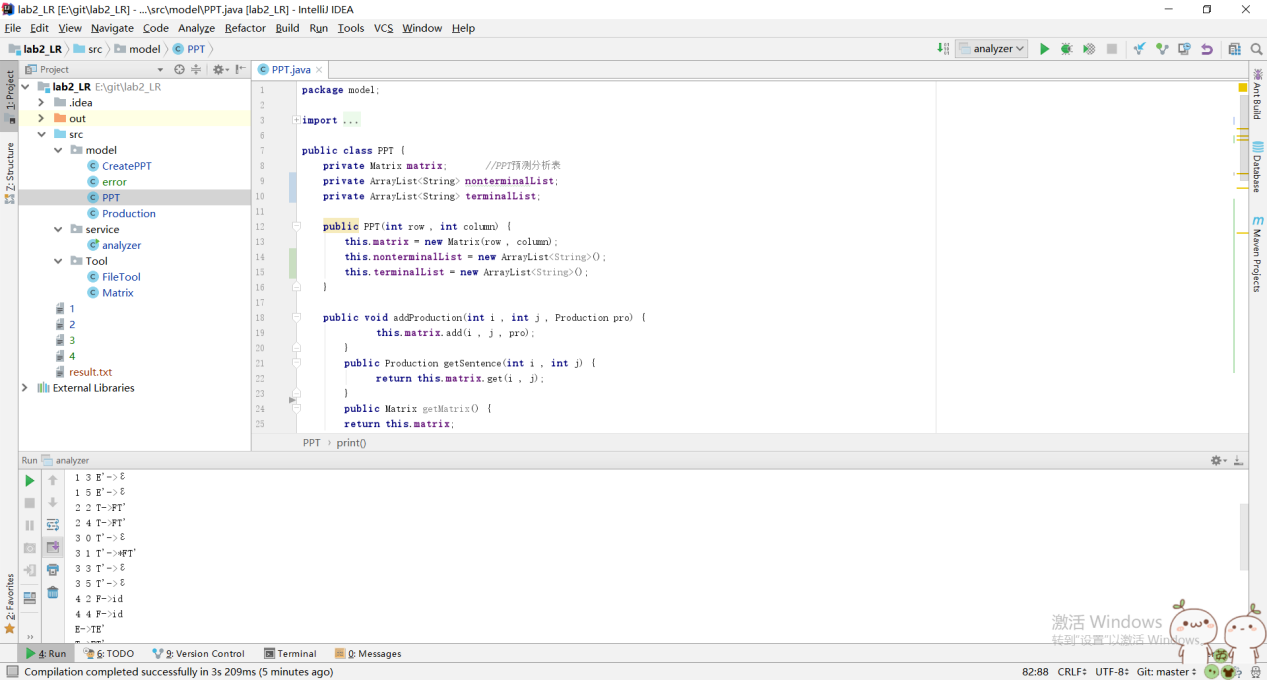


PPT类

类属性：matrix 存放预测分析表数据的矩阵；

nonterminalList 非终结符集合

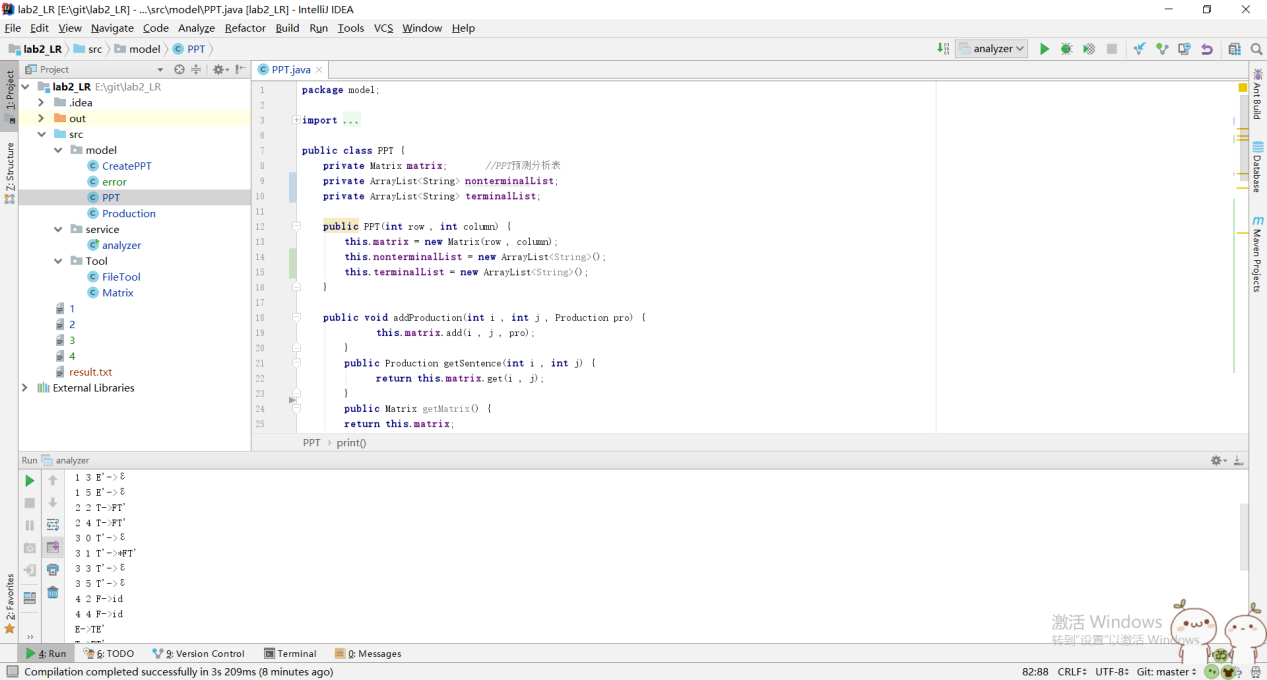
terminalList 终结符集合



主要方法：

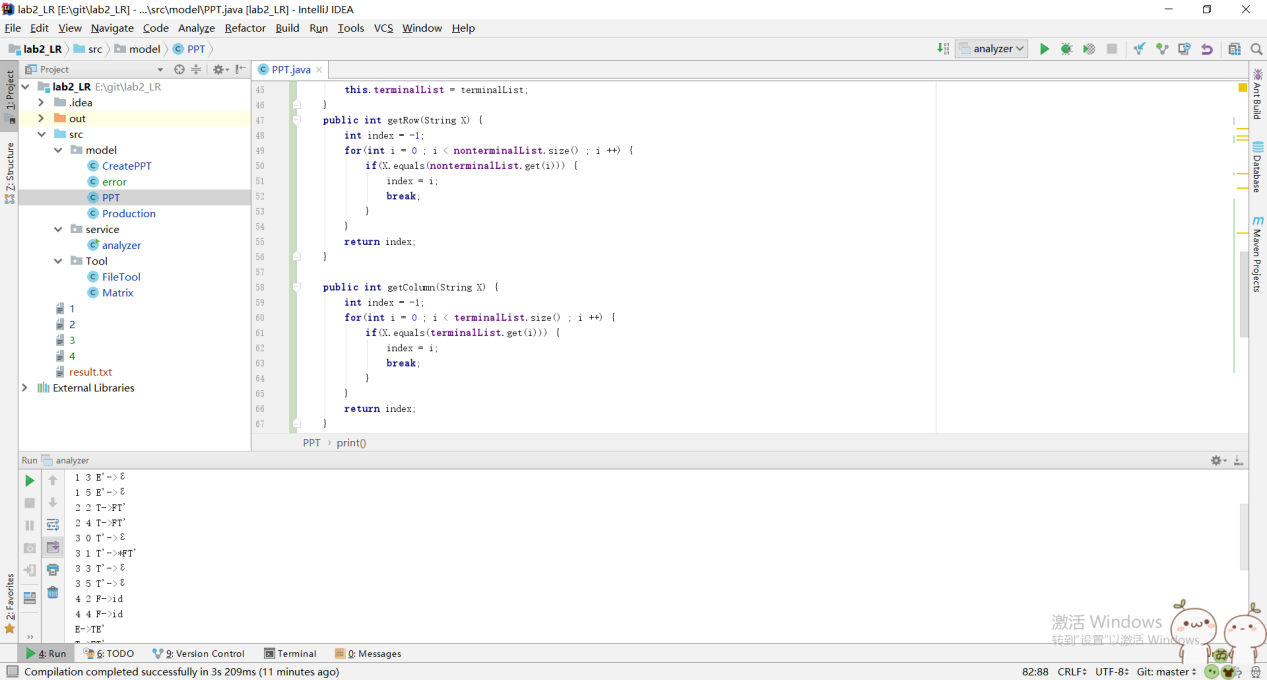
（鉴于方法比较简单，简单描述一下。）

addProduction ：向语法分析表中加入产生式



getRow：获取对应非终结符在预测分析表中的行。

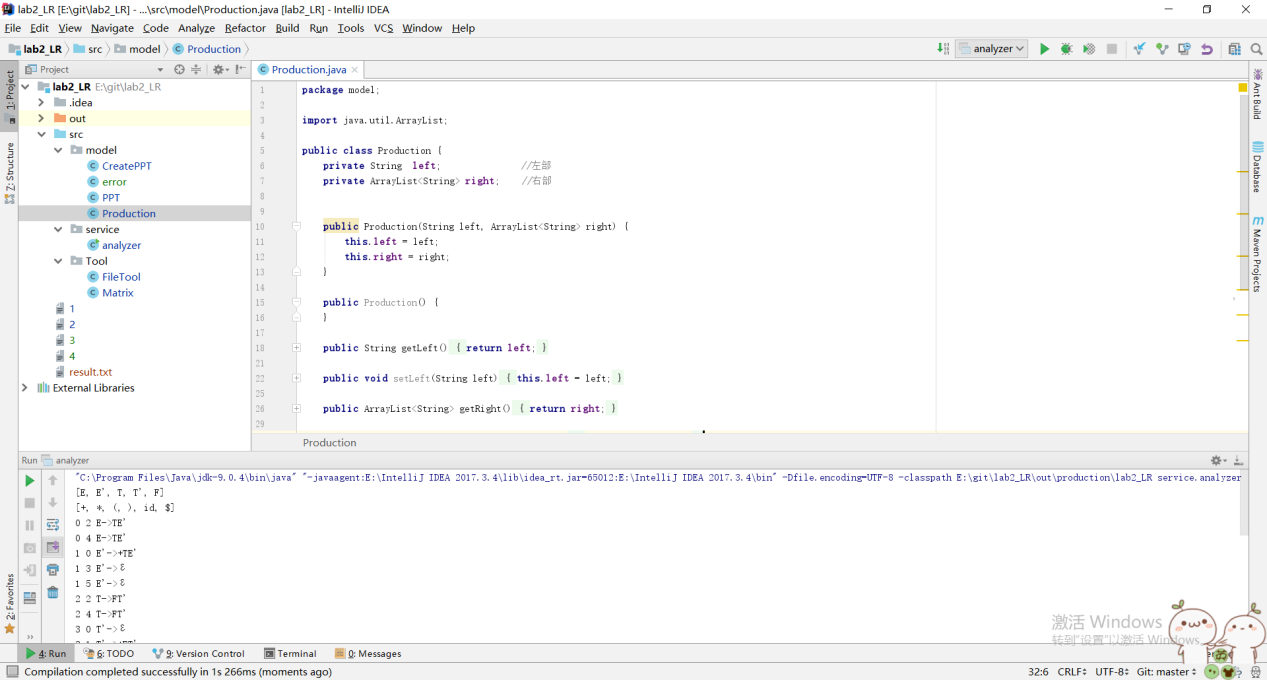
