Министерство образования Республики Беларусь

Учреждение образования

«Брестский государственный технический университет»

Кафедра ИИТ

Лабораторная работа № 2

По дисциплине ЯПИС

За шестой семестр

Тема: «Проектирование лексического анализатора»

Вариант 7

Выполнил:

Студент 3 курса

Группы ИИ-16(2)

Пешко А.С.

Проверил:

Слинко Е.В.

Брест, 2021

**Цель работы:** изучение основных понятий теории регулярных грамматик, ознакомление с назначением и принципами работы лексических анализаторов (сканеров), получение практических навыков построения сканера на примере заданного простейшего входного языка.

**Ход работы**

**Задание:** для выполнения лабораторной работы требуется написать программу, которая выполняет лексический анализ входного текста в соответствии с заданием и порождает таблицу лексем с указанием их типов и значений. Текст на входном языке задается в виде символьного (текстового) файла. Программа должна выдавать сообщения о наличие во входном тексте ошибок, которые могут быть обнаружены на этапе лексического анализа.

Длину идентификаторов и строковых констант считать ограниченной 32 символами. Программа должна допускать наличие комментариев неограниченной длины во входном файле. Форму организации комментариев предлагается выбрать самостоятельно.

**Вариант 7:** входной язык содержит последовательность вызовов процедур, разделенных символом **;**(точка с запятой). Вызов процедуры должен состоять из имени процедуры и списка параметров. В качестве параметров могут выступать идентификаторы, строковые константы, заключенные в двойные кавычки и одиночные символы, заключенные в одинарные кавычки.

import java.io.\*;  
import java.util.\*;  
  
public class Main {  
 public static void main(String[] args) {  
 String text = read();  
 System.out.println("SOURCE TEXT : " + text);

// проверка на наличие во входном тексте ошибок  
 if(!isStringSpellingCorrect(text)){  
 return;  
 }

String[] words = text.split("[\\s.,(); ]+");  
  
 words = deleteCommentWords(words);

int incorrectWord = correctWords(words);  
 if (incorrectWord >= 0) {  
 System.out.println("word \"" + words[incorrectWord] + "\" is not correct");  
 return;  
 }

boolean closeQuote = false;  
 int wordCount = 0;  
  
 for (int i = 0; i < text.length(); i++) {  
 char symbol = text.charAt(i);

// комментарии оформляются в виде : ‘/\* \*/’   
 if(symbol == '/' && text.charAt(i+1) == '\*'){  
 while (true){  
 if(text.charAt(i) == '\*' && text.charAt(i+1) == '/'){  
 break;  
 }  
 i++;  
 }  
 i++;  
 }  
 // если слово начинается с буквы, то это имя процедуры, т.к. другие параметры заключены в ковычки  
 else if (symbol >= 97 && symbol <= 122) {  
 System.out.format("%-20s%s\n", words[wordCount], "Procedure name");  
 i += words[wordCount].length() - 1;  
 wordCount++;  
 } else if (symbol == '(') System.out.format("%-20s%s\n", symbol, "Open parenthesis");  
 else if (symbol == ')') System.out.format("%-20s%s\n", symbol, "Close parenthesis");  
 else if (symbol == ';') System.out.format("%-20s%s\n", symbol, "Semicolon");  
 else if (symbol == ',') System.out.format("%-20s%s\n", symbol, "Comma");

// если слово заключено в одинарные ковычки  
 else if (symbol == '\'') {  
 System.out.format("%-20s%s\n", symbol, "Single quotes");  
 if (!closeQuote) {  
 System.out.format("%-20s%s\n", text.charAt(++i), "Single character");  
 closeQuote = true;  
 wordCount++;  
 } else closeQuote = false;  
 }

else if (symbol == '\"') { // если слово заключено в двойные ковычки  
 System.out.format("%-20s%s\n", symbol, "Double quotes");  
 if (!closeQuote) {  
 System.out.format("%-20s%s\n", words[wordCount].substring(1, words[wordCount].length() - 1), "String constant");  
 closeQuote = true;  
 i += words[wordCount].length() - 2;  
 wordCount++;  
 } else closeQuote = false;  
 }  
 }  
 }  
  
 // построчное считывание файла  
 public static String read() {  
 String input = "";  
 try {  
 File file = new File("input.txt");  
  
 FileReader fr = new FileReader(file);  
  
 BufferedReader reader = new BufferedReader(fr);  
  
 input = reader.readLine();  
  
 } catch (FileNotFoundException e) {  
 e.printStackTrace();  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 return input;  
 }

// проверка на наличие во входном тексте ошибок  
 static boolean isStringSpellingCorrect(String str) {  
 boolean result = true;  
  
 HashMap<Character, Integer> map = new HashMap<Character, Integer>();  
 String test = str;  
 char[] chars = test.toCharArray();  
  
 for (int i = 0; i < chars.length; i++) {  
 if (!map.containsKey(chars[i])) {  
 map.put(chars[i], 1);  
 } else {  
 map.put(chars[i], map.get(chars[i]) + 1);  
 }

