One of the design patterns we used multiple times was the **iterator** pattern. Since we have the player and dealer's hands stored in arrays, we used iterators to traverse these arrays to do things like generate the initial values of the hand to non-card values, calculate the total value of each hand, and even display the hands to the player. Another design pattern we used was the **state** pattern, but somewhat loosely. The way we used the state pattern was by having a turn value stored as a 0 or 1 which determined the "state" of the game. If the turn is set to 0, the game will be in the player turn state, which shows the player their cards and allows them to either hit or stand. After this state, it sets the turn to 1, so that the "dealer" AI can decide whether to hit or stand, and so on. Finally, when either the player or dealer wins, the game goes to the end game state that displays who won. Another design pattern we used was the **facade** pattern. Even though the facade is usually only used for a multitude of complex classes, in our case we used this pattern to limit the user's interaction with their own Blackjack hand to be within the rules of the game. In our game, the Blackjack class itself allows callers to do things like return the number that the card is represented by (which for us is 0 to 51), but our Executive class does not allow players to see this underlying structure and will only allow them to see the name of the card in string format or the value of the card as a number. The Executive class acts as a facade, which restricts the user's interaction with the Blackjack class to only allow them to do certain things.