Inheritance Hierarchy

- IO
  - LexArithArray
    - Parser
      - Interpreter
- Obj
  - Val
    - IntValue
    - FloatValue
  - AssignmentList
    - Assignment
    - MultipleAssingment
  - E
    - SingleTerm
    - AddE
    - SubE
  - Term
    - SinglePrimary
    - MulTerm
    - DivTerm
  - Primary
    - Id
    - Int
    - Floatp
    - Parenthesized
- CountObjects
- DFT

About Depth First Traversal and Data Structure

The objective of project 2 was to implement the functionality of depth first traversal for all connected *Obj* objects in object graph spanned by reference pointers starting from *this* object. It was achieved by using Reflection API which allows access to fields in the classes. Each fields were converted into objects and their fields was also extracted in recursive manner. Some fields had to be collected from their superclasses which was achieved recursively in similar fashion as was done in project 1. Depth first traversal was achieved in recursive way also. Every field objects extracted from the class were stored in neighbour list for that node. After the *this* object was marked visited, the same thing was done for the first child of the node but this time all the possible paths inside the first child was exploited in depth first manner before going for the second child. Marking a node *visited* allowed to not make the visit to same node twice. In that way, every object ever instantiated after *this* object was reached via reference pointer and was counted which is also shown in the output file of this project along with outputs achieved in the first project.

The objective for the first project was to implement the data structure that maintains the *Obj* and all its descendant classes and outputs number of times a *Obj* and its descendant classes was instantiated in descending order. For that, *CountObjects* class was implemented which had a method named *getSuperClass* which takes in class name as argument and recursively gets all its superclasses. Whenever new class name is seen, a new entry with value 1 is pushed into hashmap. However, if the class is already present in the hashmap, its value is increased by one.

Since classes were maintained in a hashmap, sorting was not easy to achieve. However, I implemented a function which takes in any HashMap and returns LinkedHashMap in sorted order. It does so by first breaking key and value of input hashmap into two separate ArrayLists. Then, using Collections class, the second ArrayList containing value is sorted in reverse order. Then for each sorted value, its key is found and pushed into LinkedHashMap. This LinkedHashMap is then iterated in the main to get each classes and their number of repetition in sorted order.