Andreas Bom
ab22cw

Peer Review

Workshop 3

Review on:

Adam Lundberg, al222jf

Mattias Johansson, mj223dg

Try to compile/use the source code provided. Can you get it up and running? Is anything problematic?

There is no problems to get the application up and running.

Test the runnable version of the application in a realistic way. Note any problems/bugs.

Problems:

- If dealer gets two high cards, let's say a king and an ace, and the player stands on a lower score, the hidden card that the dealer has is not shown, and the score is not calculated correctly. It looks like the dealer wins with a score of 10, when it is actually 21 (the Ace is not included in the score calculation.
- GameOver is printed three times.

Does the implementation and diagrams conform (do they show the same thing)? Are there any missing relations? Relations in the wrong direction?

- The Soft17Strategy is missing the dependency to RulesFactory.
- NewStrategy is an abstract class, and is inherited by AmericanNewGameStrategy and InternationalNewGameStrategy. The correct uml is a solid line with an arrow like a triangle[1].
- The IEqualScoreStrategy is showing a dependency to Dealer, but I can't find this dependency in the code.

Is the dependency between controller and view handled? How? Good? Bad?

The dependency is handled by using an Enum. This is, in my opinion a good way to handle the dependency.

Is the Strategy Pattern used correctly for the rule variant Soft17?

Yes it is working correctly.

[1] Larman, C (2005). *Applying UML and Patterns (Third Edition)*. Upper Saddle River: Prentice-Hall. Chapter 16:1 Applying UML: Common class diagram notation.

Andreas Bom
ab22cw

Is the Strategy Pattern used correctly for the variations of who wins the game?

Interface and classes are provided, but not implemented. I don't know if that's a mistake or if there were any problems with implementation. Here's how I did it to get the strategy to work:
In Dealer:
Create a new private variable:

```
private rules.IEqualScoreStrategy m_winnerRule;
```

In construct:

```
m_winnerRule = a_rulesFactory.EqualScoreRule();
```

In method IsDealerWinner
Delete everything and write:

```
return m_winnerRule.winner(a_player, this, g_maxScore);
```

Is the duplicate code removed from everywhere and put in a place that does not add any dependencies (What class already knows about cards and the deck)? Are interfaces updated to reflect the change?

The duplicated code is removed and put in an abstract class.

Is the Observer Pattern correctly implemented?

The player is the Subject that publish stuff (cards). The Observers is the controller, and on each card that is dealt, a pause in the game is made. I think that the bug that the GameOver is printed 3 times is due to this implementation, but I can't find what is wrong. A small notice, the DealtCard method in Player should not use `System.Threading.Thread.Sleep(1000);` This pause in the game should be handled by the view.

Is the class diagram updated to reflect the changes?

The diagram shows that the Player is implementing IBlackJackOBserver, which is not correct. There is a dependency between these, but not a realization.

Do you think the design/implementation has passed the grade 2 criteria?

I think the diagram has a few things that is not correct, maybe this should be corrected. The Strategy pattern for who wins the game, was not implemented. All the needed classes and interfaces were provided, so after a small change in the code, the pattern worked. The application had some minor bugs, but all in all, I think it passes grade 2 criteria.