Duy Huynh (Group 2 – HuSCII)
TCSS 360 – Spring '15
4/29/2015
Object-Oriented Heuristics Group Evals

# Group 7 Class Evaluation using OO Heuristics:

**3.1 – Distribute the work-load of the system evenly across several classes rather than lumping the majority of responsibilities into one god class.**

There doesn't seem to be one particular class that dominates over the rest in terms of the number of behavior. The Job class looks larger than the rest but this is due to its many accessor methods for use with the JobSchedule class.

**3.2 - There should not a class that handles all or the majority of the work in the application. This would be a god class which goes against decentralization and modular design of object-oriented programming.**

The Job class looks busy but if you take away its accessors, it reduces the class done to only displayJob(). Then looking at the JobSchedule class, it seems to have more complexity than the other classes because it is handling jobs and also has methods dealing with volunteers. This class may be revised to consider encapsulating the volunteer methods elsewhere.


**3.3 - Object-oriented design aims to keep related data and behavior modular so be wary of classes that have too many accessors. This may indicate the data should be encapsulated elsewhere.**

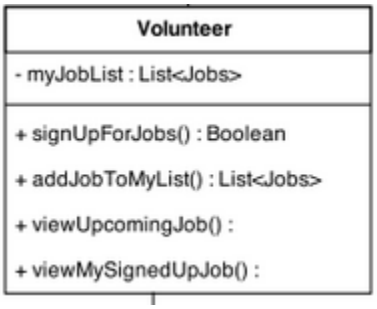

All but one method of the Job class is an accessor of some sort. This could indicate the data could be housed elsewhere, namely the Job class. There are no indications whether there are many of the Job class and just one JobSchedule since this may be the intended implementation so that would be something to look into.

**3.4 – A class should be encouraged to play with others and not just with itself. Namely, a class that does not relay or retrieve data to others and only performs operations on its own fields is susceptible to being a god class.**

The pairing of the JobSchedule and Job class may present itself as a potential god class candidate. Namely, the methods that look suspicious are getMyVolunteeredJob() and cancelMyJob() which might be better encapsulated in the Volunteer class quite possibly.

Duy Huynh (Group 2 – HuSCII)
TCSS 360 – Spring '15
4/29/2015
Object-Oriented Heuristics Group Evals

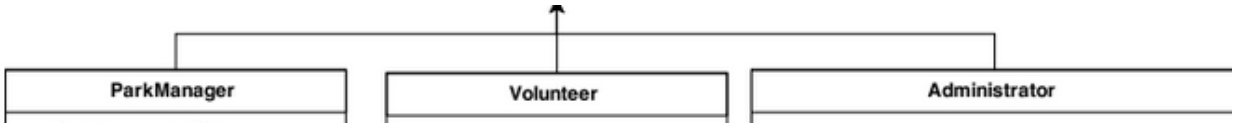
**3.5 – Model class design should not rely on the user interface but instead the user interface rely on the model class and accessors to retrieve or change the data of the model.**




There doesn't seem to be any apparent user interface implementation in this class diagram. There are some methods that might suggest some sort of user interface implementation such as viewMySignedUpJobs(), viewUpcomingJob() of the Volunteer class. My suspicions are these classes display the data so it might be better to either rename the methods if this was not the intention or to refactor them so they only represent the model data. This way, the user interface will rely on the model data to retrieve and display instead.

**3.6 – Attempt to design classes based on real-world systems or objects. This will encourage modular design that leans towards the object-oriented paradigm.**

The class diagram does a good job at representing the user roles as classes in the system (Park Manager, Volunteer, and Administrator). This helps encapsulate the data and behavior of each role. Job is also a suitable real-world object.




**3.7 – Dispose of classes that have no meaningful behavior in your system to avoid clutter. Perhaps the guilty class can instead be converted into an attribute of an existing class to promote encapsulation.**

I cannot find any one particular class that could be cut out without interfering with the system.

**3.8 - Keep your application lean by removing unneeded classes that have nothing to do with the system at hand. The proliferation of classes makes maintaining the application undesirable and convoluted.**

There does not seem to be any extraneous class that could be removed from the system. Each class represented in this diagram has a purpose and has behaviors that link them to other classes.

**3.9 - Watch out for classes that has method behavior that only serves one purpose in your system. These classes tend to have verbs in their names and are better encapsulated as methods in existing classes.**

I do not see any particular class breaking this heuristic. The Administrator class does look empty but that is only due to the assignment restrictions and should not be considered for removal.

Duy Huynh (Group 2 – HuSCII)
TCSS 360 – Spring '15
4/29/2015
Object-Oriented Heuristics Group Evals

**3.10 - Agent classes are those that help mediate between other classes but only use them sparingly in cases when it would decrease the complexity of the system.**

The JobSchedule class looks like the mediator between the three user roles and the Job class. There is sense in using the JobSchedule class to communicate between the various data fields of Job because it helps decrease the complexity of the system. JobSchedule acting as an interface, allows the three user classes to communicate with the data fields of the Job class easily. The heuristic advises to use this mediator-type class sparingly, which this class diagram does.