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, while the controller is responsible for handling input.

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, maintaining and updating the state of the board, and playing a card. 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 and playing cards should really be placed in other classes. 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.

For the functionality of playing cards, the controller simply talks directly to a class which implements the CardPlayer interface. Conveniently, TurtleMaster already had all of this functionality. It just needed to be formalized with an interface implementation and skipping an unnecessary intermediate step (previously: *CardClicked* -> Controller.onCardPlayed() -> Board.onCardPlayed() -> TurtleMaster.onCardPlayed()).

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 are also used to invert dependency so that concretions rely on abstractions. 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

The Liskov Substitution Principle is used in the CardStack Class. A CardStack is simply a JButton with a CardType. A CardStack behaves exactly how a JButton does, and can be treated as a JButton in the GUI.

Interface Segregation

At first when attempting to invert dependencies, we simply pulled everything the board needed to be capable of into one BoardManager interface. This interface was bloated, highly specific and impractical. It was then segregated into three different interfaces: A CardPlayer, a BoardManager, and a TurnManager. Segregating the interface actually forced us to adhere to the Single Responsibility Principle by pulling apart the functionality in the Board class into separate

classes (those being TurtleMover, TurtleMaster and Board).