How the Code works:

At a high level of abstraction, the Game consists of 5 classes, all interacting with each other. Namely, these are the Card, Deck, Player, Bot, and Game classes. The game is structured so that there is a central Deck object, which holds the stock deck and discard pile, which are arrays of Card objects. The player objects also hold Deck objects within them. These Deck objects hold the total cards that the player has, as well as the hand that the player wants to play. These players (which may be humans or computers) are stored in an array object, that is looped through, from player to player, to simulate the game flow (which can be forward or backwards). At each turn, the game will ask the current player for their hand. If this is a human player, an input prompt will be displayed. If this is a Bot object, the process will be automated, and the Bot will play its hand. After a hand is played, the Game checks if the hand is valid. If the hand is invalid, the Game continuously prompts the player until valid input is provided, or the user draws a card. If the hand is valid, the Game loops through hand from back to front, reading, each card’s ability, and altering the game based on it. The hand is then appended to the discard pile, and the game goes on. The game will go on in this manner until a player successfully plays all of the cards it has, in which case that player wins.

The classes, in detail:

As mentioned before, the game consists of five classes, these will be explained in-depth below.

Card:

This is the most basic class in the game. It holds a rank, value and ability, with relevant getter and setter method so that they can be accessed in-game.

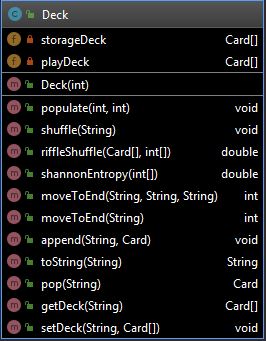


Key Methods:

* getNumber() – returns the card value
* getSuit() – returns the card suit
* getAbility() – retunrns the card’s ability

Deck:

This class holds the cards that will be in play. These cards are represented by Card objects. It can have cards removed from it, or append a card to it, depending on what is required.



Key Methods:

* append() – appends a card to the storageDeck or playDeck (i.e stock or discard pile)
* pop() – pops a card from a given location
* moveToEnd() – moves a card to the end of the deck, so that it can be popped from the
* shuffle() – this shuffles the deck
* riffleShuffle() – this is the actual riffle shuffle algorithm used to shuffle the deck
* shannonEntropy() – this calculates the Shannon Entropy of the shuffled deck

Player:

This class represents an individual human player. A Player object is the means through which any given human will interact with the game.

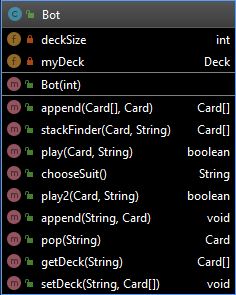


Key Methods:

* play() – this is what the player uses to play when it doesn’t have to defend itself from a pick-up card
* play2() – when a pick-up card is played, the player uses this method to defend itself, or draw the required number of cards
* chooseSuit() - the player uses this method to choose a suit, when it plays an eight

Bot:

This is an automated version of the Player class. It inherits the Player class’ methods and abilities, but is able to play automatically.

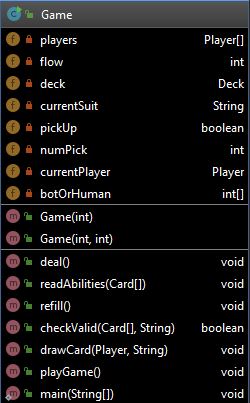


Key Methods:

* stackFinder() – this method contains the algorithm the computer uses to find the possible hands that it can play
* play() – this is what the computer uses to play when it doesn’t have to defend itself from a pick-up card
* play2() – when a pick-up card is played, the computer uses this method to defend itself, or draw the required number of cards
* chooseSuit() – the computer player uses this method to choose a suit, when it plays an eight

Game:

This is the driver for the application, this class contains a large number of methods, that fulfil the various required tasks for the game to run properly. This includes dealing cards to players, accepting player input, checking hand validity, and executing card abilities.



Key Methods:

* deal() – pops cards from the stock deck, and appends them to the players’ decks
* checkValid() – checks if a played hand is valid, based on the game rules
* playGame() – plays an individual match
* drawCard() – draws from the stock deck, and appends it to a specific player’s deck, if they have to draw a card
* readAbilities() – loops through a played hand, and activates each card’s ability
* refill() – refills the stock deck by taking cards from the discard pile
* main() – Driver for application; user can set game parameters here, and run a game