CMSC 350 Project 4

The fourth programming project involves writing a program that accepts information contained in a file about the class dependencies in a Java program and creates a directed graph from that information. From the directed graph, it produces two different kinds of displays of those dependency relationships.

A sample input file is shown below:

```
ClassA ClassC ClassE ClassJ
ClassB ClassD ClassG
ClassC ClassA
ClassE ClassB ClassF ClassH
ClassJ ClassB
ClassI ClassC
```

The first name of each line of the file is a Java class upon which other classes depend. The remaining names are the classes that depend upon the first class on that line. The first line of the above file, for example, indicates that `ClassA` has three classes that depend upon it, `ClassC`, `ClassE` and `ClassJ`. A class that have does any classes that depend on it, need not appear at the head of any line.

The main method in the class for this project should allow user select the input file from the default directory by using the `JFileChooser` class. It should then add the edges to a directed graph that defines these class dependencies.

A second class, `DirectedGraph`, should be a generic class, whose generic parameter specifies the type of the labels that are associated with the vertices of the graph.

It should contain a method that allows edges to be added to the graph, which is how the main method will initially build the graph. It should also contain a method that performs a depth-first search of that graph. The pseudocode for that search is show below:

```
depth_first_search(vertex s)
  if s is discovered
    perform cycle detected action
    return
  perform add vertex action
  mark s as discovered
  perform descend action
  for all adjacent vertices v
    depth_first_search(v)
  perform ascend action
  mark s as finished
```

When the method in the `DirectedGraph` class that initiates the depth first search is called, it should first initialize all the vertices to the undiscovered state and begin the search at the vertex that corresponds to the first name in the input file.

Another method in the `DirectedGraph` class should then allow the main method to display any unreachable classes by examining all the vertices of the graph to see which remain undiscovered.

This project should contain a generic interface named `DFSActions`, whose generic parameter again specifies the type of the labels that are associated with the vertices of the graph. It should contain four method signatures that correspond to the four actions performed in the depth first search: cycle detected, process vertex, descend and ascend.

There should be two additional classes that both implement the aforementioned interface. The first, `Hierarchy`, should produce a hierarchical representation of the class dependencies. Circular dependencies should be flagged. For the above input file, the following hierarchical representation should be produced:

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

The asterisk after `ClassC` results from the fact that `ClassC` depends upon `ClassA` and `ClassA` depends upon `ClassC`. The `Hierarchy` class should override the `toString` method, which should return a string that contains the above, after having performed the depth-first search.

The other class that implements the `DFSActions` interface should be `ParenthesizedList`. It should produce an alternate representation that is also returned by its `toString` method. For the above input file, the following hierarchical representation should be produced:

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

The main method should produce both representations. In addition it should display the unreachable classes by calling the previously mentioned method. For the above input file, the following unreachable class should be identified:

```
ClassI is unreachable
```

Code duplication should be avoided. In particular, the depth first code should not be duplicated.

You are to submit two files.

1. The first is a `.zip` file that contains all the source code for the project. The `.zip` file should contain only source code and nothing else, which means only the `.java` files. If you elect to use a package the `.java` files should be in a folder whose name is the package name. Every outer class should be in a separate `.java` file with the same name as the class name. Each file should include a comment block at the top containing your name, the project name, the date, and a short description of the class contained in that file.
2. The second is a Word document (PDF or RTF is also acceptable) that contains the documentation for the project, which should include the following:
    a. A UML class diagram that includes all classes you wrote. Do not include predefined classes. You need only include the class name for each individual class, not the variables or methods
    b. A test plan that includes test cases that you have created indicating what aspects of the program each one is testing
    c. A short paragraph on lessons learned from the project