getColumn：获取对应终结符在预测分析表中的列。



Production类：

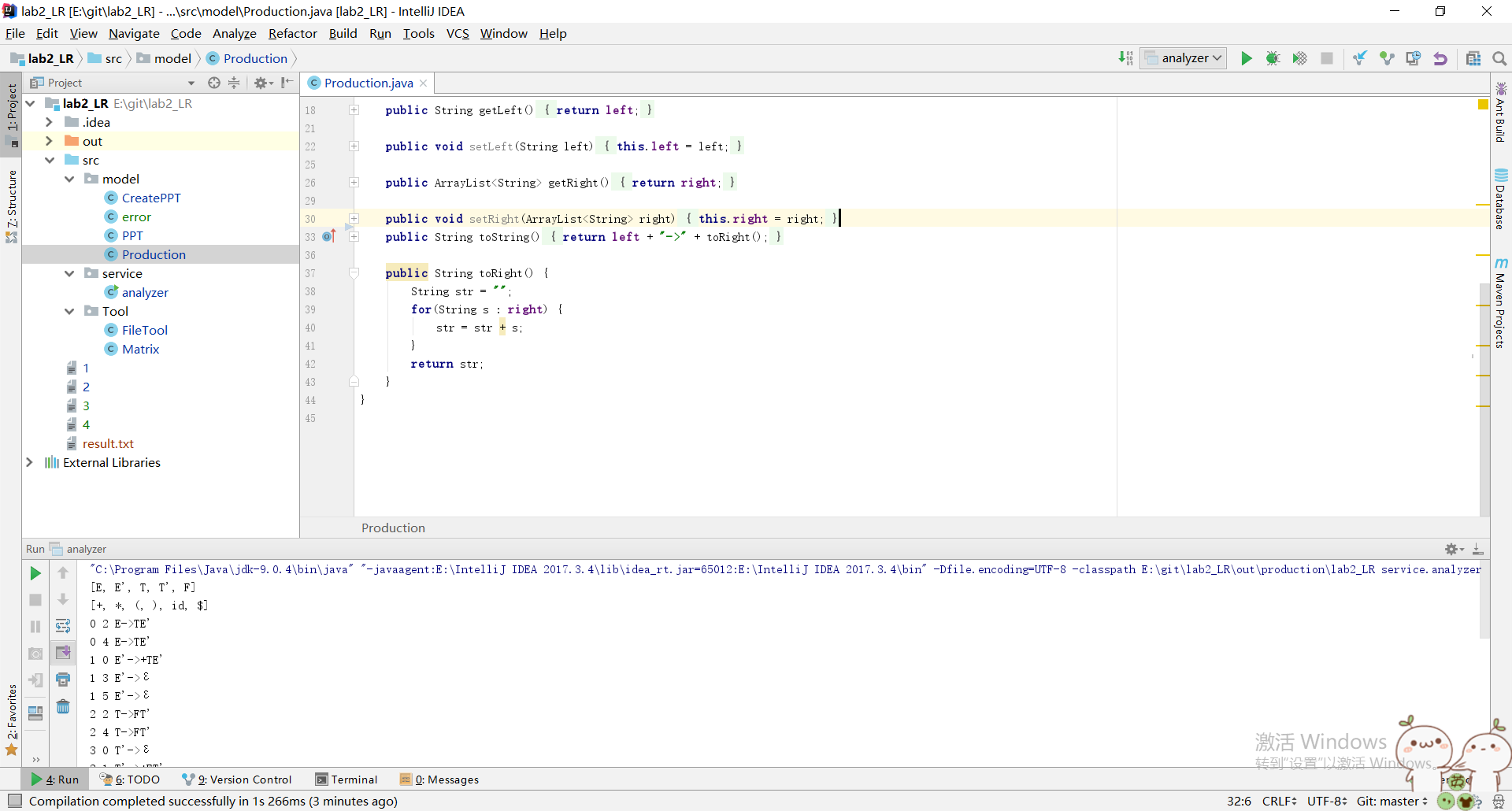
类属性：left 左部。

Right 右部。



主要方法：

toString 重写toString方法，将Production转为String类型。内部实现调用了toRight方法（将右部转为String）。



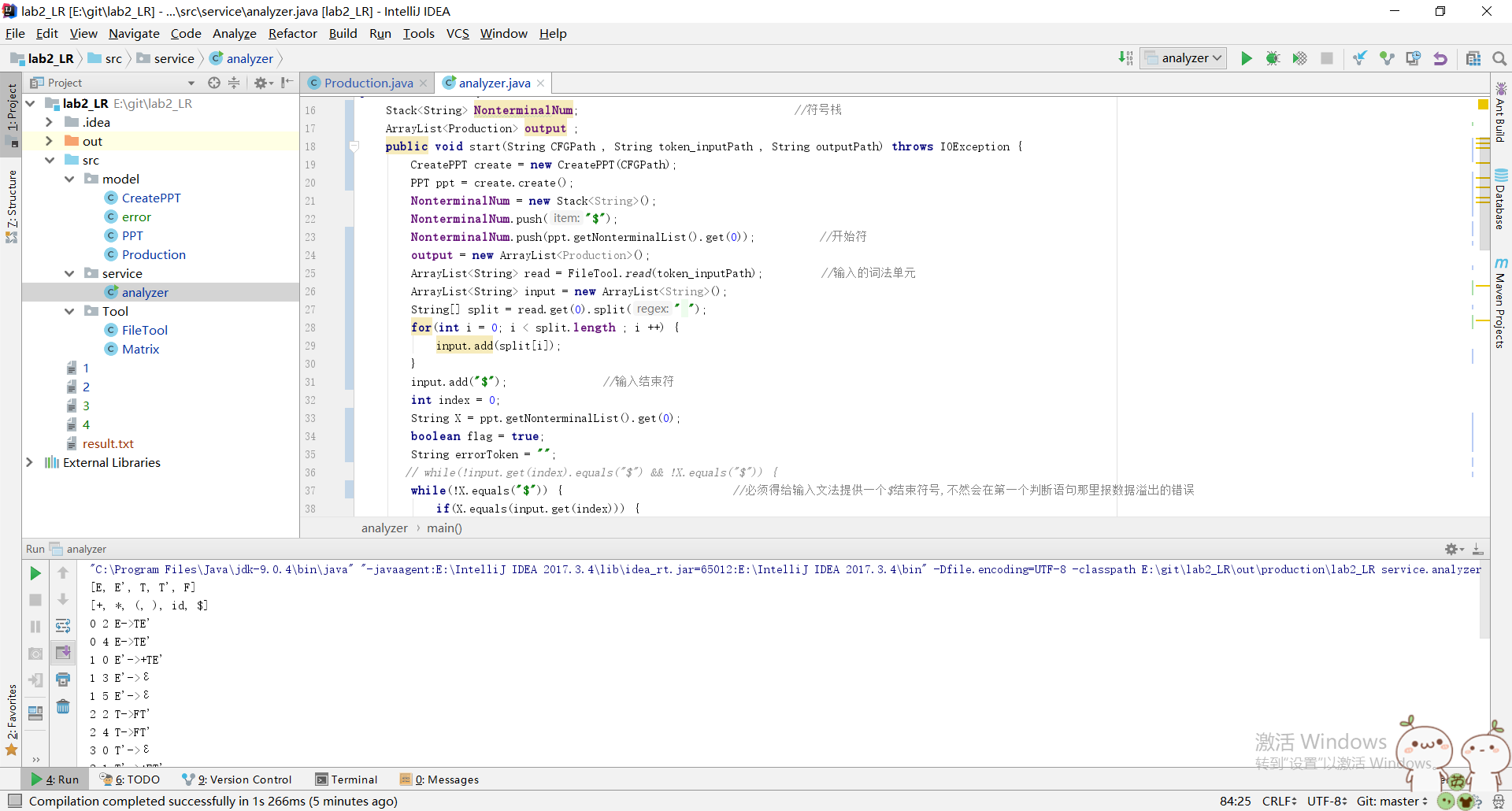
analyzer类 （语法分析类）

类属性：NonterminalNum 符号栈。

Output 输出的推导式。

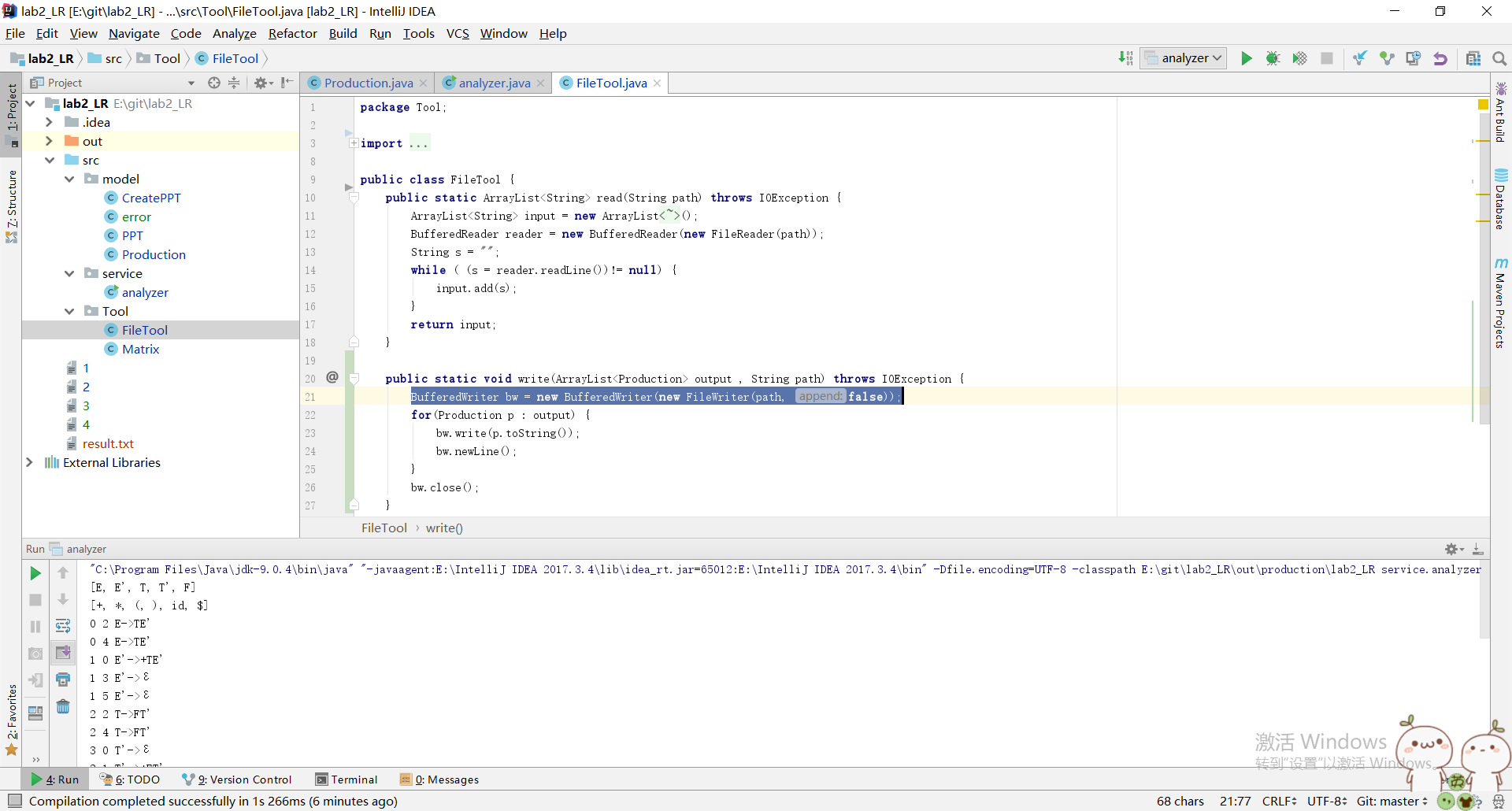
主要方法：start函数

