Peer review of Group 4 (done by Group 18, JGMOR)

Do the design and implementation follow design principles?
In the class Movement, the function addPoint both manipulates code and returns a value which goes against Command-query principle. Then, there are a lot of redundant dependencies within the Controller package as well as a lot of dependencies going both in and out of the package. This results in cohesion to be lower and coupling to be higher, opposite of what is preferred.
As far as the SOLID-principles go they are followed to a great extent. An abundance of getters and setters might be troublesome when it comes to Open-Closed principle. Could lead to unwanted dependencies and unexpected changes to variables.

Are design patterns used?
MVC, Observer and Module patterns are used.
Singleton and Facade patterns were attempted but weren't implemented properly.
No Singleton structure actually exists in the code.
Facade pattern fails due to dependencies going past the facade, rendering it superfluous.

Does the project use a consistent coding style?
Yes, code chunks are well-organised. The code style is consistent. The comments are found above the code. No redundant methods or spaces that pollute the styling.

Is the code easy to understand? Are proper names used?
It is fairly easy to understand what the code is supposed to do due to usage of java docs comments and clear documentation in RAD and SDD. Although the supposed function of the 5code is easy to understand it can sometimes be hard to grasp what the code is actually doing because of unclear method names. Some method/variable names were hard to understand what exactly they were for. An example would be the variable markedPoint in the class Game, we felt it was unclear whether this was for the spot that the player wanted to move, the selected piece or a piece that could be 'killed' by a move. This confusion was then amplified by the fact that there was another variable called clickedPoint which was used in conjunction with the previously mentioned markedPoint.

Does it have MVC structure, and is the model isolated from the other parts?
The project has a MVC structure which mostly follows the rules of a typical MVC structure. The model is isolated in its own package and from what we can see only reliant on the two interfaces Observer and TimerObserver. The first problem we found with the model and MVC structure as a whole are some unnecessary dependencies between the model and the controller. The controller has dependencies on over half of the classes included in the model which could easily be reduced. The second problem is that the controller includes functionality which is supposed to be found in the view, namely a draw function. Instead of the controller telling the view to draw it just draws images itself.
Is the code documented?
Kind of, it's well commented, but it doesn't have proper documentation in the form of JavaDoc or similar.

Does the code use proper abstractions?

The command package is well written, as it is easy to remove/add new commands and incorporate them into the design without having to rewrite a lot of code.

Is the code well tested?
Yes. There are plenty of tests (especially for the model package) that cover the most essential parts of the code.

Is the design modular? Are there any unnecessary dependencies? Is the code reusable?Is it easy to maintain? Can we easily add/remove functionality?

The program is designed with such a concept of modularization, but we can also observe unnecessary redundant dependencies in the controller package. The code is largely reusable and it can reasonably be possible to expand the program's functionality.

Are there any security problems, are there any performance issues?
The program could not be run with Maven via command prompt. Upon further inspection the pom.xml appeared to be missing a plugin for javafx, meaning the project could not be run.

Can the design or code be improved? Are there better solutions?
- All classes and interfaces should be organized into packages. Observer and TimeObserver do not belong to a specific package.
- User Story #006 does not fulfill the functional criteria of adding time to the timer when the player finishes a turn even though the User Story is marked as implemented.
- There is not a User Story including anything in regards to losing the game when your timer runs out.
- User Story #014 mentions being able to move all pieces according to the rules of chess, however the moves *castling*, *en passant* and *Queening* cannot be performed. Admittedly the User Story is named “Apply Basic Rules” so it might be clever to add another User Story for these moves.
- The arrows for the design model might need looking over. For instance in Figure 4 of the SDD the usage dependency arrow from MenuController to ChessController should be an association arrow.
- The initial sketch shows chess coordinates by the side of the board however they are not implemented and no User Story for implementation exists.
- The library java.awt is used in Board, Game, Movement, Ply. There is not however any mention of java.awt in the pom.xml-file meaning there are not any dependencies on the library. In addition to this javafx is also used which fills all the same responsibilities. Therefore, removing all dependencies on java.awt and focusing on using just the one library is better and more efficient.
- There is no mention of the libraries used in the reference section of neither the RAD nor the SDD.
- ChessTimer might need some looking over. There are two instances of the class yet, by reading the code and functions of the class, one gets the impression that there should only be one.
- There are a few inconsistencies when it comes to the implementation of the design pattern MVC(A). A lot of the responsibility that View should have is instead put on the Controller for instance the function drawPieces().