Turtle Graphics

The assignment is to implement a simple domain specific language to control a Java program for Turtle Graphics. Parsing and translation and execution of the DSL should be implemented in Jython. The files that you need (assignment.jar) can be found as an attachment to this message.

DSL

The language to control the painting is an adaptation of the idea of turtle graphics. The language has 8 statements:

- `pen_down` – turn pen on (movement draws on canvas)
- `pen_up` – turn pen off (movement does not draw on canvas)
- `move_forward` – move one step forward
- `move_backward` – move one step backward
- `move(STEPS, ANGLE)` – move `STEPS` steps in `ANGLE` direction, where `STEPS` and `ANGLE` are integer literals `(1, 2, ...)` or loop variables (see below)
- `turn cw(ANGLE)` – turn clockwise, where `ANGLE` is an integer literal or a loop variable (see below)
- `turn ccw(ANGLE)` – turn counter clockwise, where `ANGLE` is an integer literal or loop variable (see below)
- `put(XPOS, YPOS, ANGLE)` – puts the pen on `XPOS`, `YPOS` pointing in `ANGLE` direction
- ```
  for X=0 to 10 do
    STATEMENTS
  end
  ```
  repeat `STATEMENTS` 10 times (`STATEMENTS` may not contain another loop). `X` should be visible to `STATEMENTS` (but not outside the loop).

`ANGLE` is the number of degrees to turn clockwise from the current direction. `STEPS` is number of pixels

All arguments passed to statements taking arguments must be integer literals, loop variables or a combination of the two separated by one of the following binary arithmetic operations: `+, -, *`

Examples:

```
put(100, 100, 90)  - place on 100x100 facing east
move(10, 45)       - move 10 steps in 45 degree angle of facing
for X=0 to 4 do    - draws a square with the side 50 pixels
   move(50, 90)
end
```
```

The Task

The program consists of a Java application with a canvas and a textarea for turtle code. Your job is to write a Jython application that takes turtle code from the Java application, parses it with regular expressions and calls setPixel(x,y) in the Java application to draw the corresponding image.

The turtle remembers its current angle.

For example the following turtle code program draws a rectangle with the sides 50 and 100 pixels:

```
put(150, 150, 0)
move(50, 90)
move(100, 90)
move(50, 90)
move(100, 90)
```

The following turtle code program draws a tilted square with the side 50 pixels:

```
put(100, 100, 45)
for X=0 to 4 do
  move(50, 90)
end
```

The following code renders the face in Figure 1, below:

```
put(270, 150, 180)
for var = 0 to 180 do
 move(4, 2)
end

put(200, 80, 180)
for var = 0 to 45 do
 move(1, 8)
end

put(100, 80, 180)
for var = 0 to 45 do
 move(1, 8)
end

put(180, 150, 180)
for var = 0 to 45 do
 move(2, 4)
end

for x=0 to 100 do
put(100+x, 190, 230-x*2)
 move(10, 0)
end
```

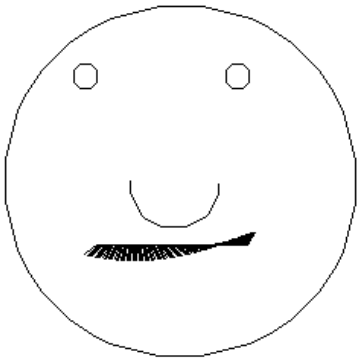

Figure 1: Funny face

For this task there is a preexisting python bridge class called JythonTranslater.Jtrans, that accepts events from the Java program. You should subclass this class and override its methods.

Restrictions

You may only interact with the Java program by calling `setPixel(x, y)` to control the painting and `getCode()` to get the code entered in to the turtle code textarea. These methods are both defined in the DYPL Java class.

You may not change the Java code in any way.

All code to implement this assignment should be Python code.

Hints

In order to solve this assignment it is recommended that you utilize regular expressions. A good idea might be to write a Python program that just parses a turtle code file in order to eliminate as many sources of error as possible for your first attempts to write Python code. Jython is only required for the actual canvas drawing when the Java application must work together with the Python application.

To get you started you get the pen up and pen down [regular expressions]:

```
"\s*pen_down\n"
"\s*pen_up\n"
```

In Python it is possible to take a string and evaluate that string as if it were python code. For instance:

```
class Spam(object):
    def __init__(self, eggs):
        if eggs == "some_method":
            eval("self.{0}()".format(eggs))

    def some_method(self):
        print("{0} executing some_method".format(self))
```

If you keep all class files in the same directory you should have less classpath problems.

Running the DYPL program

Unjar the assignment.jar file:

```
jar -xf assignment.jar
```

To run your program:

```
java -jar /path/to/jython/home/jython.jar MyJythonFile.py
```

Resources

The Jython Home Page: `http://www.jython.org`
The Python Home page: `http://www.python.org`
Regular Expressions: `http://www.amk.ca/python/howto/regex`
General knowledge: `http://google.com`