参数分别为CFG的文件路径，输入的token序列文件路径，以及输出文件的路径。



FileTool类

主要功能是文件的读和写

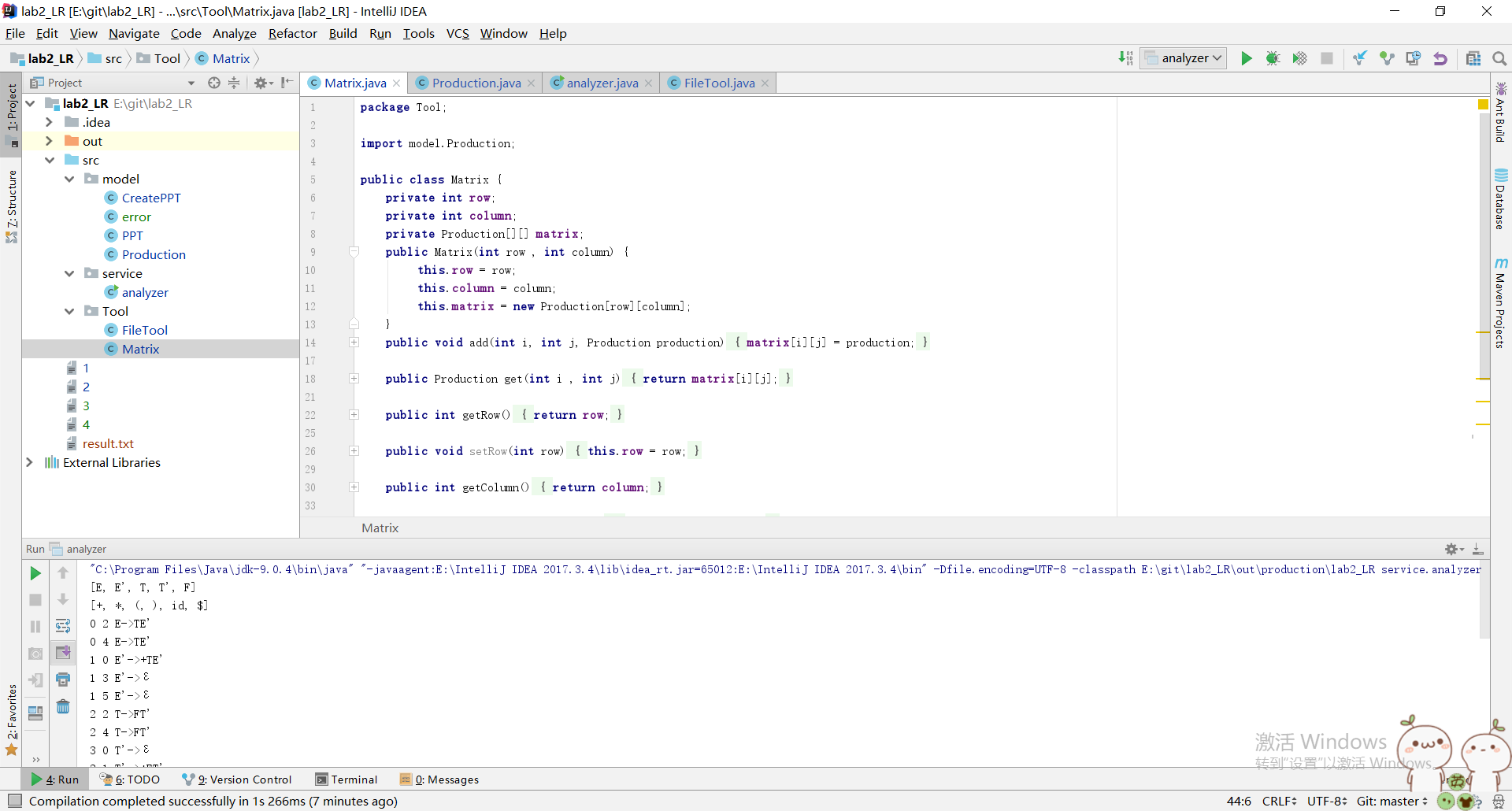


Matrix类：

矩阵类：辅助存储预测分析表的工具类

类属性分别是行，列和二维数组

方法就是常见的矩阵读写的方法。

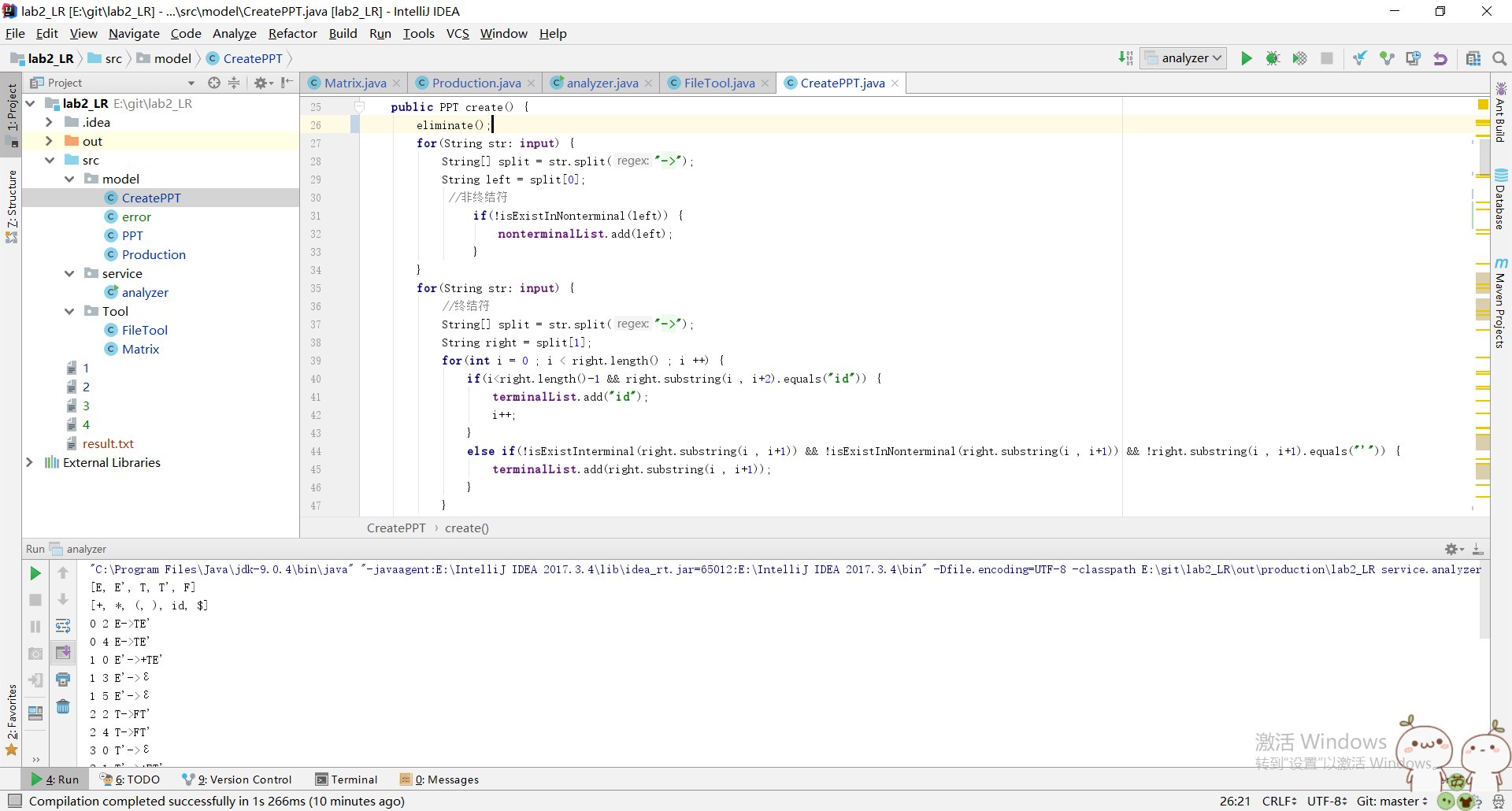


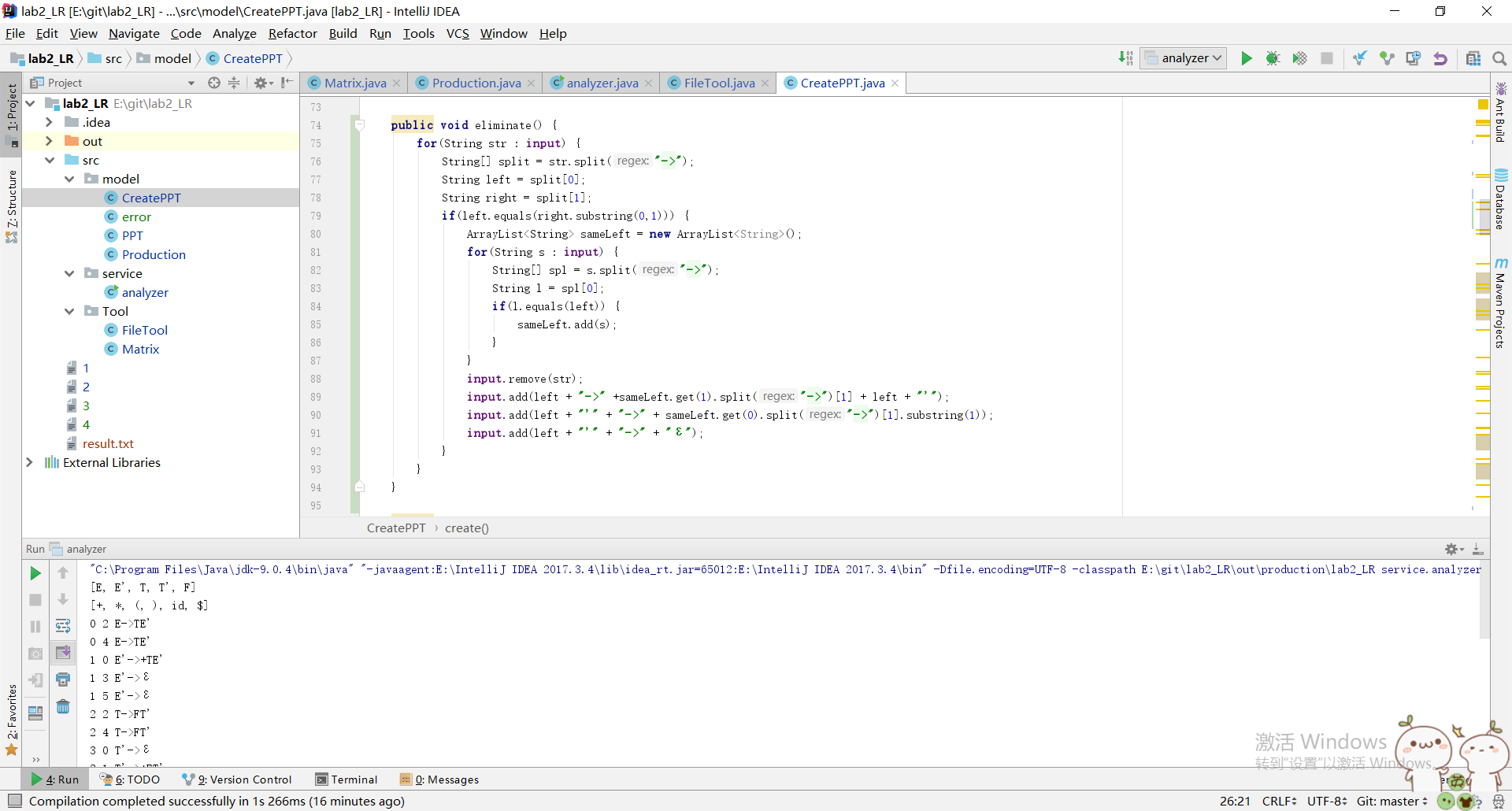
# 核心算法

**主要分为两块。**

1. **生成预测分析表PPT； （CreatePPT类里实现）**

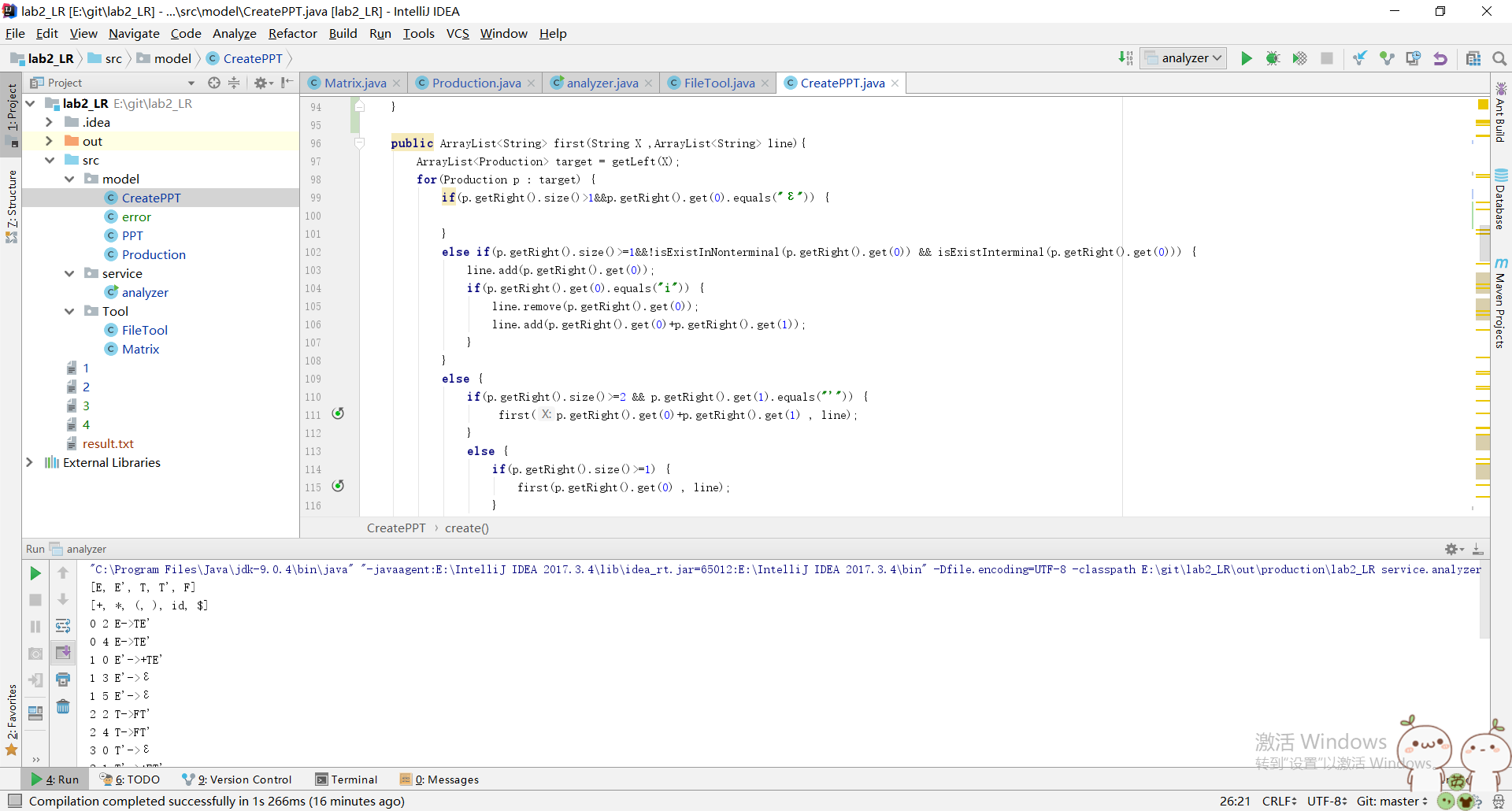
