Peer Review

review by Olga Christensen (oc222ba)
of work by Lars Wöldern (lw222i) and Nicklas Björkendal (nb222gp)

I haven't tried to compile and run the code since I don't have the Visual Studio and I didn't have time to install and study it. I have examined the code and here are my comments.

1. Implement Game::Stand

Implemented according to the Sequence Diagram.

2. Remove the bad, hidden, dependency between the controller and view (new game, hit, stand)

To solve this an `Enum` class `ViewAction` was created in the `controller` namespace.
I think it is a bad idea to put it there, since our `IView` interface is now dependent on `PlayGame` controller (needs to import/use `BlackJack2.controller`).
A better idea would be to place the `ViewAction` class into the `view` namespace and even better - to nest it within the `IView` interface (I think C# supports such implementation). This way we make our `IView` independent.

Othervise the "hidden dependency" was removed.

3. Design and implement Soft 17

To check if the Soft 17 rule applies to the dealer's hand a condition is passed `if (a_dealer.CalcScore() == 17 && a_dealer.hasAce())` .
The method `hasAce()` is implemented in the `Player` class, which is good, since it is the information expert.

4. Design and implement a variable rules for who wins the game

An interface `IWinStrategy` in the namespace `model.rules` is now responsible for checking if the dealer is the winner. This allows easy implementation of new winning strategies.

Since the value of `maxScore` is involved in calculating the winner, this value was made public (originally it was private), so it can be accessed directly from other namespaces (i.e. `a_dealer.maxScore`). This is a bad idea to make a private field public since it would allow other members to modify it, instead I would suggest making the field private again and passing this value as a parametr to the method `isDealerWinner(Player a_player, Player a_dealer, int maxScore)` .

5. Refactor code to remove duplication

New method `public void GiveCardToPlayer(Player a_player, bool isShown)` in `Dealer` class replaces all occurences of the duplicate code.

6. Implement Observer pattern

The interface `GameObserver` was placed in `controller` namespace which led to violation of MVC separation principle - now `model.Player` has an association relation to `controller` [1]. Hence the interface `GameObserver` should be placed in the `model` namespace.

Also it is a common practice to use a `List` to collect all observers for a certain class, since we might have more than one. So current implementation with a single possible observer may limit future code developments.

Also I wouldn't recommend subscribing an observer in the main class `Program`, I think that the `PlayGame` class is the information expert in this case. So it should be done there in the constructor.

7. Class Diagram

Class `Game` is missing a dependency to `GameObserver` interface.

Additionally

When working on some else's code it is a good idea to follow the established naming convention. In this case method's name should start with capital letter (e.g. `public void tempGamePause()`, `displayWelcomeMessageAndHands()`, etc).

References:

1. Model-View-Controller, microsoft.com

2. Larman, C., Applying UML and Patterns 3rd Ed, 2005, ISBN: 0-13-148906-2