## A picture containing diagram Description automatically generatedModel UML Diagram

## These model classes are unique since it’s never directly in contact with Threads. Furthermore, the game depends on these model classes to start and create an instance of objects.

## The Card class with the attributes value and timestamp, where the value presents the integer value of the card and timestamp is how long the card is staying stationary. The getVaule behaviours of the class Card depend on the attribute value. Similarly, getTimestamp, udateTimesamp and resetTimestamp behaviours of the class Card depends on the attribute timestamp.

## The Pack class with the attribute pack, which is an array of card objects. The readPack behaviour takes in the filename as a parameter and then creates a pack from the file by reading the data and adding it to the pack attribute. Finally, getPack behaviour returns the pack attribute.

## Diagram, schematic Description automatically generatedAPI and Implementation UML Diagram

We planned to create API of some classes that will be interrupted by Threading to make it easier for future development in being thread-safe. Where, we created a Hand, Deck, PlayerDeck and Player interface. These API interfaces will be implemented to their own implementation classes. The advantage of doing this is that without bothering about the implementation part, we can achieve the security of the implementation. In java, multiple inheritances are not allowed, however, you can use interface to make use of it as you can implement more than one interface. This will also make it easier for future adaption and maintenance.

The DeckImpl class will implement Deck interface and contain a cards attribute, which is a queue of card objects. The getCard behaviour returns and removes the first element in the queue of cards that are contained within DeckImpl. The addCard method adds a specified card to the queue. Finally, isEmpty returns true if the queue is empty else false.

The HandImpl class will implement Hand interface and contain a cards attribute, which is an array of card objects. The addCard behaviour adds a top card from the deck to the last index of the array as the hand is always in sorted order and we can just check if the last element is null. The removeCard method uses the leastFrequent function to find the list of the least common cards in the hand and then picks a random card from that list to remove and uses the removingCard function to remove it. The getHand function returns the hand that is stored in the object. Similarly, sortHand sorts the array into numerical order using Comparator from java.util. Lastly, isWin behaviour checks if the hand is a winning hand by seeing if all the cards in the hand are the same. Furthermore, the private behaviour leastFrequent counts up how many of each card is in the player's hand and then if the players preferred denomination is not contained within their hand then the least frequent card is added to a list of Card type so that if there is a tie then the calling function can choose which card to remove. Finally, removingCard takes a card as a parameter and removes it from the list by creating a temporary copy removing and removing the element of that and then setting the real list equal to the temporary one.

The PlayerImpl class will implement Player interface and contain attributes name, preferredCard, hand, deck, isStop, playerDeck, nextPlayerId, winner and winningHand. The initHand behaviour hands out the cards from the pack to the player from their respective decks. Also, to keep it in line with the specification the decks are all made in a round robin handing out method, which is the same as round robin handing out the cards from the pack then putting the excess cards back into each deck. The setNextPlayerDeck takes a particular deck as a parameter and then sets that decks as the current deck for the respective player. The getPlayerDeck method returns the deck of the player. Finally, the start method:

- Runs in a while loop until a player has won.

- Checks to see if the player has won and if not, it does its next turn by removing a card and then picking up a card.

- If the player has won then it displays the hand and which player won then it sets the AtomicBoolean to true and then the while loop is broken out of.

- Writing gameplay to their respectable file and deck to the deck file.

## A picture containing text Description automatically generatedCardGame UML Class Diagram

This CardGame class will be main class, which will be runnable and control point to this game. As this will be the main class, we will implement the threading of players in this class and make this whole project thread-safe.

The CardGame class with the attribute players, which is an array of PlayerImpl objects. The start function sets up a list of CompleteableFuture objects for threading. Then, starts the thread for each player and turn the list into a new list when all the objects in the completableFutures list finishes their actions. Finally, joins the thread. The createPlayers initialises and appends the players to a list of PlayerImpl types. The dealPack method deals the cards from the pack in a round robin way of dealing cards to each of the players decks. And setPlayersInitialHand adds these cards to the players hand for their starting hands. Finally, setDiscardPlayerDeck it sets the next players deck as the discard deck for the current player as in Player 1s discard deck is player 2s and so on.