**按照龙书上讲的步骤，首先我们得对这些产生式进行预处理；于是我先调用了eliminate方法来对得到的input进行直接左递归的消除处理。大致思想就是先遍历全部input，对于左部和右部首个符号相同的产生式，即为直接左递归。找到直接左递归的产生式后，得到与之左部相同的所有产生式。通过添加新的左部，将直接左递归拆解。**



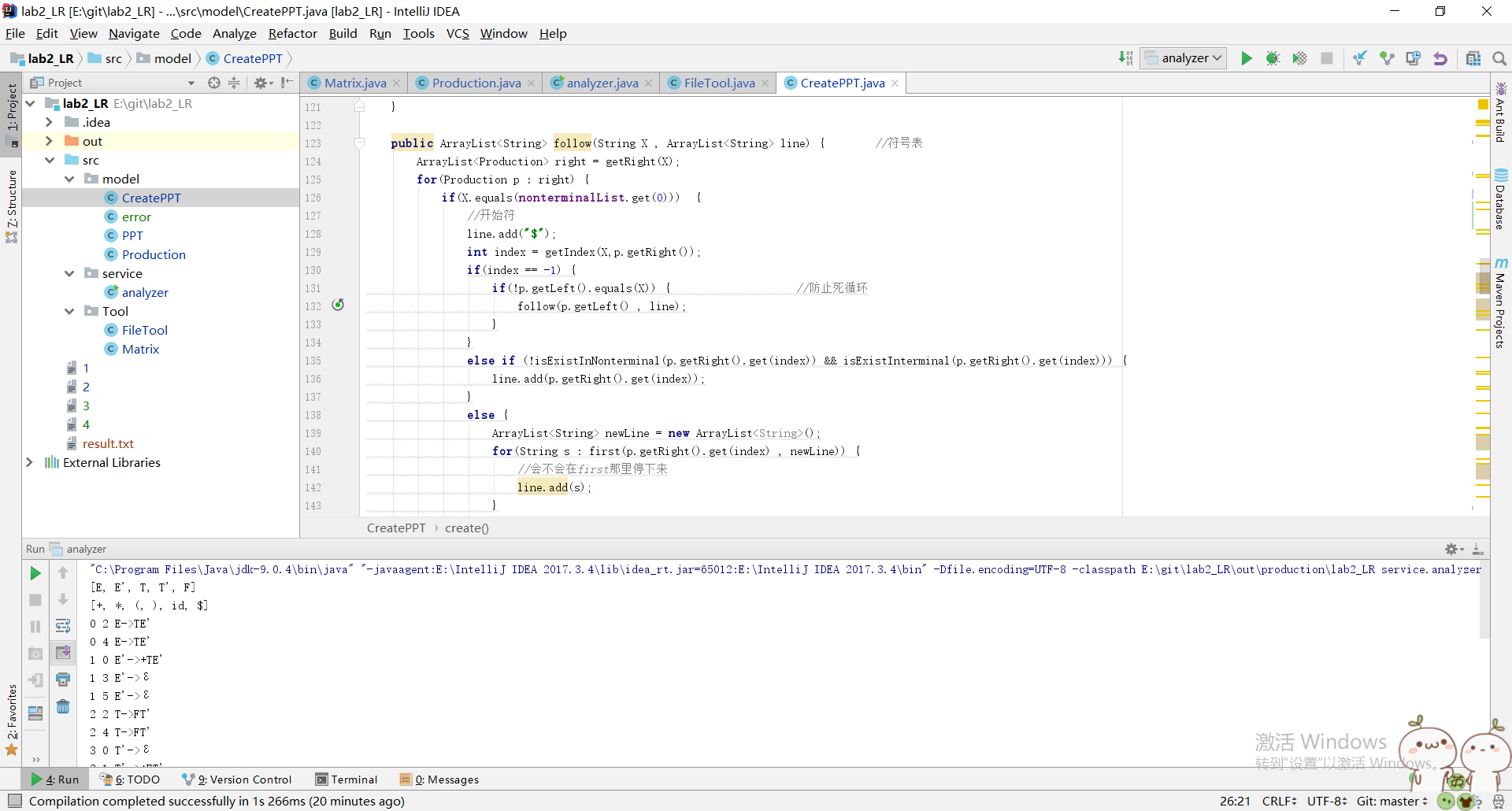


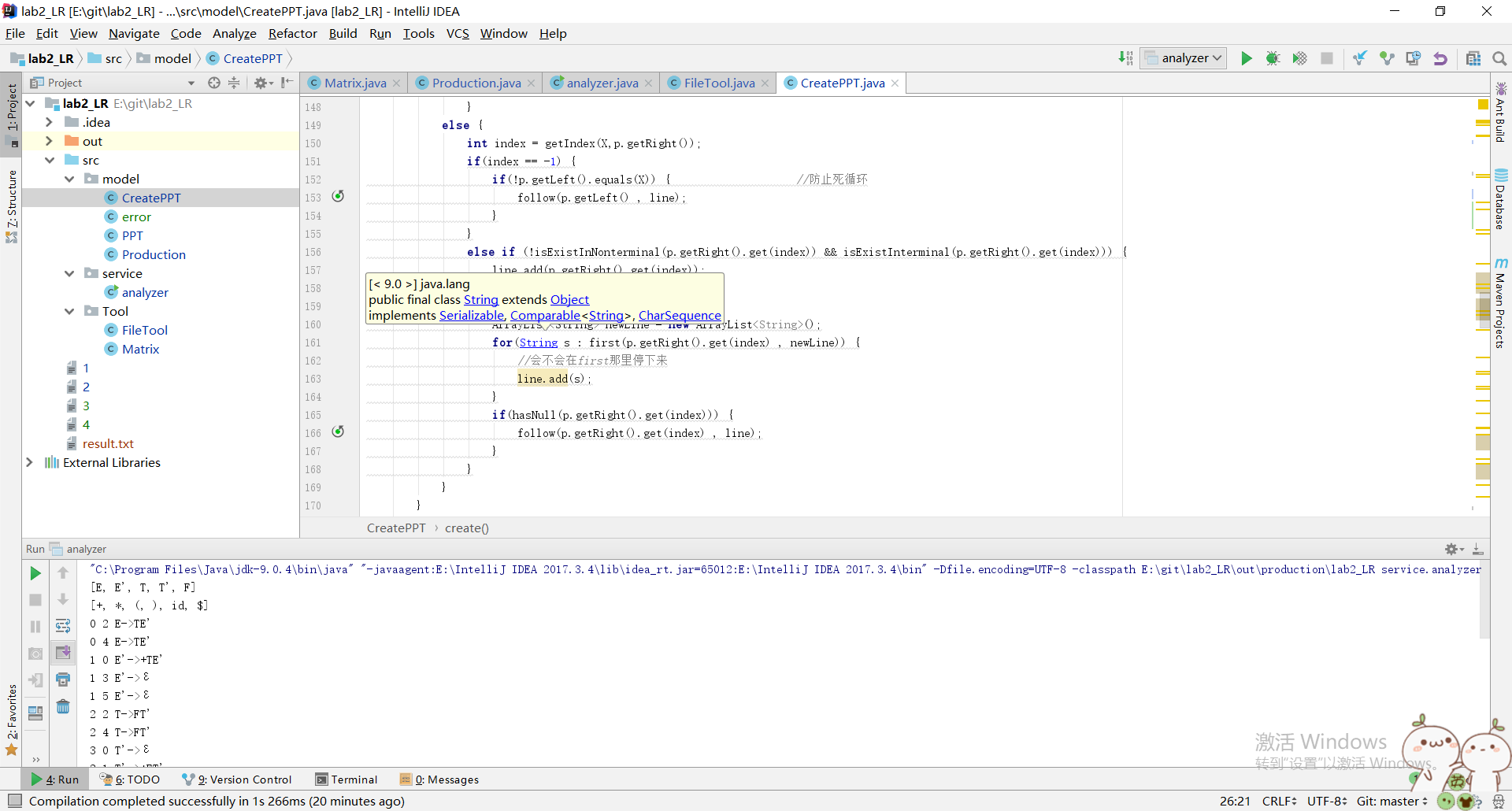
**然后我们就是需要求解每一个非终结符的first和follow了，对于first函数，只需要用到递归，就可轻松实现。循环里是该非终结符在产生式左部的所有产生式。**

