Peer-review

Group 24 - Fyra eller högre

By group 10 - AMWE

We can tell that it is far from finished, terminal instead of GUI, and todo's exist in the code as well as the documents. But we like your intended GUI look. And think you have good abstractions overall.

You have a domain model that is not up to date and missing classes. Some arrows are missing direction, and the model package consists of only a single type of arrow (Is-A). In the UML both enums used are not titled by name of the enum, but by the different value names.

The User stories are not formatted in a correct fashion and "Start a new game" and "Play the game" are marked as implemented even though not all criteria are met.Furthermore, we think the project is lacking in the amount of user stories. Two user stories for a group of four over 8 weeks seems very thin.

The code submitted had one syntax error, in menyStyle.css, but the class in question is currently unused so the code compiles.

We note some contradicting coding choices in the same file. There are a lot of contradictions which seem to stem from the decision to make the application work in a terminal. A lot of methods not relevant for that have dead or unmaintained code. The group members might benefit from more communication instead of leaving code comments. Some parts are reusable, like Card and Player. But for the most part there are contradictions and dead code which can complicate reusability.

The game class does not follow SRP which means the code is harder to maintain because of possible unexpected side effects. To fix this the class needs to be divided into several classes so every class only has one reason to change.

Some ints in for-loops, while-loops and Lists, seen in Game and Leaderboard are hard coded. This means a change of the existing functionality would make the code harder to maintain. To solve this, use variables instead.

Functionality cannot easily be removed or added, because of exposed class variables and inconsistencies. Example: the Game class both assigns health to the player's life and calls a method decLife() from the player class to change the int.

When it comes to design patterns we only found one at the current iteration of the program is MVC but without real use at the moment since View or Controller are not used. Observer and Facade design patterns are not yet implemented even though it says so in RAD.

The code is to some extent documented but not all methods or classes that need them. Furthermore the documentation does not follow the conventions of JavaDoc which is industry standard. Some methods have comments that don't describe their behaviour, only describe when they're called. Comments also vary between English and Swedish which is not consistent.

Choices of names in the application are overall good with some exceptions. E.g. the method name equals() in Card.java is maybe not optimal and should be changed to something more telling of its behaviour.
The code uses proper abstractions where needed. Like ArrayList generalized to List in method parameters.

The tests of the application cover 70% of classes and 54 % of methods in models. Which is reasonable at this stage of development, but still leaves room for improvement.
Some issues, both performance and security, were found. Performance issues are hard to judge at the moment since the program only uses the terminal at the moment. But one specific example which hinders the performance is using TimeUnit.Seconds.Sleep() to regulate game flow. Players instance variables are not private and are even accessed directly by code in Game, which is a security issue and does not follow Open Closed Principle. It is easily addressed by making them private.

The code is quite hard to read at the moment since there are a lot of comments and system.out.println statements inside methods. Since the Game class has too many behaviours it is also hard to understand.

There isn't isolation in the app currently, the model is currently both printing messages to view/terminal and handling user inputs. But this issue should be solved when the application catches up with the domain model, and implements MVC.

The way Board and Card are dependent on the Enums Shape and Color could probably be improved. It is less than ideal that both classes have that dependency since Board already has a dependency with Card.

The enums Shape and Color work well at the moment and might do so for the production cycle of the program. Adding behaviour to a Shape or Color would be hard. It would follow Open Closed Principle if they were separate classes instead of an Enum but this should be seen as a very minor critique since we as a group don't see why behaviour would be needed.

We think that your plan to implement the Facade, Observer and Strategy design pattern, is well thought out and would help the design of your program.