**Peer Review Haofei Yan**

**Architecture**

While reviewing the source code, I found that in the model class *Server* handles responsibilities such as printing messages to the user, which should be handled by the view.

Since the model class has been delegated the responsibilities of the view and controller, and the controller does is handle events, I spent some time looking for actual business logic, which I must conclude have been weaved into the model class(es). It is difficult to navigate through and get an overlook of the source code; however I would conclude that the degree of coupling is low. Therefore, as the model writes messages to the user, it is quite hard to change the interface.

On a brighter note, the requirement of a unique identifier for each instance of the member class is fulfilled, using the class java.util.UUID[[1]](#footnote-1).

**Code quality and implementation**

Unfortunately the naming of variables is poor. For example, letters are used for variable names and generic names are used for methods, such as *doSomething*, which does not tell the next developer anything about what it contains or does. This is perfectly illustrated by this snippet from the *Main* method in *Main*:

“model.Server s=new model.Server();

view.Console v=new view.Console();

controller.Controller c=new controller.Controller();

while(c.Dothings(v, s));”

Also the generic name boo in the *Server* class is very insipid and does not tell me much about what it does, all I could gather is that it is a boolean:

“if(boo==true){

list.set(position, choose);

} ”

Furthermore types are also used as variable names, such as string *str*, also foundin the *Server* class. I could not, however find duplication or dead code, which one might attribute to the unreadable state of source code. To make matters worse, the source code has not been commented at all, which decreases the readability further.

**Design Quality**

On a more positive note again, data is encapsulated, but the model class Server has too much responsibility, and the controller is very small, from which I conclude that the cohesion is low. It is object oriented, however the quality of the implementation is poor.

**Documentation**

The class diagram and sequence diagram may be principally and factually correct, but do not give much further value to a developer who might wish to continue, expand upon or change the system. Also, only the creation of and listing of member(s) are shown in the sequence diagram, but it does not give an intricate description of the system, and does not provide a proper overview.

**Conclusion**

In conclusion, the MVC pattern rules are clearly violated, the coupling is low and most of the responsibility of the application lies in the *Server* class, which makes decreases the readability – which already suffers from poor naming and commenting, further. The documentation was of little help, which increases the difficulty of changing the interface, or updating the system for example. Therefore, I sadly must conclude that I do not think the grade 2 criteria are fulfilled.

**FROM:**

Henric Gustafsson

hg222dv

1. Class UUID (2015-10-08) http://docs.oracle.com/javase/7/docs/api/java/util/UUID.html [↑](#footnote-ref-1)