**如果右部是空串，跳过；如果右部的首位是终结符，就直接将它添加进去。如果右部是非终结符，就用first递归。**



**对于follow函数，用到递归以及调用first函数。循环里是该非终结符在产生式右部的产生式。如果是开始符，加入$;如果该非终结符是最后一位，并且左部和它不相同（会死循环），递归follow。如果紧跟着的是终结符，加入它。如果紧跟着的是非终结符，首先调用first函数，将紧跟非终结符的first加进去，再检查该非终结符是否会推出空串，如果会的话，则递归调用follow。**





1. **得到预测分析表后进行语法分析； （analyzer类里实现）**

**相比较于生成预测分析表而言，进行语法分析并不是特别复杂。**

**首先获得输入的token序列，并在末尾加上“$”的结束符号。在符号栈里先压入“$”和开始符。这样初始状态就已经好了；**

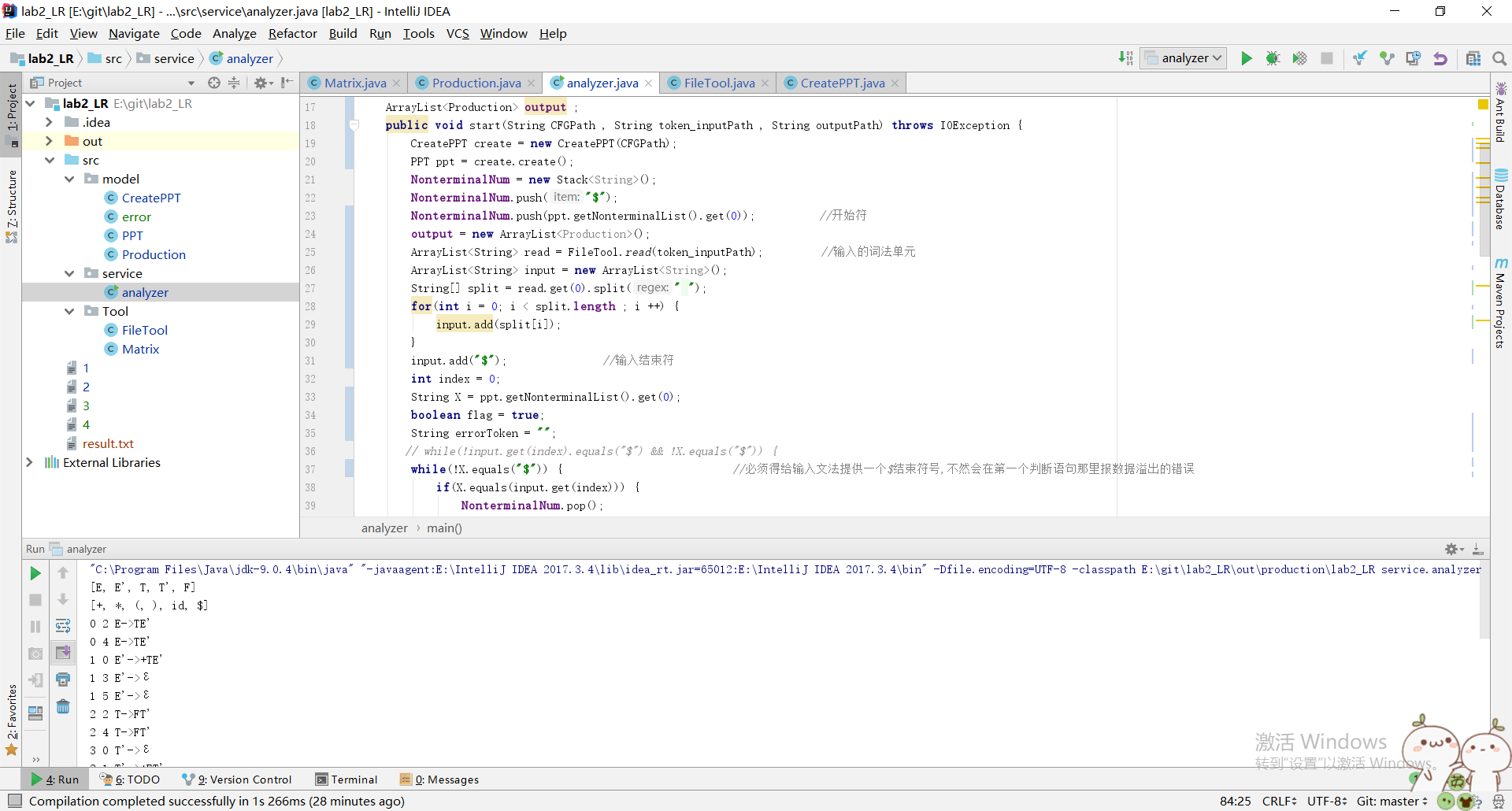
**然后当栈非空，如果栈顶符号等于token序列所指符号；出栈，token序列指针后移。**