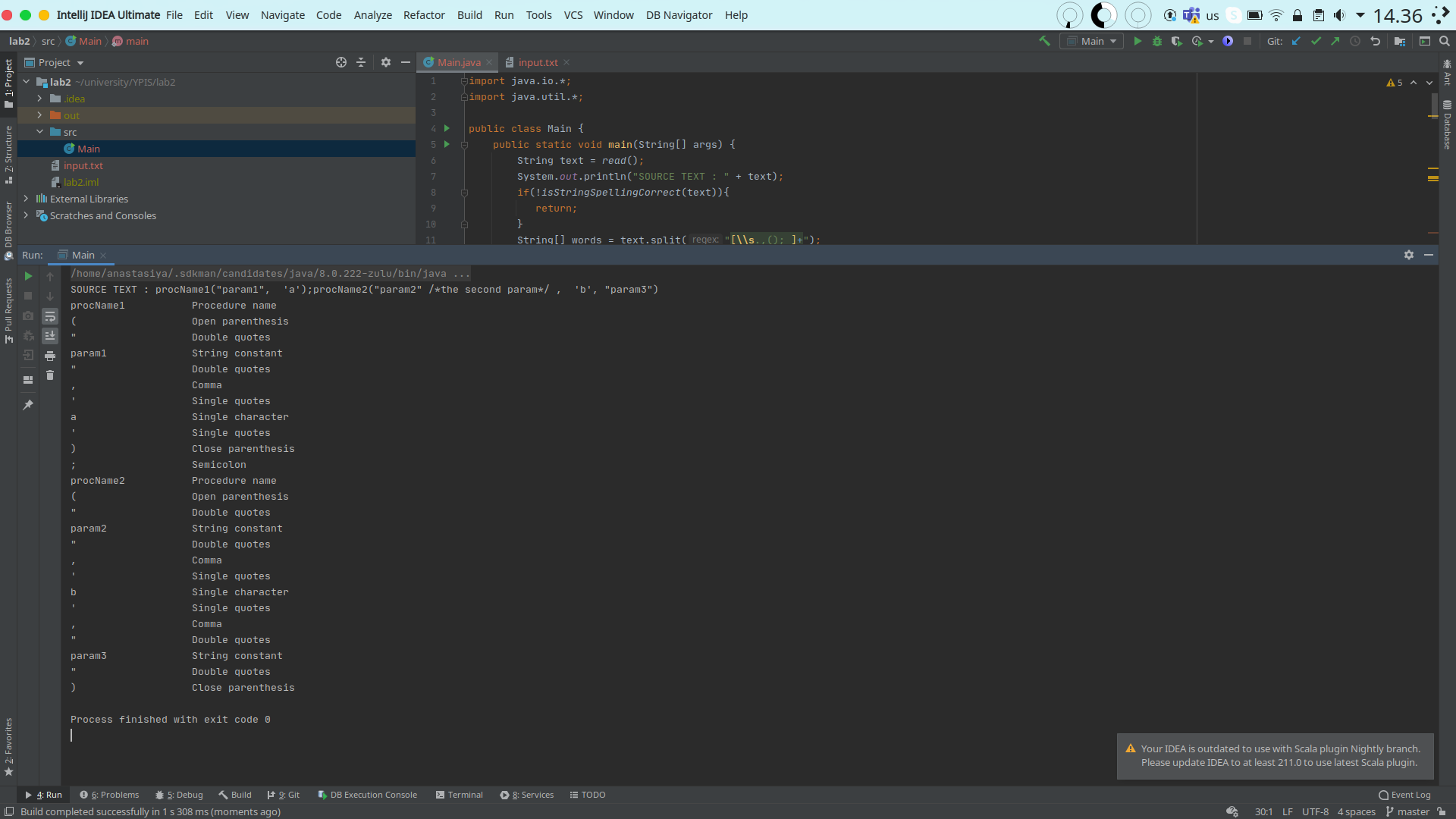
if (chars[i] == ')' && chars[i + 1] != ';') {  
 System.out.println("symbol ; was expected!");  
 result = false;  
 }

