Project 1: JavaCC Parser and

Profesor: Silvia Takahashi

Course: Lenguajes y Maquinas (Ingles)

Section: 1

Daniela Uribe 201923291
Juan Felipe Patiño 201922857

7 Octubre 2022

Abstract:

Using JavaCC and nested Java classes to create a self-contained interpreter and executer for coding structures passed as text in a pre-defined world for a Robot.

Key Words: Parser, JavaCC, Java, Tokens, Instructions, Code

1. Introduction:

The task at hand as to take the previous project, Project 0, to its next step. A parser checking in JavaCC that works. That means that not only must the syntax be correct at all points, but also that if there are lines of executable code it's executed in the self-contained program of each input. First, we have the constraint of working with the Robot.jj class which means we are limited in ways of interacting with the world, especially since for most of the methods there is no direct equivalent. The work was divided in two parts mainly, the first one was checking syntaxis of all the declarations, and the program. The second part was the functionality, or applicability, which included storing the created functions and being able to run the executable code and have it affect the game. Two achieve these two we used the JavaCC file which also allowed us to use Java in some parts, including creating two classes in Java for our use.

2. Materials and method

a. Java Classes

The first nested class we have is "Instruction". We stablish final static String variables for each pre-existing instruction type, or block, and a case for user-made functions. Afterwards we have many different constructors that have input options according to those that are in commented list of functions before each constructor. Finally, we have a return function for all the values within a given instruction. The purpose of this class it to effectively store instructions that can be later easily executed, and also adapt according to the value of any given parameter.

The second nested class we have is "Parameter". The parameter is a way to store the numeric value of a local variable for a function such that it can change its value according to the parameters passed to a function when invoked.

b. Main Methods

After tokenizing key words, phrases, functions, and more, we begin with our main input structure. This is made up of other functions for each possible valid structure which are variables, many procedures, some commands to execute all encapsulated by the appropriate begin and end statements.

The procedure can create new functions, without executing them, to later be used. We check its basic structure and store the used parameters within a HashMap of parameters to later be used while creating the instructions. After the declaration we have the actual instructions within the procedure, in which we consider any valid pre-existing function or an already created function as valid values, which then is created as an instruction object and stored in the array for said function. In the case of conditional statements, we check them separately and return their instruction object, which goes within the control structure instruction.

For the commands we have a similar structure to procedure, however instead of creating we execute the code where appropriate or call upon the instructions of the function to be executed.

To execute we go over every instruction of the array given as those to be executed and according to the case in the name of the instruction the appropriate steps are taken to reflect the expected changes in the world according to the current values of the parameters or variables.

c. Function Methods

Afterwards, we have methods related to function validation regarding their parameters, variables, and other. We also can search the values said parameters or variables by asking all three cases

and returning the valid one as we use the number 1000 as the base, invalid, case. It's important to highlight that we have a very specific method in charge of nested function usage as they are a fringe case that needs special attention to be dealt with due to the nature of the parameters used. Function Methods

d. Control Structure

For each control structure we have our own separate method. For example, for repeatTimes it is a simple java for executing the inner code. On the other hand, the while shares with the if a conditional checker that evaluates the Boolean value of the conditional statement and according to this value executes, or doesn't, the next block of code. In the case of the while this happen until it is no longer a valid question statement, while the if does it only once. The conditional statement is taken and then each possible conditional question is evaluated in its own method, where the most complex is the is valid, which must evaluate several possible methods without executing them. For this we created unique methods for each case such that it has the same steps as the normal method without affecting the world of the robot.

e. Pre-existing Methods

While some methods are already implemented in the world with a simple method for many others, we had to create valid equivalents that usually were compromised of one or more RobotWorld methods suited to our needs. Often the whole code was taken with adaptions as needed for simplicities sake.

f. Search Methods

We have several search methods as we can never be sure for which specific type of data is being called. For example, we can either have a variable or a number as the valid one, and we check and return the number it is supposed to be

3. Results

The results were a successful JavaCC file which did everything asked. We checked it by executing the example code, but also many other possibilities. Likewise, we tried invalid instances to ensure it was correctly detecting errors along the way and it successfully did.

4. Discussion

a. Difficulties

There were several problems we had to face while developing our solution. One of the first ones we faced was the finnicky nature of JavaCC and having to use Extended BNF the whole way through while checking all the code, while not a particularly challenging issue it was one on the first encountered ones. The next was how to properly store everything we needed in the constraints of Java, as we needed the name, the parameter, the instructions and more for each function, and we can't use arrays of any type as, for example, python would allow. That is why we created several HashMap for each needed case of storing something. Which leads to the two biggest issues that we had to overcome, storing the instructions of each method, and actively updating the references for a method. We solved this by using the java classes so that when a user made function is called the instructions are easily found and executed while dynamically updating parameter values thanks to the two java classes.