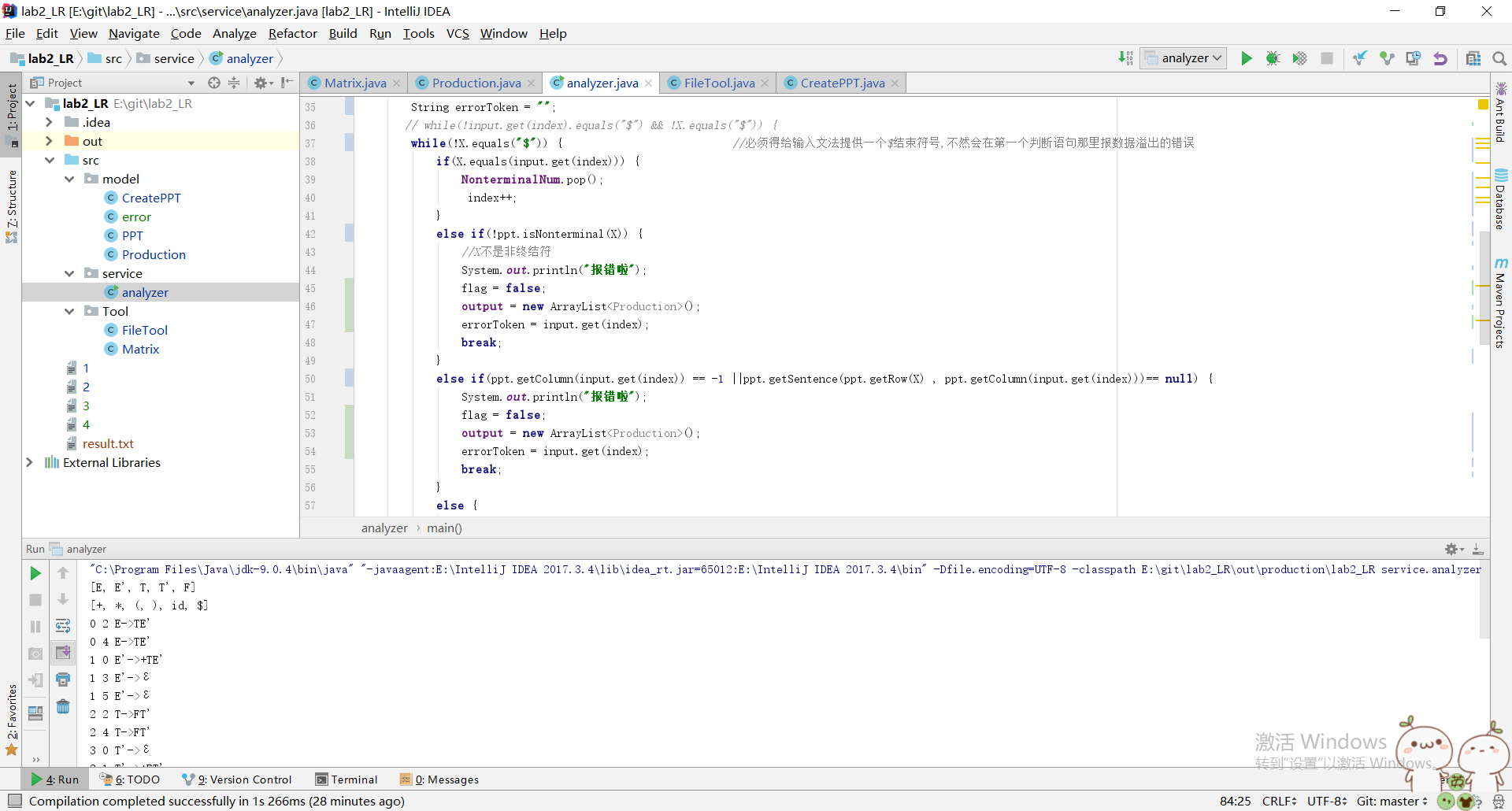
**如果栈顶符号是一个终结符（即token序列不匹配），报错。**

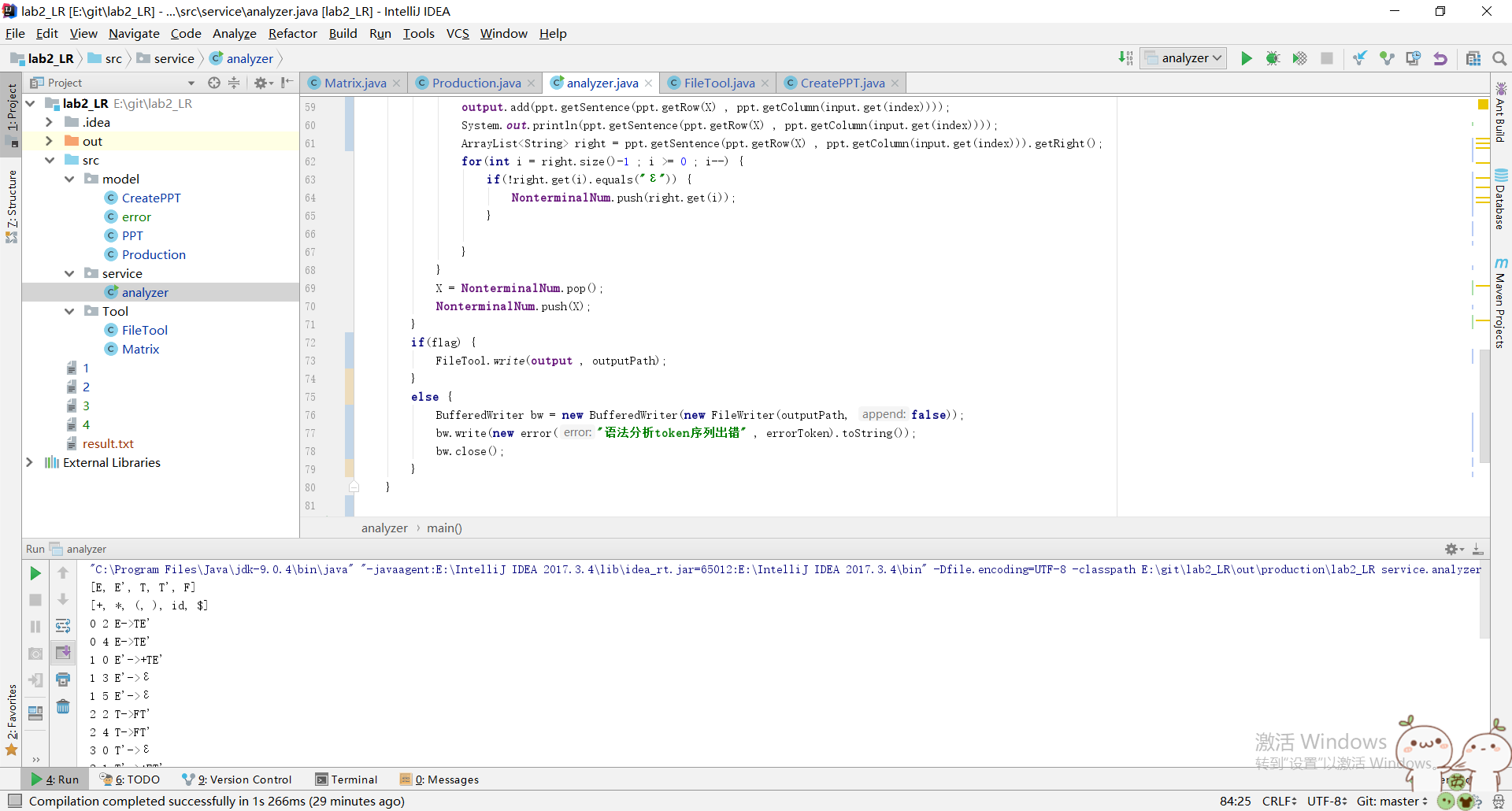
**如果预测分析表内对应的产生式为空 ， 报错**

**如果预测分析表内对应的产生式不为空 ， 弹出栈顶符号，并将推导产生式的右部按照倒序地顺序压栈。**

**依次循环，直到栈为空。**

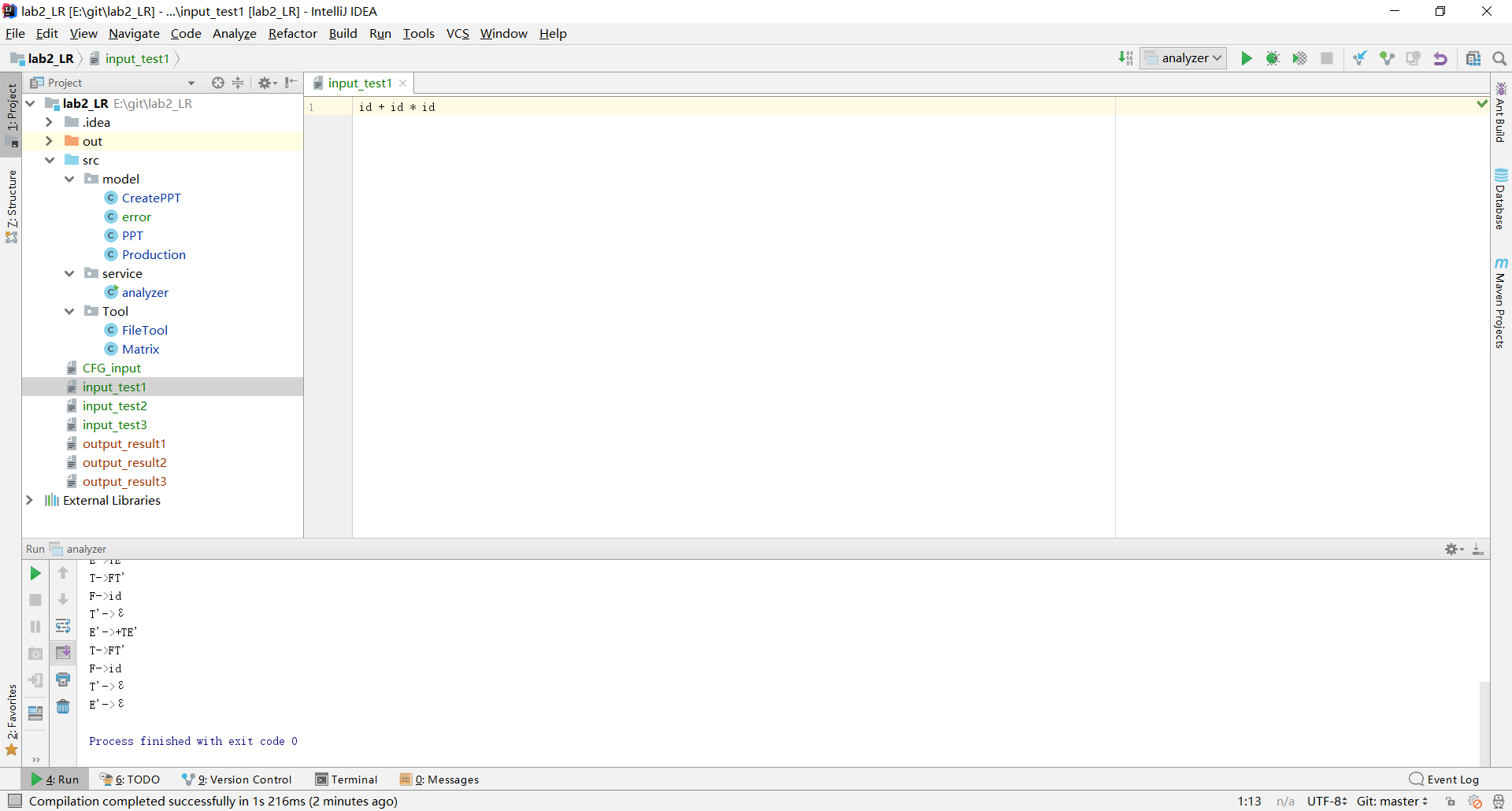




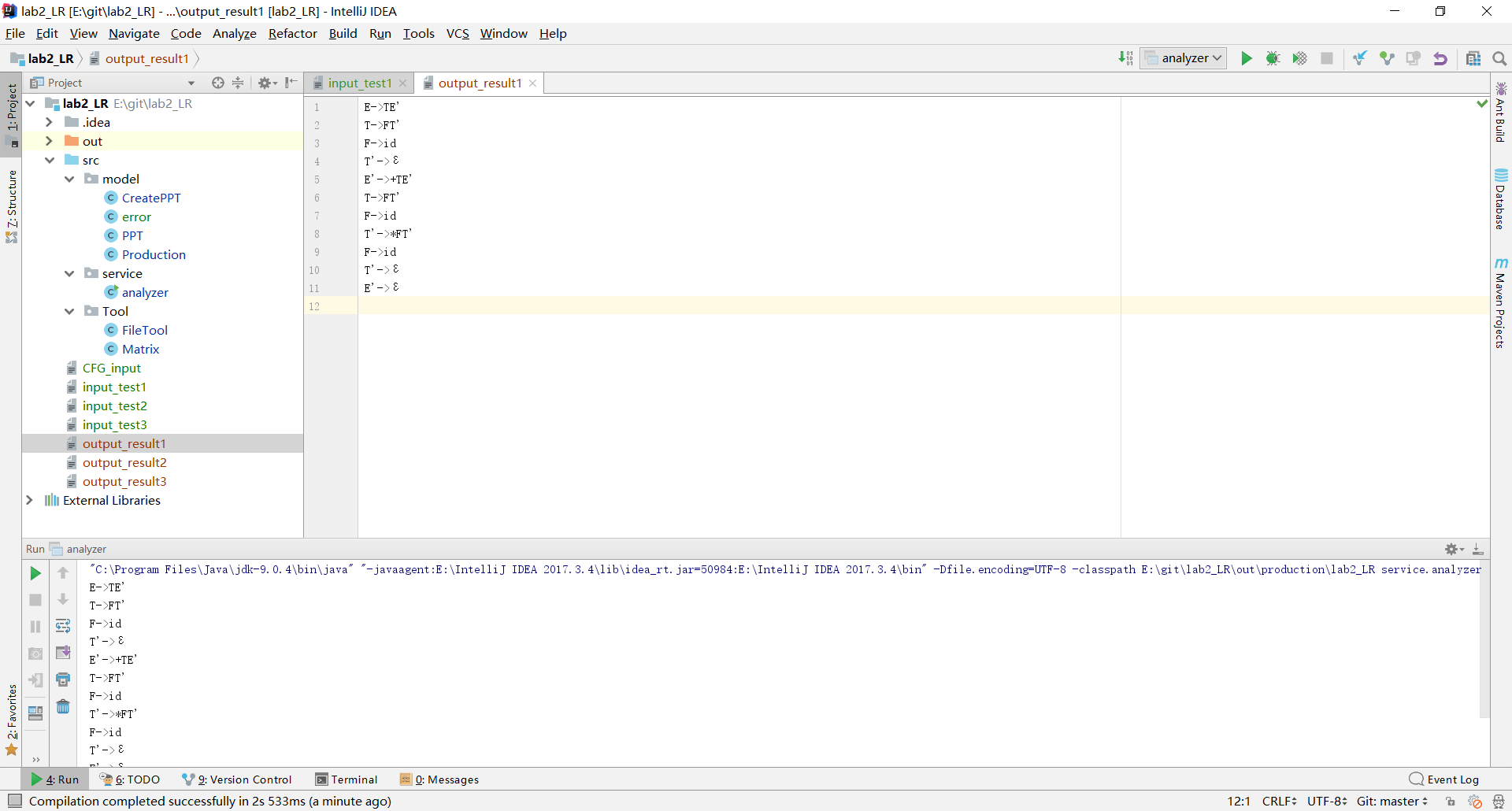


# 运行截图

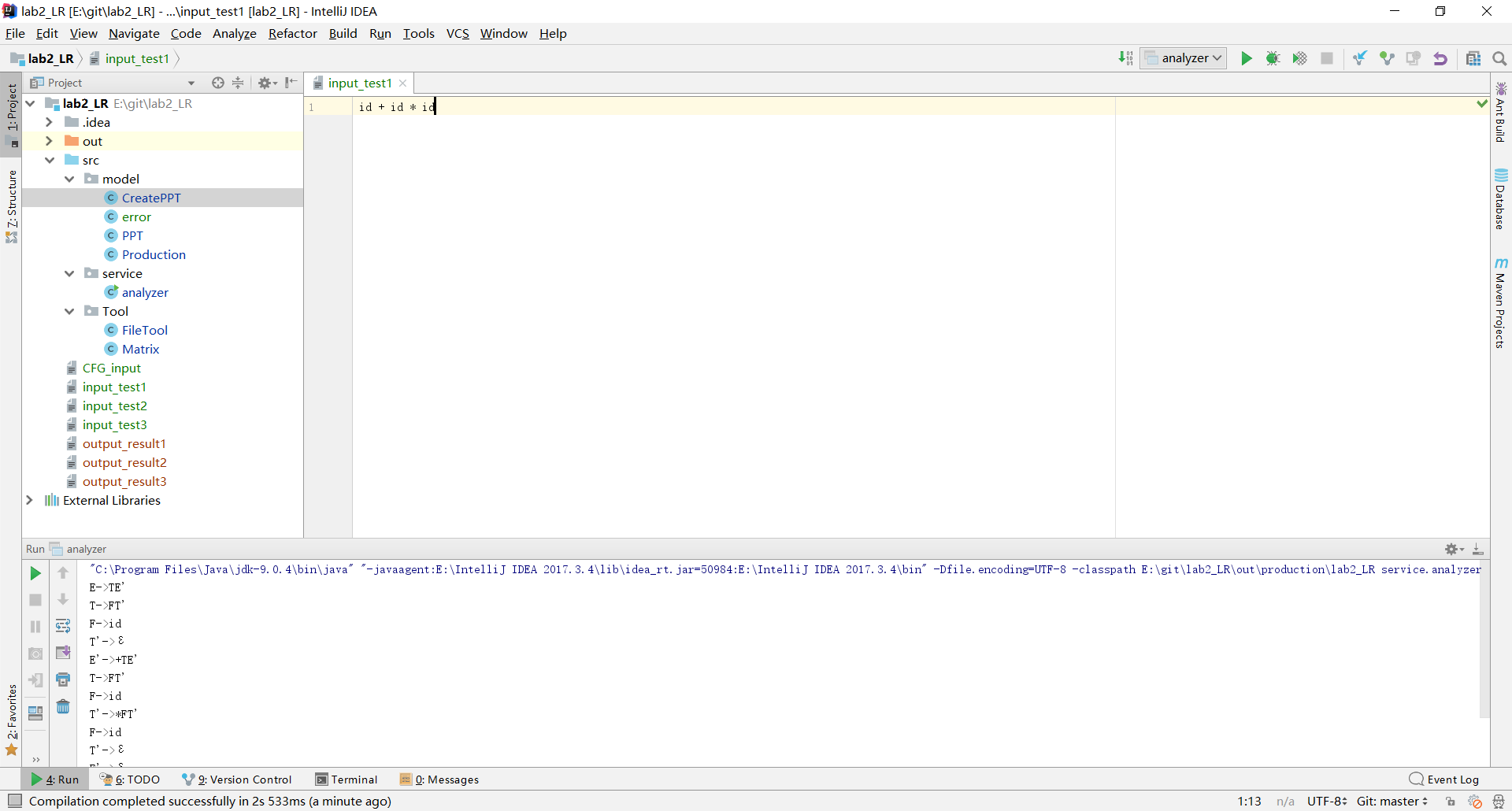
Input\_test1输入文件：



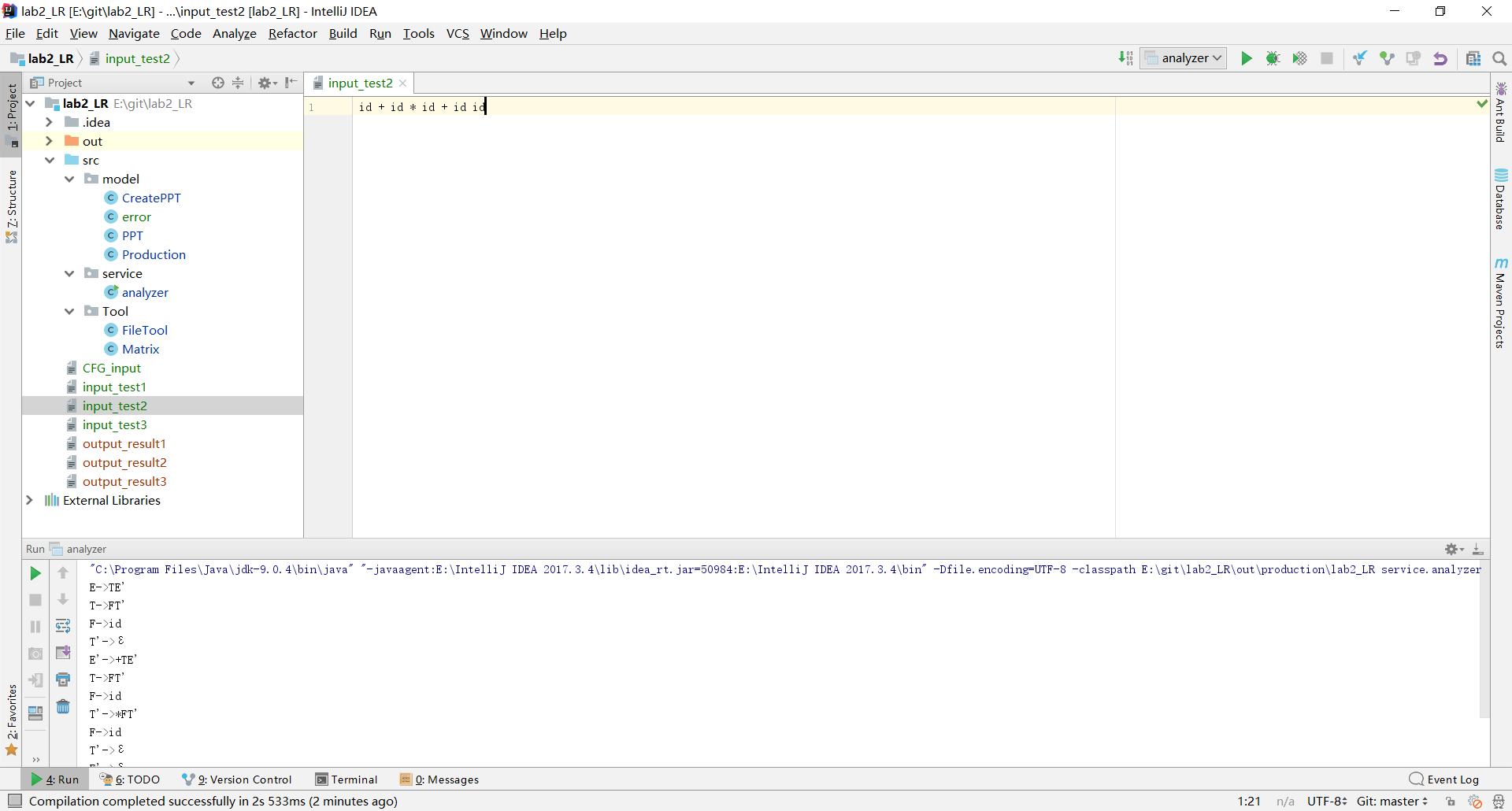