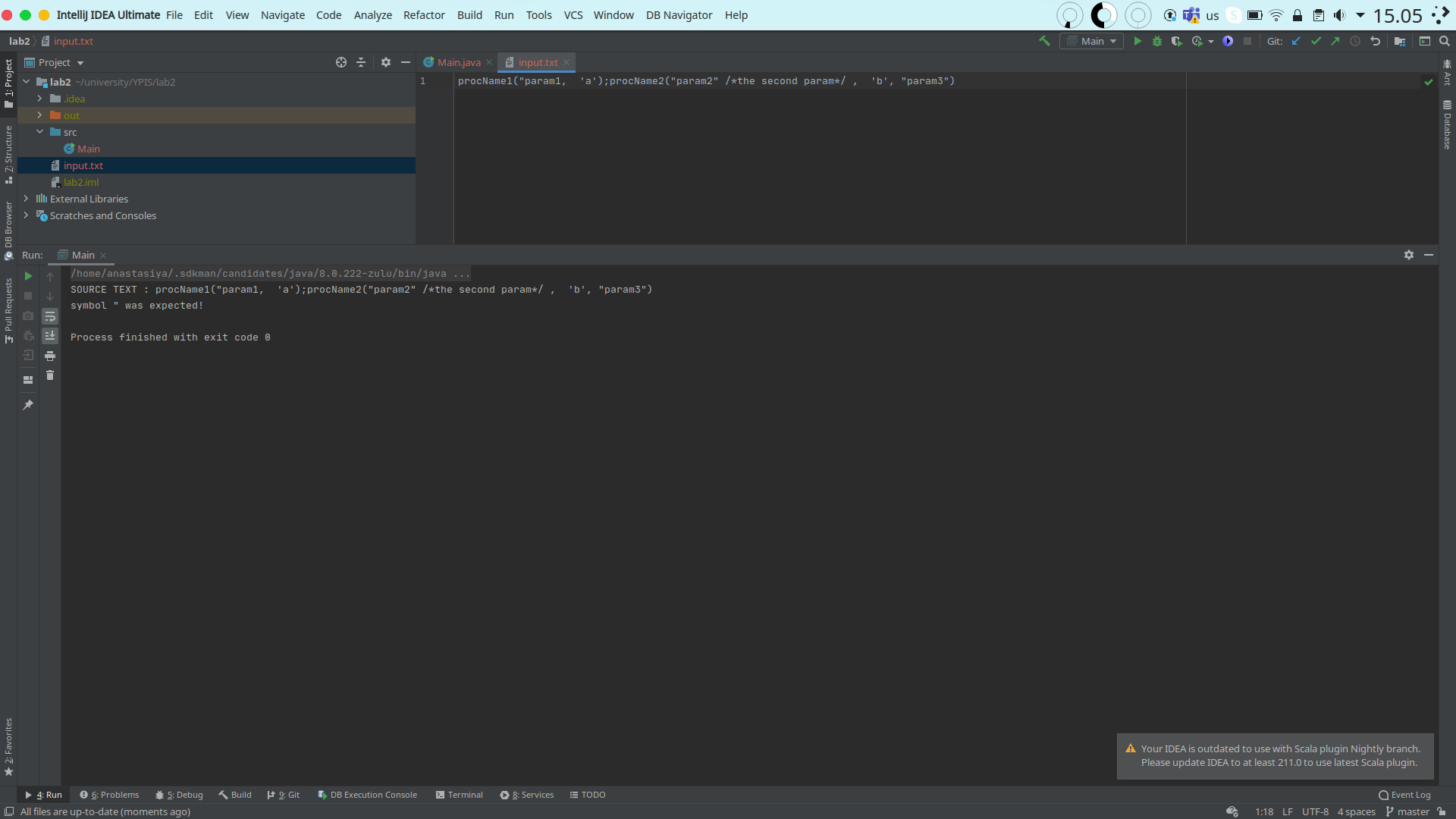
}  
  
 if (!map.get('(').equals(map.get(')'))) {  
 char c = map.get('(') > map.get(')') ? ')' : '(';  
 System.out.println("symbol " + c + " was expected!");  
 result = false;  
 }  
 if (map.get('"') % 2 != 0) {  
 System.out.println("symbol " + '"' + " was expected!");  
 result = false;  
 }  
 if (map.get('\'') % 2 != 0) {  
 System.out.println("symbol " + '\'' + " was expected!");  
 result = false;  
 }  
 if(!correctComments(str)) {  
 System.out.println("comments is not correct!");  
 result = false;  
 }  
 return result;  
 }

