Blackjack Design Doc

This Java program (running on JDK 1.8.0_111) has 5 classes. In this command-line version of blackjack, there is only one player and one dealer. So, the player (you) is playing against a dealer.

1. **`Player`** : the Player class is meant to represent both the
   - **Instance Variables**
     - `balance` (double) : keeps track of players monetary balance
     - `bet` (double) : keeps track of current player bet
     - `hand` (Hand) : keeps track of the player's hand; a hand is an `ArrayList` of cards
   - **Constructors**
     - Default constructor: this is used to instantiate the dealer. The dealer does not have a balance or bet to be initialized
     - `Player (double balance)` : the player (you) needs to set a balance to set.
   - **Methods**
     - `increaseBet` (double amount) : this method is used to increase the bet when doubling down
     - `placeBet`() : This method prompts the user to place a bet and does input validation and checks that the bet is valid (i.e. does the player have enough money to place the given bet). If the bet is invalid, it will prompt the user to enter another bet.
     - `newHand` : instantiates a new Hand using the constructor in the Hand class
     - `hasBlackJack`() : returns true if the sum of the hand is equal to 21 and false if it is not
     - `hasBust`() : returns true if sum of the hand is greater than 21 and false otherwise
     - `hit`() : draws a new card from the Deck object and adds that card to the player's hand
     - `doubleDown`() : calls `increaseBet()` method and `hit`() method

2. **Card** : The Card class contains two enumerated types (`Suit` and `FaceValue`) that represent all the possibilities for a standard 52-card deck. The `FaceValue` enum has a constructor that sets an integer instance variable called value that represents the blackjack value of a given card**.**
   - Instance Variables
     - `faceVal (FaceValue)` : represents the face value of a card
     - Suit (Suit) : represents the suit of a card (Hearts, Spades, Clubs, Diamonds)
   - Constructor
     - `Card(Suit suit, FaceValue faceVal)` : sets suit and faceVal of card
   - Method
     - `toString()` : String representation of a card

3. Deck
   - Instance variables
     - `cards (ArrayList<Card>)` : list of Card objects
   - Constructor
     - `Deck()` : uses nested for-loop to create all Card objects and add to the cards ArrayList
   - Methods
     - `draw (Card)` : gets the first card object from the ArrayList , removes that Card from the ArrayList and then returns the Card object itself
     - `shuffle(void)` : Uses random number generator to iterate through each card and put it in a random position in the ArrayList
     - `swapCard(void)` : a helper method of the `shuffle()` method to swap two cards

4. Hand : Represents a hand of a given player of dealer
   - Instance variables
     - cards (`ArrayList<Card>`) : list of cards in the hand
   - Methods
     - `sumOfHand (int)` : Iterates through the hand and adds the `faceVal` of all the cards in the hand

5. Blackjack : this is the driver class which pulls everything together and dictates game logic
   - **Static Variables**
     - Player `player` : represents player
     - Player `dealer` : represents dealer
     - Deck `deck` : represents deck of cards for the game
     - boolean `outOfMoney` : determines if player is out of funds
     - `Scanner` : reads in user input for game decisions
   - Methods
     - `main` method : Contains while loop based on `outOfMoney` variable and continually calls the `deal()` and `optionsManager()` methods
     - `deal()` : asks player to place a bet and verifies that the bet is valid
     - `dealNewGame ()` : Instantiates new `deck` and shuffles deck; Draws two cards for dealer and player; only shows the first dealer card
     - `getGameResults ()` : determines if dealer got blackjack, dealer bust, dealer won hand, tie, or if player won hand based on the `EndResult` enumerated type created at the top of this file.
     - `endgame()` : takes `EndResult` enumerated type generated from `getGameResults()` method and either gives or takes away money based on the hand outcome.
     - `stand ()` : includes logic for stand action in blackjack
     - `hit()` : includes logic for hit action in blackjack; relies on `hit()` method in `Player` class

- `doubleDown ()` : includes logic for double down action in blackjack; relies on `doubleDown()` method in Player class.
- `optionsManager()` : acts as interface for user to determine if they want to hit, stand, or double down
- `displayHand ()` : Iterates through card objects in a hand and prints them to console.