Output\_result1输出文件：



控制台输出：



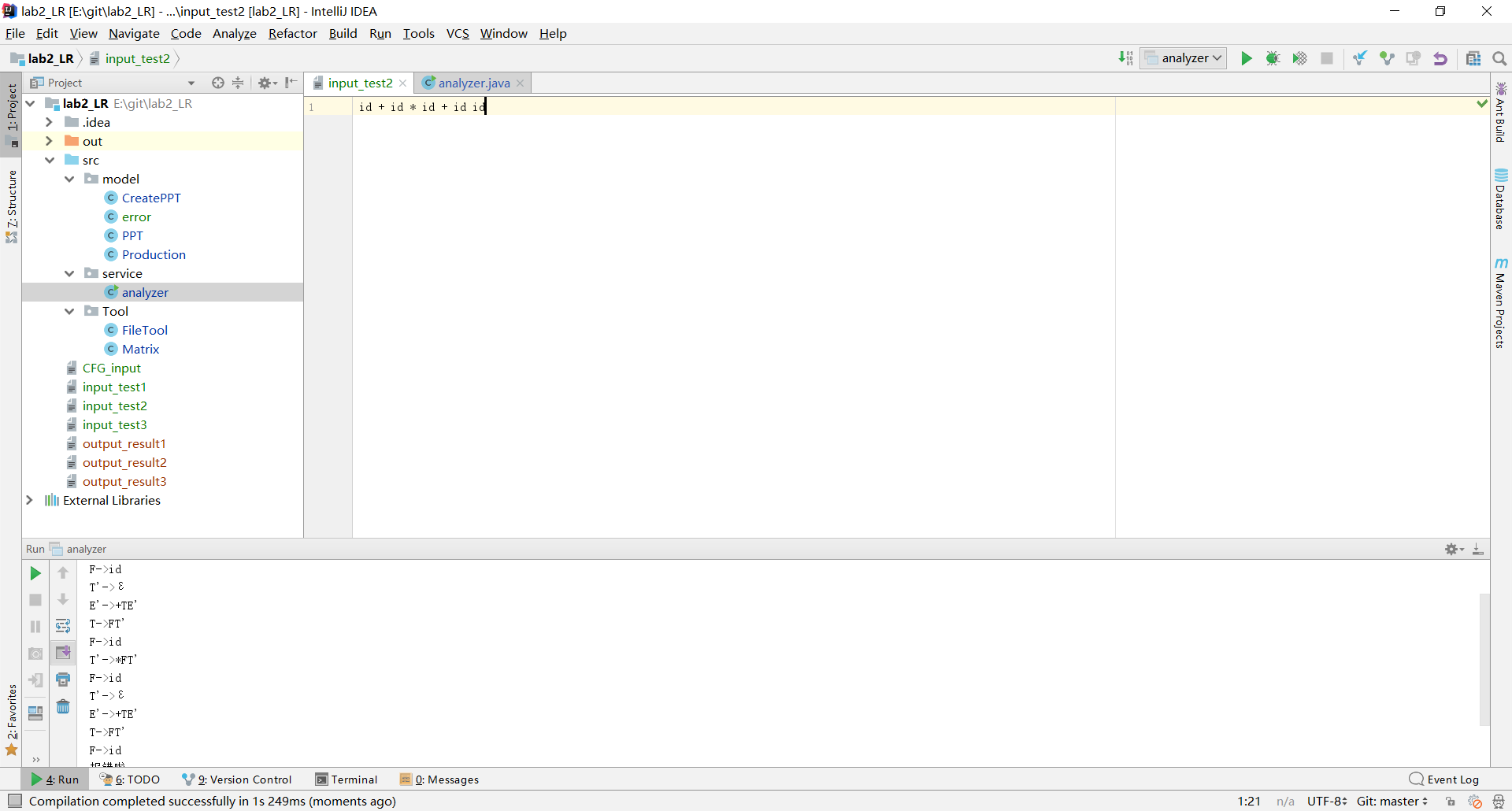
Input\_test2输入文件：



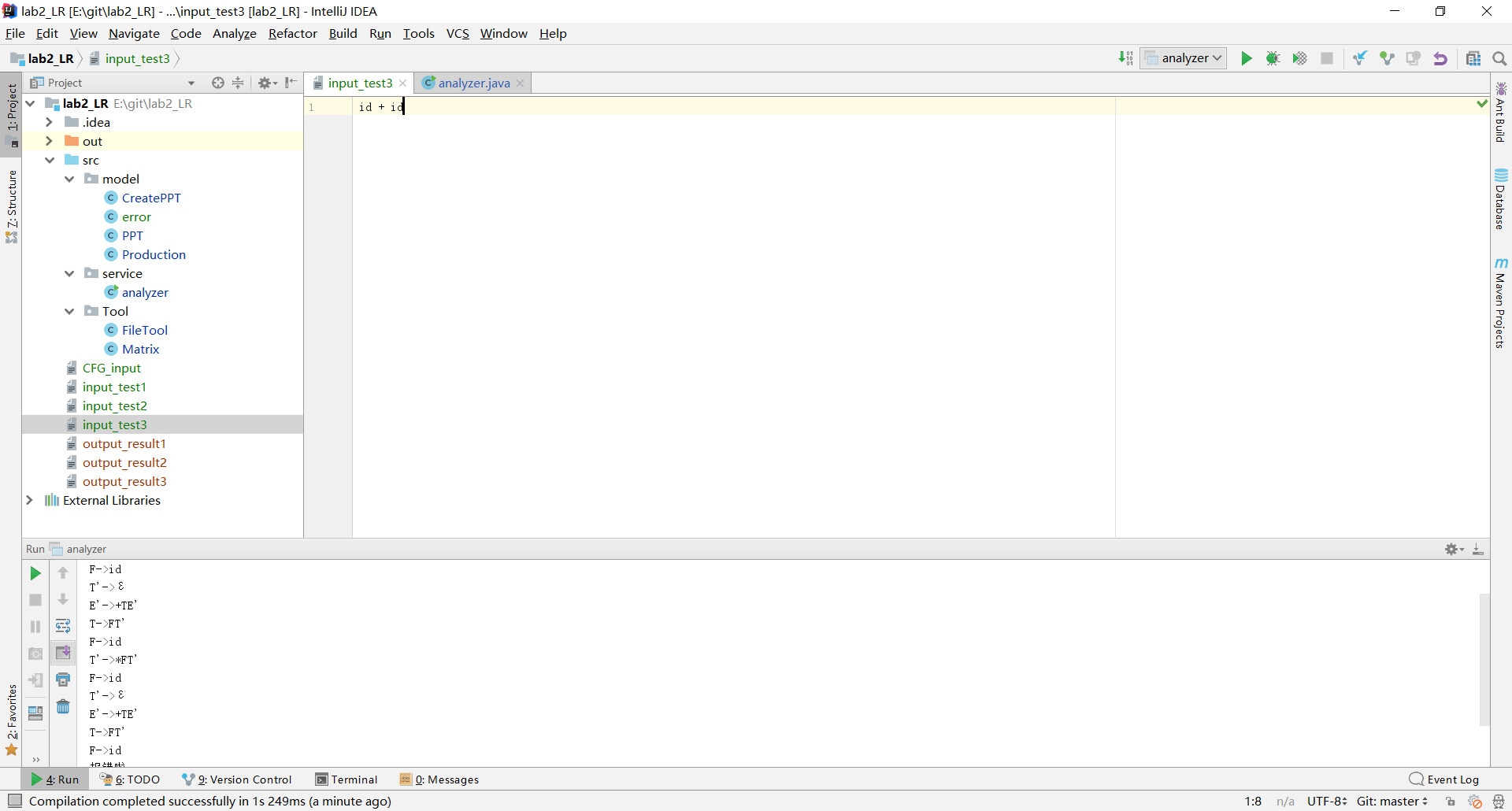
Outoput\_result2输出文件：



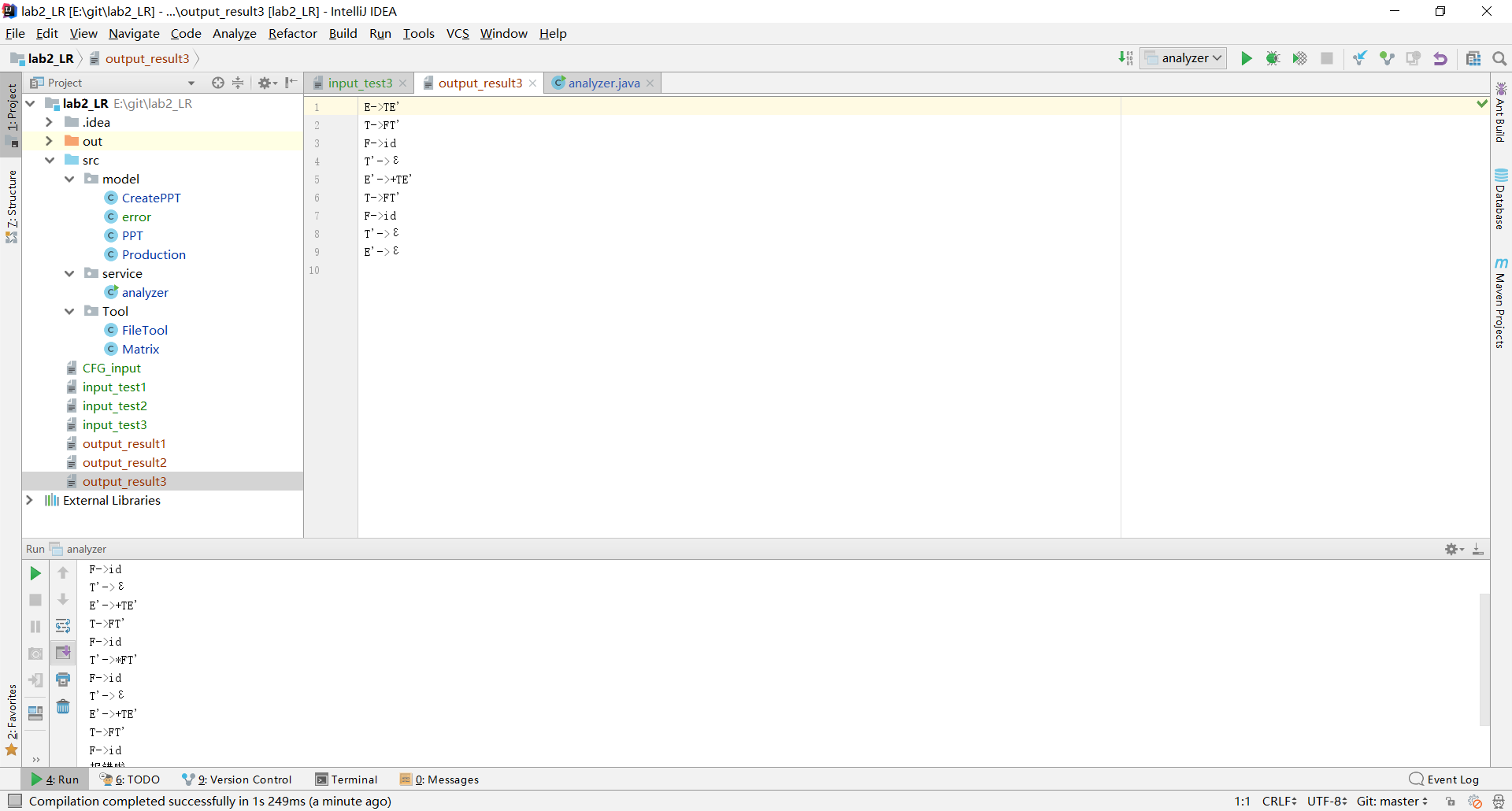
控制台输出：



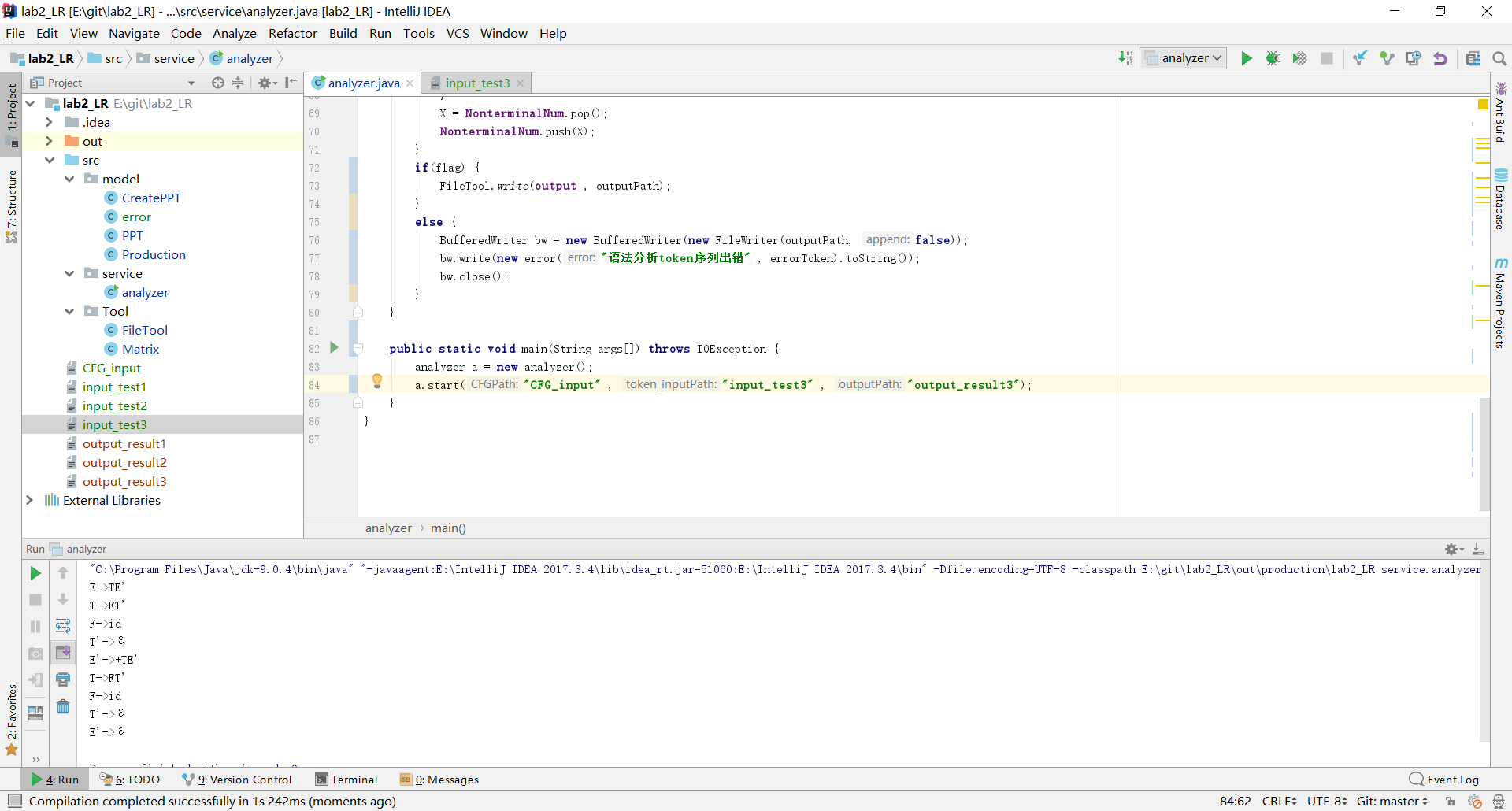
Input\_test3输入文件：



Outoput\_result3输出文件：

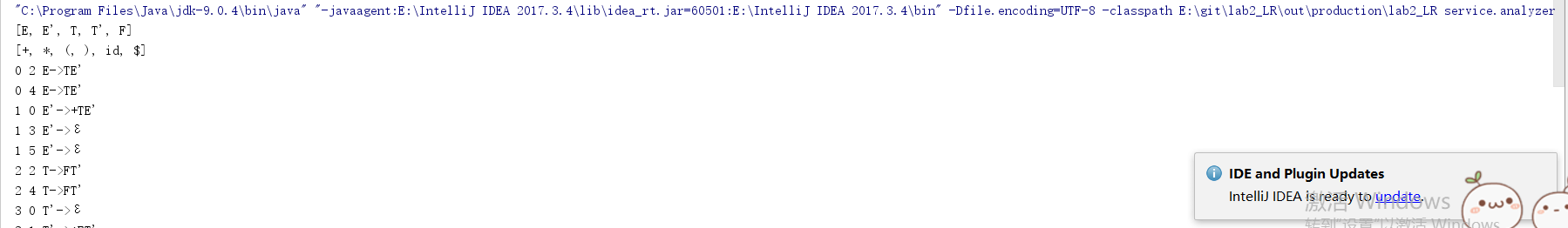


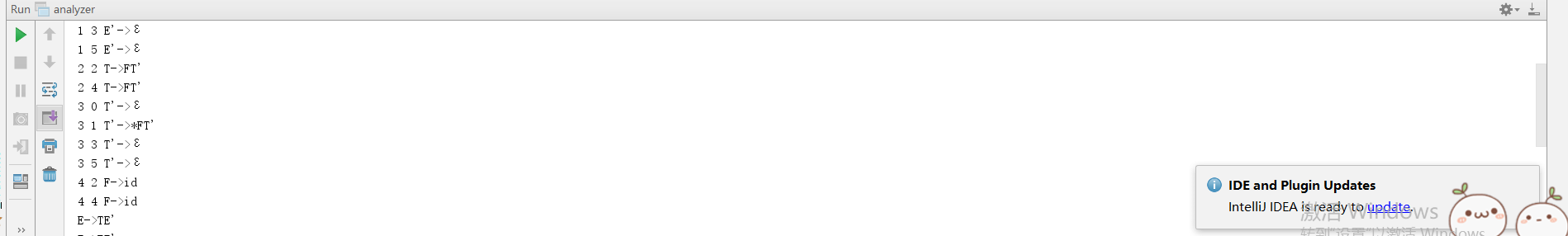
控制台输出：



程序入口就在这个main函数里，如果助教想测试的话，只需要把路径换一下就好了！

预测分析表输出检验：





# 问题与解决

# 一开始忘记在token序列后面加上”$”结束符号，导致在执行第一个条件判断语句的时候，会报字符串指针越界的错误。因为有一种情况（碰巧被我测试的时候碰上了）当把最后一个token弹出的时候，如果没有”$”，循环可能还会继续走下去，因为栈这时未必为空，而不添加”$”的token序列已经到头了。所以当进入第一个条件时，指针会加一，也就会报越界错误了。因此，我后来的解决方案就是在得到的token序列后加上”$”。

# 感受与总结

实验发现，感觉最难实现的还是生成语法分析表，大部分的代码实现都是花在了写生成预测分析表PPT上了。得到语法分析表之后，进行语法分析器就容易得多。所谓“工欲善其事必先利其器”，差不多就是这个道理吧！