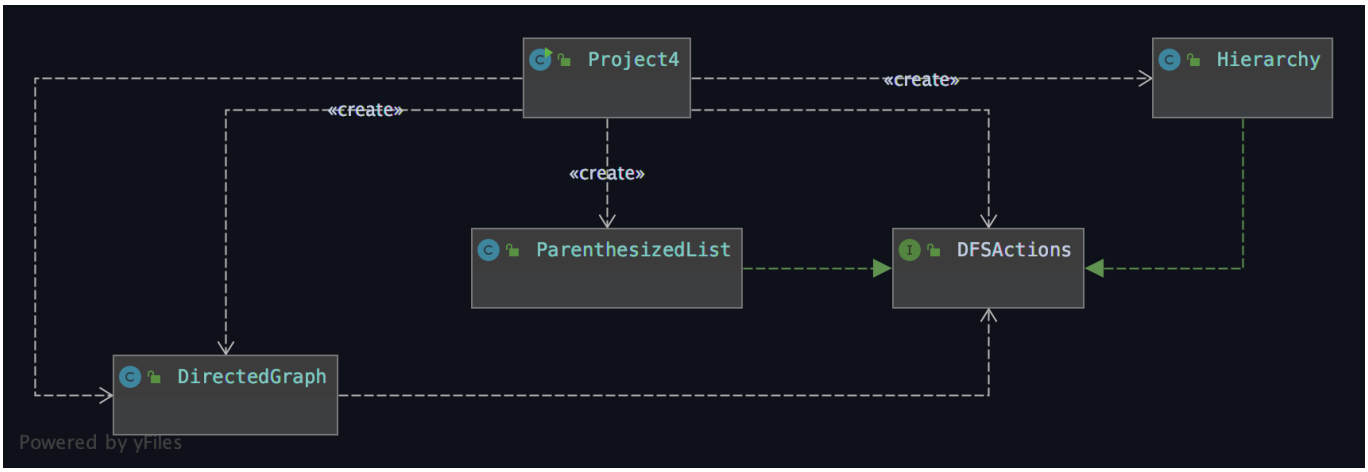
CMSC 350 - Project 4

Author: Tyler D Clark
Date: 10 May 2020

Description: A program that allows the user to select a file to be read. From that file, graphs are to be constructed using the DirectedGraph class. The program prints two different displays for the graphs, which use the ParenthesizedList and Hierarchy classes. Those two classes both implement the DFSActions interface, which creates the method bodies for actions in key locations of the Depth First Search.

UML chart



Test cases

Test Case	Input	Notes
1	input.txt	Test case of a graph with circular dependencies. Also the example from the instructions.

```
ClassA ( ClassC  *  ClassE ( ClassB ( ClassD ClassG  ) ClassF ClassH  ) ClassJ ( ClassB ( ClassD ClassG  )  )  )

ClassA
    ClassC *
    ClassE
        ClassB
            ClassD
            ClassG
        ClassF
        ClassH
    ClassJ
        ClassB
            ClassD
            ClassG

ClassI is unreachable

All read Vertices:
ClassA: ClassC ClassE ClassJ
ClassC: ClassA
ClassE: ClassB ClassF ClassH
ClassJ: ClassB
ClassB: ClassD ClassG
ClassI: ClassC
```

fig1. Output from processing input.txt

Test Case	Input	Notes
2	input2.txt	Test case of a graph with no circular dependencies.

```
class1 ( class2 ( class4 ( class8  ) class6 class8  ) class3 ( class6 class9  ) class4 ( class8  ) class5  )

class1
    class2
        class4
            class8
        class6
        class8
    class3
        class6
        class9
    class4
        class8
    class5


All read Vertices:
class1: class2 class3 class4 class5
class2: class4 class6 class8
class3: class6 class9
class4: class8


Process finished with exit code 0
```

fig2. Output from processing input2.txt

Test Case	Input	Notes
3	input3.txt	Test case of a graph with unreachable classes.

```
reachable1 ( reachable2 (  * reachable3  ) reachable3 reachable4 reachable5 reachable6  )

reachable1
    reachable2 *
        reachable3
    reachable3
    reachable4
    reachable5
    reachable6

unreachable1 is unreachable
unreachable2 is unreachable

All read Vertices:
reachable1: reachable2 reachable3 reachable4 reachable5 reachable6
reachable2: reachable1 reachable3
unreachable1: reachable3
unreachable2: reachable4
```

fig3. Output from processing input3.txt

Test Case	Input	Notes
4	input4.txt	Test case of a graph with no unreachable classes.

```
reachableA ( reachableB ( reachableC reachableD ( reachableE  )  ) reachableC reachableD ( reachableE  ) reachableE ( reachableF  )  )

reachableA
    reachableB
        reachableC
        reachableD
            reachableE
    reachableC
    reachableD
        reachableE
    reachableE
        reachableF


All read Vertices:
reachableA: reachableB reachableC reachableD reachableE
reachableB: reachableC reachableD
reachableD: reachableE
reachableE: reachableF
```

fig4. Output from processing input4.txt

Lessons learned

This project taught me a lot about the graph data structure, and the recursive implementation of the depth first search. Once again, I had the opportunity to get practice with custom data structures like the linked lists in this exercise, which were implemented with edges and vertices. If I was to do this again, I would likely make the nested static classes also implement Iterator so that I would have less verbose iteration. As I implemented DFSActions for Hierarchy and ParenthesizedList, I definitely felt more comfortable with the algorithm for DFS. At first when preparing for this project, I wanted to use a stack to traverse the graph, but now I see that recursion is a better (and now more comfortable) choice.