Ido Shoshani

Reflection on OO Project

My project ended up having a very small scope, just a simple Logo dialect. However, I know that I have a tendency to overcomplicate and feature creep my projects without actually cleaning them up at all, so I decided to make a well-documented, understandable small project instead of a larger messier one. Even then though, I definitely came upon a number of issues, some of which I managed to solve.

The first thing I had to learn was understanding when the dialect was running. I initially tried to initialize all my objects at the top level. This went quite poorly, as the dialect runs during compile time, so objects may be declared, but trying to initalize, say, the graphics application, we end up trying to initialize a graphics application within the compiler, which isn't entirely reasonable. Instead though, I tried to apply a lazy initialization process. All the top level variables are initialized as a notInitialized object, which only has the method == to check if a variable points to that notInitialized object. Within the dialect methods then, before I could use any of my variables, I needed to use the corresponding lazy initialize method for that variable. I actually call all of them in the method *show* but in order to be forgiving of the case where the user does not call show first thing in their program, I also initialize whatever variables are used in other places as well.

Once I had that working, and all my tests running as expected, I needed to add in some rules. However, I came upon some issues here as well. While getting rules to check variable declarations s was easy enough, getting those to play nice with the method declaration in the dialect just did not work. As soon as a method with a parameter was declared, it would stop at compile time. I tried to follow the examples from the minigrace github, but even the rules from dialect-example.grace copy-pasted in failed, so I suspect those may be deprecated. I was having an issue with aNode.FromNode, so in future work I would like to be able to fix whatever that is. For the moments I have simply disabled method declarations within the dialect, with an apology in the error message. This was another case where I decided that feature-creep could wait, and instead focus on having a solid core deliverable.

When I went back to testing though, I found that in checking for classes and methods, I also couldn't test my code anymore! So I settled for importing the module instead of using it as a dialect, so that at least my methods would certainly be working. Another thing on the todo list would be to disable classes unless they came from a test suite.

I also had a lot of fun creating the documentation website for my Module. My aim was that even a novice programmer should be able to use this dialect and be able to use it, so I decided to instead of using a README text or markup file, to create a full website with code and graphical examples. I also tried to space out the restrictions and methods of the module, so that a novice user could easily find what they are looking for on the webpage. I also decided to include some notes to anyone else who wants to fork or expand the project, just in case.