Workshop 3

Changes Report
by Linda Ott Olander

Hidden dependency between controller and view:
We changed Linda's solution to Ningrui's solution. Linda's solution still had a dependency between controller and view. In Ningrui's solution, the controller PlayGame.cs gets the input from the view, and stores it in a variable called input. Then PlayGame() looks in the view (can be SimpleView or SwedishView) and sees what the user has chosen. For instance, if the view shows that the user has pressed "p" (WantToPlay(input) in SimpleView returns true), then NewGame() in Game.cs is called.

Observer pattern:
In her original solution, Linda implemented the Observer pattern by having the method Update() inside of the interface IObserver. Playgame.cs implemented IObserver. In PlayGame.cs the method Update() first showed DisplayDealerHand and then DisplayPlayerHand.
We have now changed this to Ningrui's solution when it comes to the Observer pattern. In Ningrui's solution, the interface Observer.cs is located in the model. PlayGame.cs implements interface Observer.cs. In the constructor of PlayGame.cs, there is the line a_game.AddObserver(this) which sends in PlayGame.cs to Game.cs AddObserver() method. In Playgame.cs there is also a method Update() which first adds a delay and then displays the dealer hand and then the player hand. SimpleView.cs and SwedishView.cs have a method delay() that uses Thread.sleep(1000) to cause a delay when showing the hands. Game.cs has the method AddObserver(Observer obj) which calls AddObserver() on both the player and dealer instances, and sends in the Observer object into the AddObserver() method on each one. Player.cs holds a list of observers. The AddObserver() method mentioned earlier in the Player class, adds the Observer object to the list of observers. When DealCard() in Player.cs is called, it calls on method CallObservers() in Player.cs. CallObservers() goes through the list of observers and calls Update() on each of them. So Update() is called in PlayGame.cs every time a card is dealt.

Who wins the game:
This was not so well implemented in Linda's solution, so we used Ningrui's solution here also. In Program.cs, a while loop checks that choiceMode() is true in the controller class PlayGame.cs. choiceMode() in PlayGame.cs displays a welcome message and then gets the input from the view. If the user has chosen that the player wins on an equal score, then the variable _isWhoWin in PlayGame is set to true, and the choiceMode() method returns false. If the user instead chose that the dealer wins on an equal score, _isWhoWin is set to false, and choiceMode returns false. This means that the while-loop where the user chooses whether the dealer or player wins on an equal score stops, and instad ctrl.Play() is run in Program.cs.

Soft 17 Pattern:
Here we chose to use Ningrui's solution, because it is basically the same as Linda's solution.