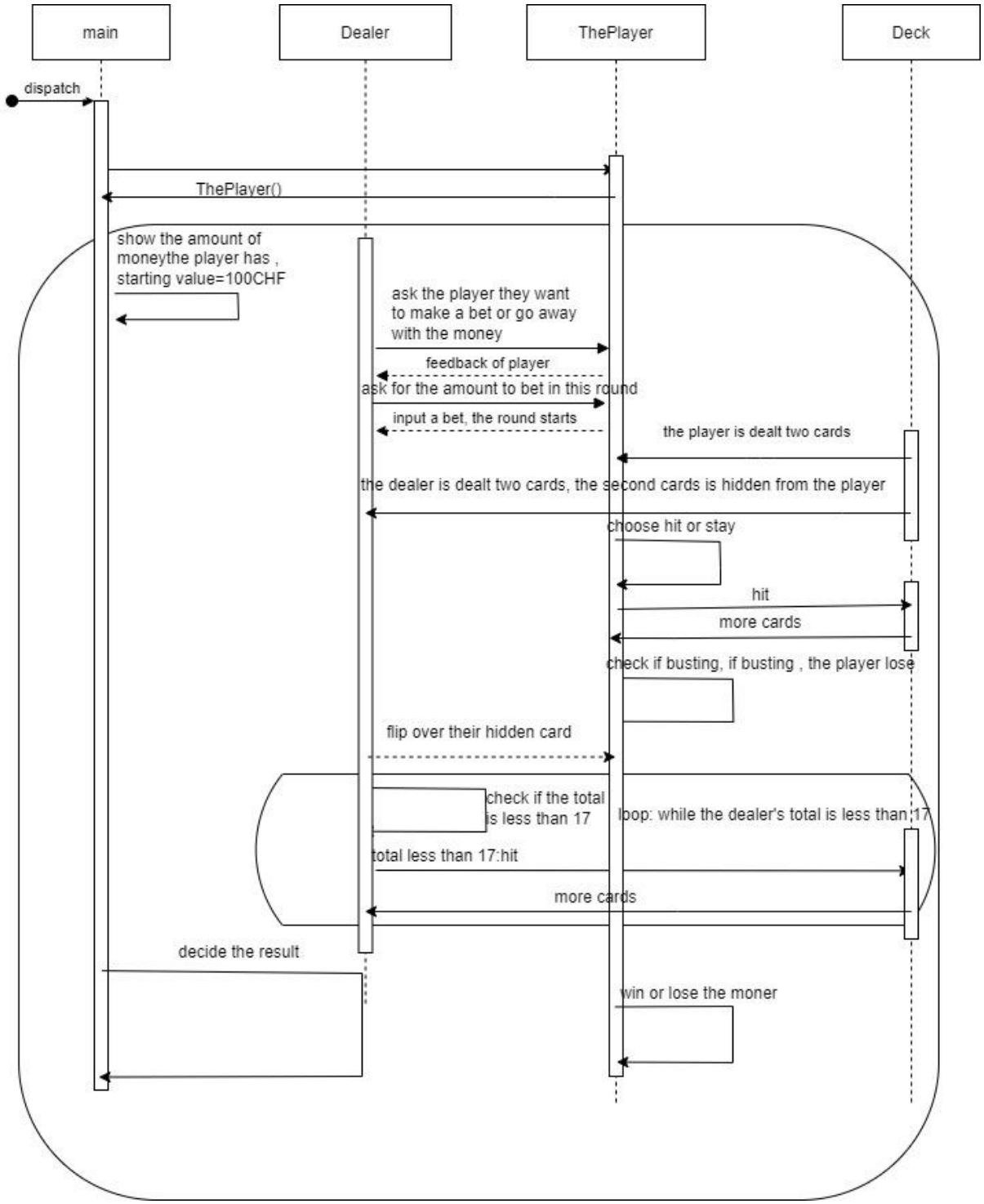
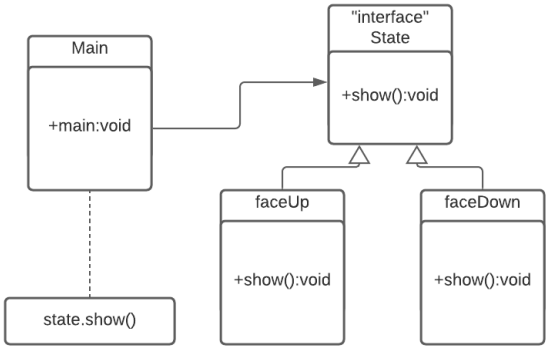


Our game running as this sequence diagram, we implemented the steps of the blackjack game in Main class.

