Christoffer Lindkvist
001101-2838
D7032E Home exam

D7032E questions:

1. **Unit testing** The code doesn't check for players when starting the game in its vanilla state, which is a violation of requirement 1 as player count can only be 2-5 players in a game.
   Requirement 3a s broken as defuses are added into the deck like:
   6-numPlayers which violates the 2 defuses in the deck for 2-4 players and 1 defuse in the deck for 5.
   11g is also broken and doesn't let one target a player and request a card with three. It always returns "not a viable option".

   Testing:
   All variables are public, and therefore can be accessed directly.
   However there will be problems when testing 1. as new players are created upon connection, as well as 4 & 7 because without modifying the code we can't get the state of the deck before and after the shuffle, so we can't see if it shuffled between.
   The problem with testing a lot of these requirements is that in the current state there are not a lot of return values to test for in JUnit, hence you gotta write a lot of checks and jump through a lot of hoops to get desired testing results. It is simply not built with testing in mind.

Christoffer Lindkvist
001101-2838

2. **Quality attributes, requirements, and testability**
   The requirements on 17-18 are poorly written because, as stated previously; you cannot do unit testing properly as it doesn't have a lot of returns and the code is a disaster as a whole, you can also not run/test parts of the code without running the entire code, therefore you cannot observe state changes in one single part testability suffers due to this, and because of the way it is structured you cannot easily add more cards or functionality without rewriting huge chunks of the code. so both modifiability and extensibility suffer as well.

Christoffer Lindkvist
001101-2838

3. Software architecture code review:

   The code from a Modifiability standpoint: is bad to put it lightly. All values of the game are hard coded which is not too bad, but if I want to play with 3 seconds nope time instead of 5 I have to jump in and find the nope time in the code and recompile. Or if I want to make Attack hit for 3, I gotta jump in and find the attack prompt and action. Or if I want to remove all shuffles from the game I have to jump in and change it and recompile.

   The code from an Extensibility standpoint: is horrible as the entire code is incohesive and insufficient. Cards and cardactions are handled all over the place in the code. If I wanted to add a new card, I'd have to rewrite a ton of code, and add it to the cards enum, then add an additional for-loop to the deck to include my card, and then go deep down into the Game class and add our card as a viable option AND write the function in the already cluttered monkey code. This would be somewhat fine if it was cohesive, all in one place; or if it was sufficient, handled in separate classes; but instead you got to find every line affected in a mountain of incohesive and insufficient code to handle it.

   The code from a Testability standpoint: is awful. As stated in question 1, there are no returns to test against, nor can a lot of requirements be checked easily without modifying the already existing code. A lot of work has to be put in to make the code even remotely testable for all 13+ requirements as the code is extremely coupled.

Christoffer Lindkvist
001101-2838
D7032E Home exam

4. Software Architecture and refactoring:

The classes of the program:
Player: The class player is pretty much just taken from the source code and put into a separate file with additional return functions and proper variable encapsulation. For example, instead of going Player.hand you go Player.getHand() which returns the hand.

Game: Game has in the new and improved version been split up into several classes and smaller more primitive functions. Unfortunately there is a bug which crashes the game if someone that isn't server passes the turn. The other classes that spawned from the original game class is CardAction, as well as Discard and Deck. Game also communicates gamestate to players and handles currentPlayer, which also could get split up into smaller primitive classes, in hindsight. While better it still suffers from a lot of coupling.

Discard: Discard is very primitive, featuring only and arraylist, and functions to add a card to discard, and to check the amount of nopes in the top cards.

Deck: The deck design and creation is handled in here, which in hindsight could have gotten split up into a deck factory reading JSON values, however I never got to that so the deck is using hard coded values for the cards and options, but builds it more efficiently as you may add cards without needing to poke around in the main deck building function. From an Extensibility standpoint it's better than before, but from a modifyability standpoint it is more or less the same as the values are hard coded as a "temporary" solution.

Card: Cards are primitive as well, as not a lot happens in them. We have the construction which specifies cardtype from the enum, and a function that returns type and does the action. the action function calls on the CardAction class with the cardtype as parameter.

CardAction: Here is where most of the gameplay is handled, as all actions from the cards are handled with new instances of the cardaction. This is great for extensibility, and it's more cohesive than before as now if you want to add a new card you can easily find all places in need of modification to add it.

Client/Server: These are the same, I just put them in separate classes.

ExplodingKittens: Once home to all of the functions just starts up the necessary classes.