**Single Responsibility Principle:**

In order to better enforce the single responsibility of classes, a few changes were made. First, the action listeners for CardStacks were taken out of the CardChooserDialog class. Previously, the CardChooserDialog class was responsibility for displaying the options a user has during their turn as well as retrieving and translating their input into method calls. The retrieval and translation of input is really a responsibility that should fall on the Controller. So the CardStackListener was moved into the Controller class, which now has a method to register a CardStack. With this change, the CardChooserDialog is now responsible only for displaying the options a user has on their turn.

The board class was also in violation of the single responsibility principle. Previously, it was responsible with managing turns, holding a reference to each player, and maintaining and updating the state of the board. The board class is the central class of our model, and it makes sense for it to be responsible updating the state of most model classes, but we felt that the responsibility of managing turns should really be placed in another class. On this basis, the TurtleMover class was created. It represents the TurtleMover of the actual game, which is more or less responsible for delegating turns and determining which turtle should move when a card is played. It implements the TurnManager interface (more on this in the dependency inversion principle), and contains a reference to four turtle masters. Each TurtleMaster is responsible for translating a card to a legal move for the turtle it controls, or returning illegal move if the move is not acceptable. The board no longer contains a reference to the four TurtleMasters, but rather a reference to the four Turtles. I think this makes more sense for the boards responsibility of holding the state of all objects currently on the board, as the Turtles have a position, but the TurtleMasters do not.

**OCP & Dependancy Inversion:**

One of the most substantial changes made was the addition of interfaces allowing the controller to “plug into” the model. This makes our code more extensible as new functionality can go into the interfaces as opposed to requiring re-writes of functions which are already implemented. These interfaces also function to invert dependency. Our controller now knows nothing about the model except for the three interfaces it uses to interact with it (those being BoardManager, CardPlayer and TurnManager). In principle, the implementation of anything in the model can be completely re-written without the controller needing to change at all.

**Liskov Substitution Principle**

We didn’t see any place where a polymorphic inheritance structure would be appropriate at this stage in the design, so the Liskov Substitution Principle was not used.