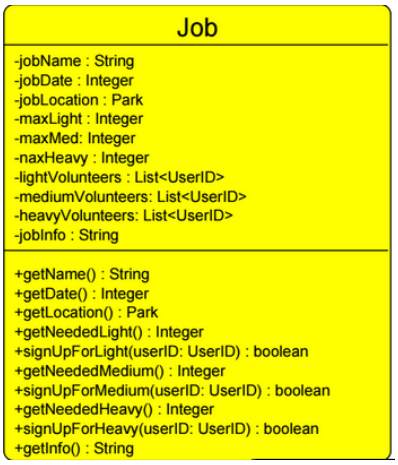
Duy Huynh (Group 2 – HuSCII)
TCSS 360 – Spring '15
4/29/2015
Object-Oriented Heuristics Group Evals

Group 4 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.

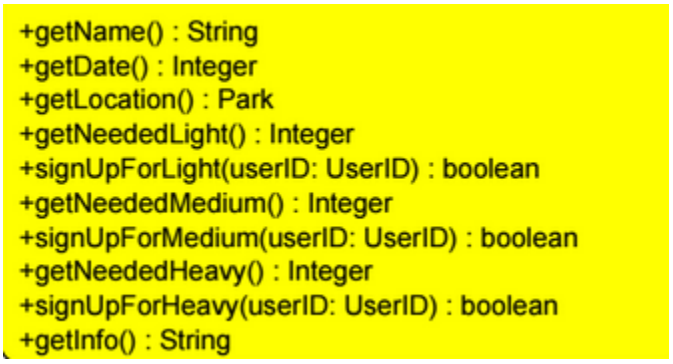


At first glance, the Job class stands out as being a potential candidate for being a god class just because there are so many fields and methods to it. Upon closer inspection however shows that most the methods in Job are accessors to its many fields. Scanning the rest of the classes, the top-level classes are sharing work equally without one dominating the rest.

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.

There does not seem to be any one class that does far more than the rest. Job does noticeably have more methods than the others but those are only addressing its own internal data as oppose to behaviors that interact with other classes.

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.



The Job class does have many public accessors for its internal data fields but these are for the Job Handler class to use. Since there are many Job instances in this particular class diagram, it would not make sense to encapsulate data fields of Job into the Job Handler since there is only one Job Handler.

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

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.

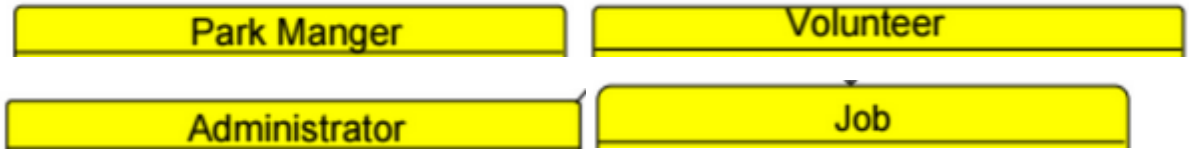
All classes have data fields to be communicated to each other and have behaviors that retrieve or send said data. Job has plenty of public accessors for its data fields.

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.

I do not see any indicators of an interface in this class diagram. Wherever I see accessors for data in the model classes, I can tell that the interface will be the one relying on the model to extract and display the information, as it should be.

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.

This class system does a good job at encompassing the three users as classes which represent actual roles in the real-world. The Job class itself is also a good representation of a Park Job because it has all the data that a tangible Park Job would have as if it was posted in the classifieds.



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.

Each class in this system is needed to complete a user story so I see no reason to remove any of them.

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.

I do not see any classes that could be eliminated without damaging the current design of the classes. Each class has its own particular role. The User class doesn't have any behaviors associated with it but it looks to be an abstract class that the three user roles are inheriting from, so I do not see this as an issue.

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 violations of this heuristic in the class diagram. The Administrator class is looking sparse but I believe that is the result of only having one user story for Administrator in this assignment.

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 agent class in this system would be the Job Handler class which plays a large role in mediating between the three user classes and the Job class. This mediator class looks to decrease the complexity and doesn't look like it could be encapsulated within the Job class since the system has many Job instances at any given moment while there is only one Job Handler.