Kjetil Svalestuen / kjetisva
INF3110 Mandatory 2

Design Document

Implementation

I implemented my solution using *Java 8*. Java is object-oriented and supports various features such as abstract classes, interfaces, enums and so on. In addition, I know this language well which made the choice quite obvious to me. I could have written the solution in another language such as Python, but wanted to go the safe route this time.

When designing the data structure, I started with the precode, modified it to my liking and took some inspiration from my ML solution. I use three interfaces, each with a unique method, to distinquish my classes.

- interface Robol
    - public void interpret();
- interface Statement
    - public void interpret(Robot robot);
- interface Expression
    - public int evaluate(Robot robot);

First, there is the Robol-interface, where all implementations need an interpret()-method with no return value. This is used by the Program and Robot-classes. Then there's the Statement interface, implemented by all statements. Lastly, there's the Expression interface where all implementations need an evaluate()-method with an integer return value.

Two of these interfaces need a robot as an argument. After the robot object has had its grid set by program, it is passed on to elements in the abstract syntax tree as an argument of the interpret()-function. The elements then modify variables of the robot, including the VarDecl-list (Assignment-commands) and the position (Move-commands) using set-functions.

In contrast to my ML solution, I store values themselves in each VarDecl-object. This means that each expression only needs to be evaluated once, either in the initial VarDecl list or through an assignment, and Ident-commands need only to get an integer-value.

When interpreting Move-commands, I assure that the robot stays in its static grid. If it falls off, I throw an OutOfboundsException and exits the Java instance. I also made a NoSuchVariableException that I raise in case an Assignment-command has an identifier for name that does not exist.

Kjetil Svalestuen / kjetisva

Running the Program

The ‘Mandatory’ main class creates AST representations of all the example programs supplied in the assignment text. These should be easy to read, write and modify.

You run the program by compiling all Java source files (*javac *.java*) and then running the Mandatory-class (*java Mandatory*). All hardcoded programs should then run in sequence, ending with an *OutOfBoundsException* on the last example.