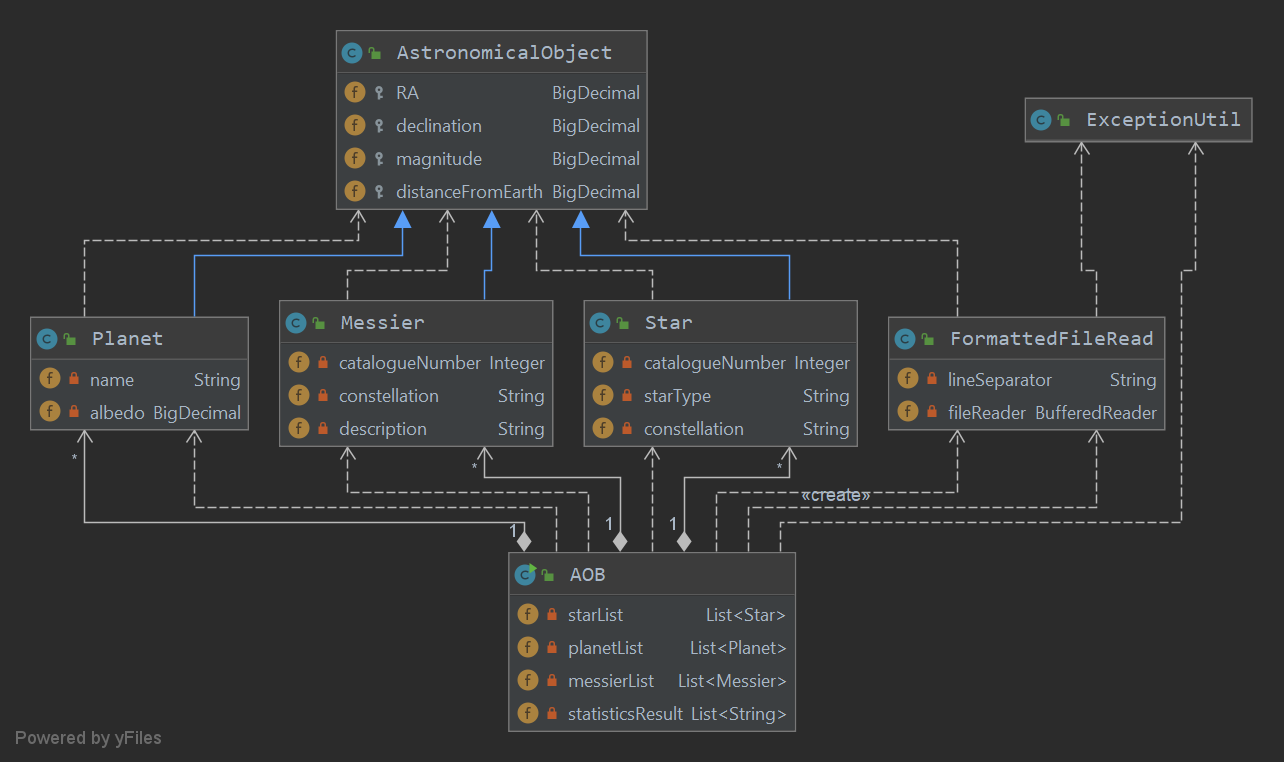
**Stage1**

# UML Class diagram



# Design justification

The AOB class is the entrance of Stage1 and demonstrates all relevant methods.

Astronomical objects are implemented under an inheritance hierarchy. Superclass AstronomicalObject classcontains all common attributes of all astronomical objects. Subclass Planet, Messier, Star extend these common variables and define their exclusive characteristics at the same time.

FormattedFileRead class is a generic class. It can read formatted files which contain information of astronomical objects and generate corresponding objects, storing them into a list.

ExceptionUtil is used to print exception information in a uniform style and exit the program.

# Justification of choice of collections

Choosing ArrayList as containers to store astronomical objects have three reasons:

1. List can add elements faster than HashSet and TreeSet, because it does not need to calculate the position of new element in this collection. Specifically, HashSet should calculate new element’s hash value to decide its position and TreeSet should compare new element with other old elements to find appropriate place for new element.

2. ArrayList expends less memory than LinkedList, when they save the same number of objects. ArrayList uses array as a base structure. Meanwhile, LinkedList uses linked nodes as its foundation. As a result, ArrayList can avoid the cost of creating nodes and maintaining nodes’ connections.

3. Java API affords a sort method in List which can satisfy the sort demands in following questions. Moreover, the time complexity of this sort method is O(n lg(n)), so it can finish sorting task in a reasonable time even with a large scale of data.

Additionally, I use HashMap to count the number of members of different constellations, because the same constellation has the same hash value and it can be found in HashMap at a O(1)time complexity. Apparently, the efficiency is still affordable with the increasing of objects.

# Summary

Based on my own experience, the most useful knowledge I learnt from first 4 weeks is how to analyse the complexity of algorithms and use it into practice. Specifically, time cost and memory cost are two important dimensions to judge a program. However, I always focus too much on solving a problem rather than thinking about what kind of algorithms can be more efficient solutions. Indeed, this is an indispensable part of programming and I should consider it as a common criterion in my future study.

Another important concept I learnt from our lecture is “we are all lazy people”. Remembering this sentence is an effective method to remind myself to keep thinking about the structure of my program. Is there any redundant code? Can I use polymorphism to improve it? That is a simple clause but can make an important role to help me form a right thinking of program.