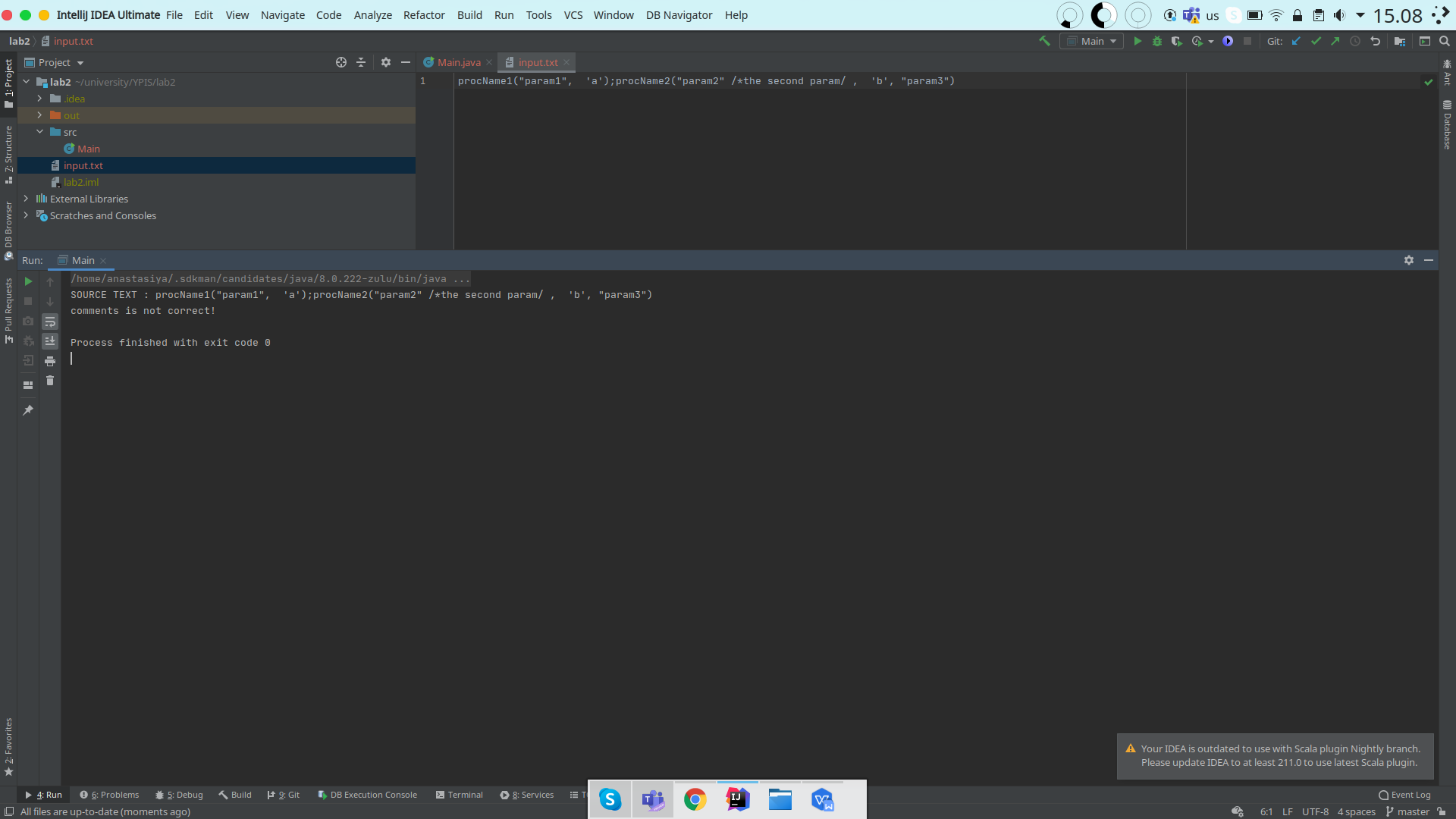
// проверка корректность оформления комментариев

static boolean correctComments(String source) {  
 boolean commentOpen = true;  
 for (int i = 0; i < source.length() - 1; i++) {  
 if(source.charAt(i) == '/' && source.charAt(i+1) == '\*'){  
 commentOpen = true;  
 }  
 if(source.charAt(i) == '\*' && source.charAt(i+1) == '/' && commentOpen){  
 commentOpen = false;  
 }  
 }  
 // комментарии закрыты  
 return !commentOpen;  
 }  
  
 static String[] deleteCommentWords(String[] words) {  
 int start = 0, end = 0;  
 List<String> list = new LinkedList<>(Arrays.asList(words));  
 for (int i = 0; i < words.length; i++) {  
 if(list.get(i).charAt(0) == '/'&& list.get(i).charAt(1) == '\*'){  
 start = i;  
 }  
 if(list.get(i).charAt(list.get(i).length() - 2) == '\*'&& list.get(i).charAt(list.get(i).length() - 1) == '/'){  
 end = i;  
 }  
 }  
  
 list.subList(start, end+1).clear();  
 words = list.toArray(words);  
 return words;  
 }  
}

