Peer review of Sarpreet Singhs Workshop 3
by Linda Ott Olander

I tried running the program by first installing the latest Java Runtime Environment as according to the instructions provided by the group. I checked that it was installed correctly by entering java -version at the command prompt. It returned Java (TM) SE Runtime Environment, so it should work. I then navigated to the folder where the java project was located and tried to run it by entering “java Program”. I then got the feedback “Could not find or load main class”. I opened the Program.java file and saw that it was in a package, I tried to run “java BlackJack.Program”. This gave the same result. I then tried the other approach, by dragging and dropping the .jar file on the console. Nothing happened. I clicked enter, nothing happened. I then tried entering “"java -jar (jarfilelocation)\" (I of course entered the real file location with C:\.. etc). It then gave me the message “Unable to access jarfile”.
I downloaded NetBeans with JDK and created a new project, and added the files to that. I was now able to run the code and test the program.
When I type ‘p’, the code seems to run in a loop. I get several instances of Dealer Has: and Player Has:. At first it says Dealer has: Score: 0, and then Player Has: Eight of Hearts, Score: 8. Then comes another Dealer Has: Two of Hearts, Score: 2 and Player Has: Eight of Hearts, Score: 8. Then another two rounds. So the code doesn’t seem to work as intended so far, at least not when I run it through NetBeans. I now type ‘h’ to try the hit command. This seems to work better, the application now only shows one round of Dealer and Player. When I type ‘s’ to test the “stand” command, the console shows two rounds of dealer and player hands and scores. Then it shows “Game over”. So I think the code might be running the rounds a few times extra.

Does the implementation and diagrams conform
I installed NetBeans with JDK in order to view the code. I’ve been reviewing the code and the diagram, and they seem to conform well. The names “controller, view and model” are missing from the diagram packages, the little “flaps” at the top. You’ve added the names inside the packages instead. Perhaps they could be added to the “flaps” instead.

Is the dependency between controller and view handled? How? Good? Bad?.
Yes, the dependency has been removed and the functionality is implemented in PlayGame.java in the method Play() instead. The controller is still dependent on the view, since it gets the input through a_view.getInput(). So perhaps a dependency arrow should be added from controller to view in the diagram. Changes to the view in getInput() method would affect the controller (PlayGame.java). The SimpleView class is well set up concerning how it gets the input from the user, it uses try/catch statements which is good. But the controller is still dependent on the view for information. Perhaps the input could be stored in the model somehow, but the controller would still need to get the information from the view. So I don’t know how to build a better solution myself.

Is the Strategy Pattern used correctly for the rule variant Soft17?
Yes, the Soft17HitStrategy class extends the BasicHitStrategy class which implements the interface IHitStrategy. According to Larman [1, section 26.7], the strategy pattern should be

implemented by creating multiple classes, each with a polymorphic method. Each algorithm should be defined in a separate class, with a common interface. This is how this project is implemented so the Strategy pattern is used correctly.

Is the Strategy Pattern used correctly for the variations of who wins the game?
There is an interface called IWinStrategy with a method IsDealerWinner(). This interface is implemented according to [1, section 26.7] by classes DealerWinStrategy and PlayerWinStrategy. So it does seem to be used correctly in the variations. The difference is small between the classes, but it is there.

What class already knows about cards and the deck?
According to the old class diagram, there is no class that knows about both the card and the deck. The Player class knows about the Card, and the Dealer which is inherited from the Player, knows about the Deck. So I suppose the Player could be said to know both about the Player and the Dealer, but it's not crystal clear.

Is the duplicate code removed from everywhere and put in a place that does not add any dependencies?
In the original code, the duplicate code (Card c = deck.GetCard();) was present in the Dealer class in the method Hit(). This has been replaced by a method call to method GetCardFromDeckAndDealTo(). This method holds the aforementioned code and also c.Show(visibility) and a_player.DealCard(c). So this code is still in the Dealer class as before. I did a search in the original java repository for this "duplicate code", and could only find it once in the Dealer class. So where was it duplicated to begin with? Anyway, no dependencies were added.

Is the Observer Pattern correctly implemented?
In the Observer pattern, one should define a "subscriber" or "listener" interface. [1, section 13.7]. Subscribers implement this interface, and the publisher registers subscribers who are interested in an event and notifies them when an event occurs.
There seems to be an observer pattern implemented in the code, as the class PlayGame implements the interface IObserver. There is also a Subscribe method in this class that adds the subscribers to the Game instance. The Player class also has a AddSubscribers and NotifySubscribers method. It seems correctly implemented, but I'm wondering if this addition is what causes the problem with the multiple rounds when I run the code. Maybe it's something else.

Is the class diagram updated to reflect the changes?
Yes.

Do you think the design/implementation has passed the grade 2 criteria?
If the problems are fixed with the multiple rounds, I think it should pass.

References:

1. Larman C., Applying UML and Patterns 3rd Ed, 2005, ISBN: 0131489062