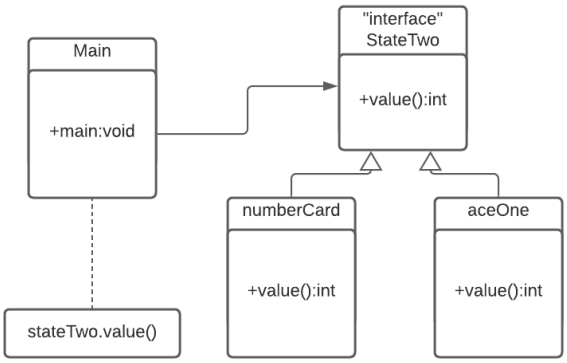
We added one step after the Dealer flips over the second card, the player has a chance to choose if double the bet money.

We decided to implement the classes Card and Deck, ThePlayer and Dealer. Due to us only being two people in the group, we decided to implement the rest of the code in the main class out of time reasons, even though we know it would have been possible to create more classes with methods to implement the steps of the Blackjack game to make the code more object oriented.

The class Card contains the attributes Rank and Suit, which are of type enum. It has methods for getting the Rank and Suit, but not for setting them, since we don't want the Cards to change after being created in the Deck. It also has the method show, which is implemented in the States faceUp and faceDown, so we could display the cards either way. We used a second State pattern for the method value, which determines the amount of points each card has. This StateTwo has two States, numberCard is the original state, which determines the value of all the cards with the Ace having the initial value 11. The State AceOne is specifically applied to the Ace Card, so it can switch value from 11 to 1 if the sum of the cards exceeds 21. This step is implemented in the main class.



Class Diagram of State pattern



Class diagram of StateTwo pattern

In the class Deck we then created all the cards and added them to an Array List. The method shuffle uses the built-in function collections.shuffle, and the method draw removes one card from the deck.

The class Dealer has an attribute integer dealerpoint and the three methods sumofdealerpoint, showdealerpoint, and getDealerpoint, which respectively compute the sum of the dealers points, display them to the user, and to get the dealerpoint.

The class ThePlayer also has the corresponding attribute playerpoint with the methods sumofplayerpoint, showplayerpoint, and getPlayerpoint. It additionally has the attribute integer amountmoney, which is the players money and is set to 100 at the beginning of the game. It also has the method sumofmoney, which returns the sum of the money. This class also has two methods to check if the users input is valid. InputValid tests the case when the user is asked to answer with 0 or 1, e.g. when asked to hit or stay, and InputValidMoney is for when the user is asked to enter an amount of money, to check if they only enter three digits between 0 and 9. It only checks the validity of the input, we check if the number entered actually is smaller or equal to the available amount of money in the main class.

In the main class we then implemented the actual steps of the game. Each round starts by asking the user if they want to bet by entering 1 or walk away by entering 0. We use the method InputValid to each time the user has to 1 or 0, and request them to enter either each time they dont. If they enter 0, the game breaks and therefore ends, if the enter 1, they are asked to enter the amount of money they want

to bet. If they don’t enter a number, they are asked to enter a number and if tha t number exceeds their available amount of money, they are asked to enter a number less than their available amount. As soon as they do, the game acctully begins, by giving the user two cards that they see as well as displaying the sum of the point s of the card. We also use an integer acesnumber1 to count the amount of aces with value 11 the player has. The same happens with the dealer, but one of the dealers cards gets displayed face down, and the displayed sum of points obviously only sho ws the points of the cards lying face up. The integer acesnumber2 corresponding to acesnumber1 is used for the dealer.

Then we use a while loop to automatically make the round end with a bust on the pl ayers side if the amount of points(int points1) exceeds 21. In this loop, the user is asked to hit or stay, again by entering 1 or 0. If they enter 1, they receive a nother card, and its value is added to points1. As soon as this value exceeds 21, there is another while loop within, checking if there are any Aces amongst the pla yers cards, which then change their state to AceOne. If points1 is over 21(if ther e aren’t any Aces) or the player decides to stay (enter 0), the loop ends. As lon g as that doesn’t happen the player will keep being asked to hit or stay and get a new card each time.

The same happens on the dealers side with points2, but the dealer hits every as lo ng as their points are below or equal to 17, then stops. Also one of their cards i s hidden. We added the additional step that the User gets to double the amount of their betted money before the dealers hidden card gets flipped over, if they have enough money available.

If points1 is over 21, the user busts and loses anyways. If points2 is over 21 but points1 isn’t, the dealer busts and the player wins the money. If both are equal to or less than 21 they get compared. If points1 is bigger than points2 the user w ins, if points2 is bigger the dealer wins, if they are the same it’s a tie. The u ser gets their betted money back when winning or having a tie.