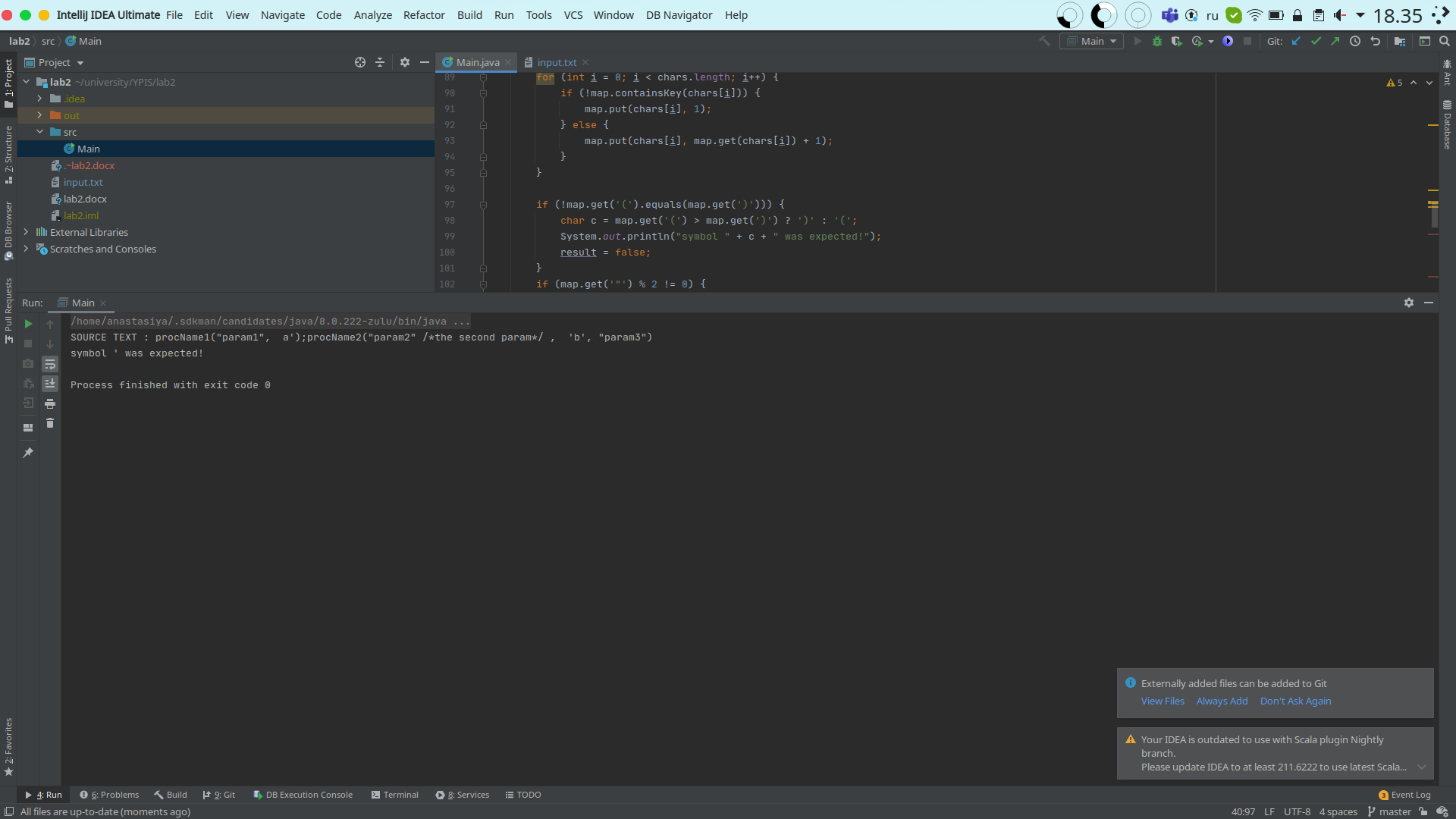
// метод проверяет правильность имени процедур  
// возвращает индекс неправильного слова  
// или -1 в случае, если неправильное слово отсутсвует  
static int correctWords(String[] words) {  
  
 // проверка первого символа названия процедуры (должна быть буква или нижнее подчеркивание)  
 int incorrectWord = -1;  
 for (int i = 0; i < words.length - 1; i++) {  
 String word = words[i];  
 char symbol = word.charAt(0);  
 if (symbol >= 97 && symbol <= 122 || symbol == '\_' ||  
 symbol == '\'' || symbol == '\"') {  
 } else {  
 incorrectWord = i;  
 return incorrectWord;  
 }  
 }  
  
 // проверка на наличие запрещенных символов в середине слова  
 for (int j = 0; j < words.length; j++) {  
 String word = words[j];  
 for (int i = 1; i < word.length(); i++) {  
 char symbol = word.charAt(i);  
  
 if (symbol != '\'' && symbol != '\"') {  
 if (symbol >= 97 && symbol <= 122 || symbol == '\_' ||  
 symbol >= 48 && symbol <= 57 || symbol >= 65 && symbol <= 90) {  
  
 } else {  
 System.out.println("symbol " + symbol + " is unresolved in this context!");  
 incorrectWord = j;  
 return incorrectWord;  
 }  
 }  
 }  
 }  
 return incorrectWord;  
}

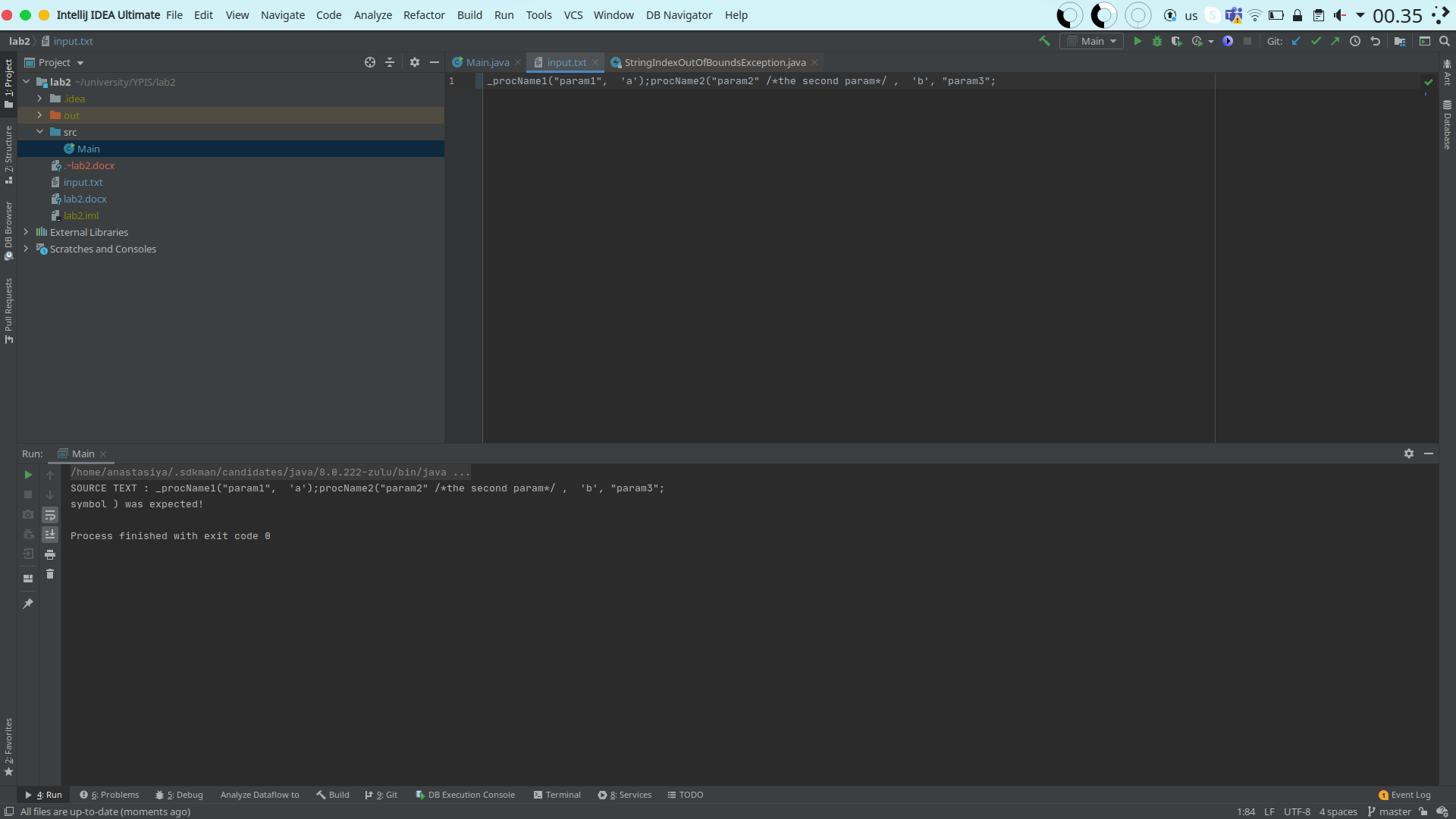


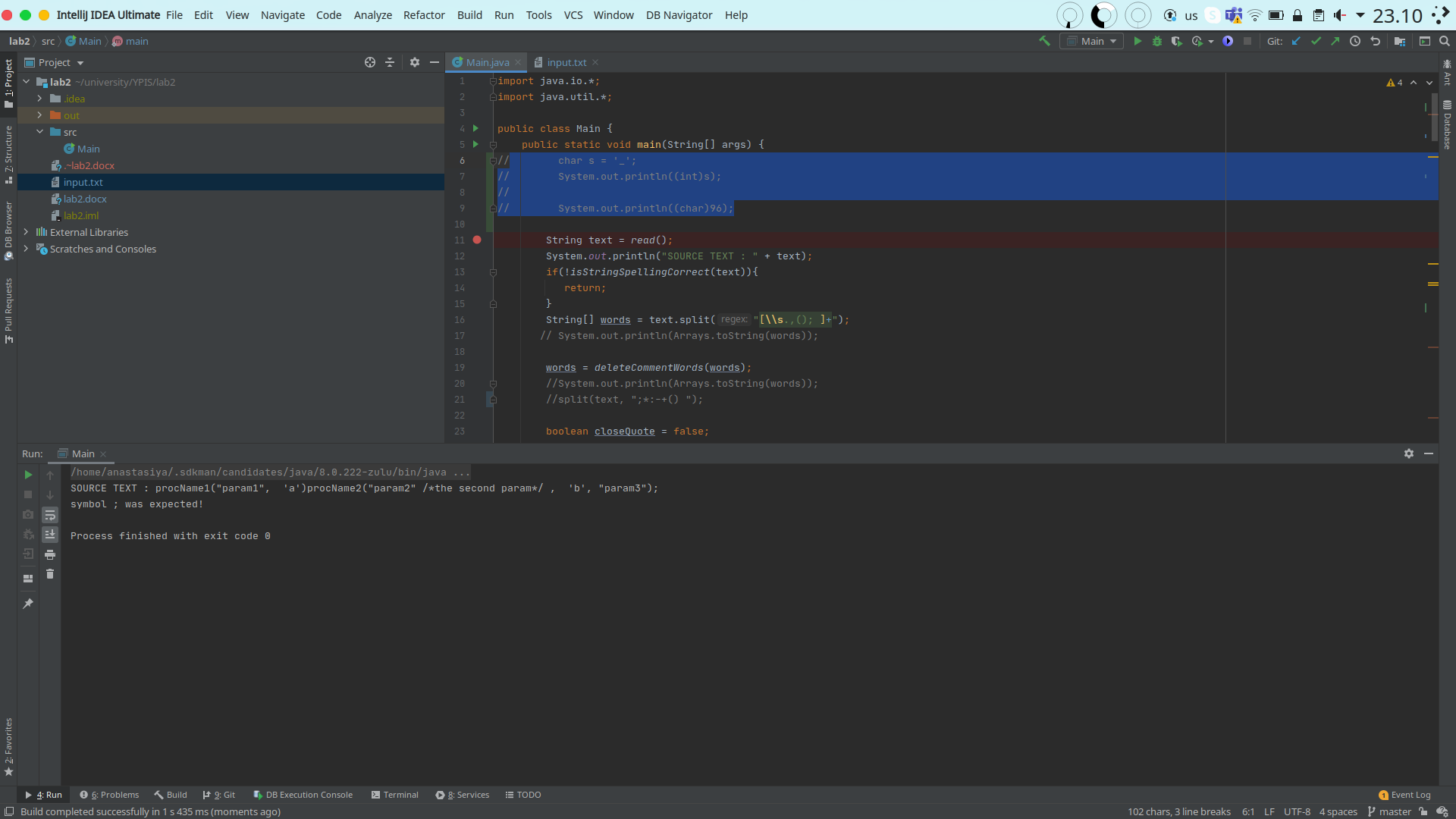


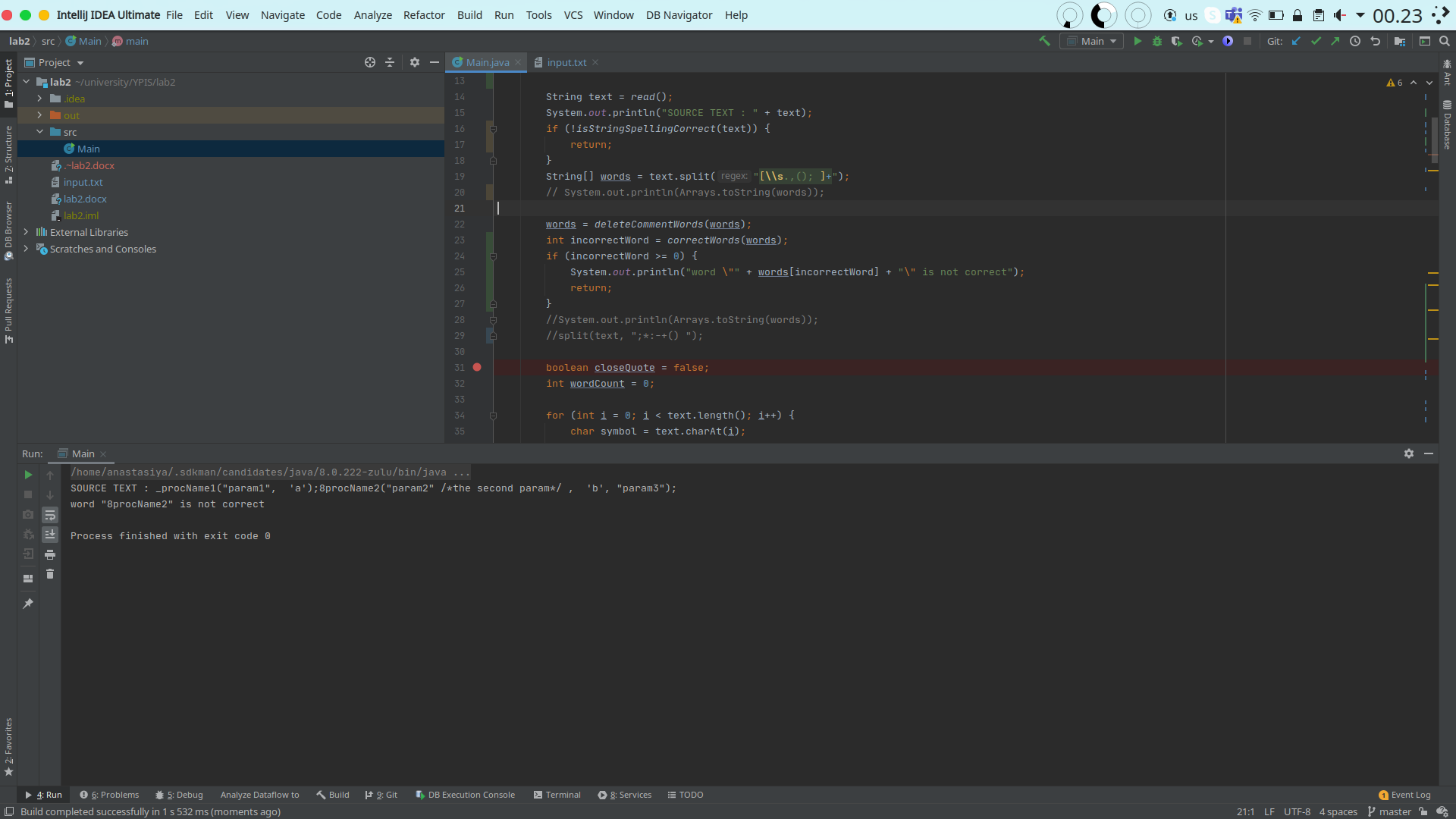


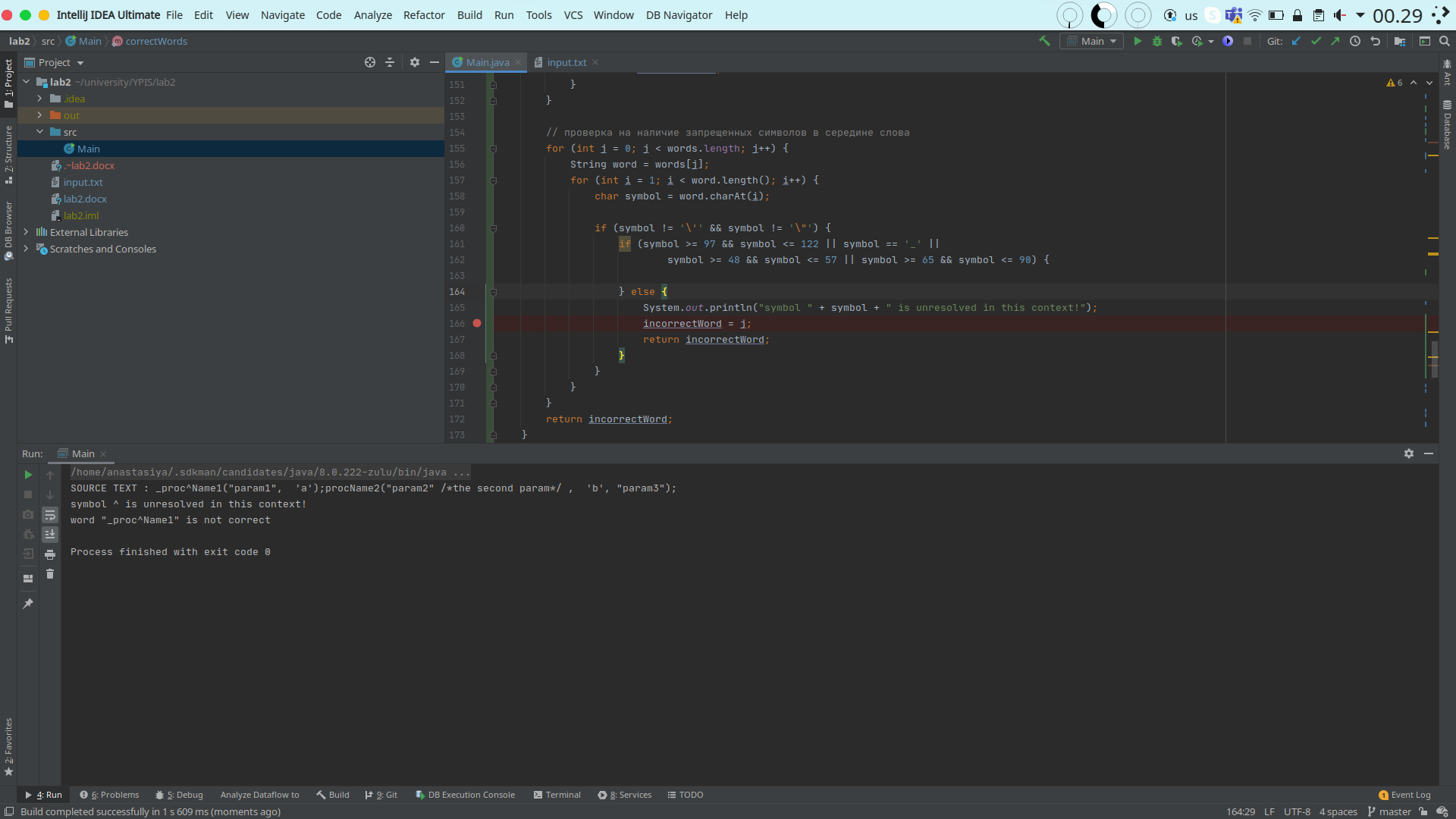
Добавлено:











**Вывод**: изучили основные понятия теории регулярных грамматик, ознакомились с назначением и принципами работы лексических анализаторов (сканеров), получили практические навыки построения сканера на примере заданного простейшего входного языка.