In this assignment, you are going to simulate a simple card game called Belote. We first present the rules of the game and then proceed with instructions to follow. Make sure to read **all the instructions carefully** as you proceed through the assignment.

Rules of Belote

In the Belote game, a pack of 52 cards is distributed among 4 players. Each turn, each player plays a card based on the following rules:

- The first player plays the first card in hand
- All other players play the first card of the same suit in hand
- If a player does not have a card of the required suit in hand, they play the first card in hand

At the end of the turn, the player who played the highest card *in the required suit* wins the four cards on the table. The player who won the turn starts the next turn. The game goes on until no cards remain in hand. At the end, each player sums the value of the cars they won. The player with the highest score wins the game.

Instructions

Follow the instructions below carefully and feel free to ask the TAs for help during the class or by email.

Step 1: the `Card` class

A card will be represented by three variables:

- a suit (clubs, diamonds, hearts, spades) that we will represent by an integer 0, 1, 2, 3 in order to simplify the problem
- a name (1, ..., 10, king, queen, jack) that we will represent by a string
- a value that we will represent by an integer. The value of cards 1 to 10 is simply the same number. The value of a jack is 11, a queen is 12 and a king is 13.

To simplify, you may set the name equal to the string conversion of the value (for example, the name of a jack would be the string "11").
Create a `Card` class (in a Class.java file) that contains the non-static variables described above. In addition, implement the following non-static methods in the Card class:

- a constructor `Card(int value, String name, int suit)` that takes as input the card value, name and suit.
- a `printOut()` method that takes no argument and prints out the card value, name and suit (use `System.out.println()`).

Compile the Card class and make sure that you do not get any error message. Since the class does not contain any `main` method, you may not test it. Simply compile it.

Step 2: the `Player` class

A player will be represented by four variables:

- a name that we will represent by a string
- a score that we will represent by an integer
- a counter of cards that we will represent by an integer
- an array of `Card`

Create a `Player` class (in a Player.java file) that contains the non-static variables described above. In addition, implement the following non-static methods in the Player class:

- a constructor `Player (String name)` that takes as input the player name and initializes the counter of cards to 0, the array of cards to an array of 13 cards, the player name to the method argument and the score to 0.
- an `addCard (Card)` method that takes as input a card and puts it in the array (this might be a good place to increment the counter of cards)
- a `playFirst()` method that takes no input and returns the card that the player would play when he plays first (i.e the first card in the array). This might be a good place to decrement the counter of cards. The method should return `null` if there is no card left and replace by `null` the card that is being played in the player's hand.
- a `play(int suit)` method that takes as input the suit that the first player put on the table and returns the card that the player should play (i.e the first card of the same suit, or the first card of the hand if the suit is not present in the hand). In the latter case, simply make a call to `playFirst()`. This might also be a good place to decrement the counter of cards.
- a `winTurn(Card[] cards)` method that will be called when the player wins the turn. The input to the method is the set of (four) cards on the table. The method returns nothing and simply updates the score of the player.

Important note: since you cannot remove a card from the array once it has been played, use the `null` object to replace each card once it has been played. In the `playFirst()` and `play()` methods, when you loop through the array of cards, simply skip the cards that are equal to null before testing their suit.
Compile the `Player` and check that you get no error message. The Player class does not have any `main` method so you cannot test it. Simply check that it compiles well.

Step 3: the `Belote` class

The game will be represented by two static variables:

- an array of cards
- an array of players

Create a `Belote` class (in a Belote.java file) that contains the variables described above. In addition, implement the following methods in the Belote class:

- a constructor `Belote()` that takes no argument and creates four players (Adam, Bob, Charlie and Daniel), creates 52 cards (four suits of 13 cards) and distribute the cards in any order you wish (start with a simple distribution model and add shuffling later on if time allows).
- a `main (String[] arguments)` method that creates an instance of the class Belote and simulates the game. To do so, use a while loop that exits when `playFirst()` returns `null`.

Compile the `Belote` class and check that you get no error message. Then, test your simulator by running the file `Belote`

Important tips

- Do not try to code everything at once. Once you have created the `Card` class and the `Player` class, create a basic `main` method in the `Belote` class that simply calls the Belote constructor and prints out the list of cards each player has in hands.
- Use the `System.out.println()` method often to get feedback about the state of the algorithm at every step (when you create a player, when a player plays, which card is played, etc.)
- Do not forget to use the `new` operator every time you create a new object. If you simply use the `=` operator, you will create a *reference* to the object. If the object is overwritten by `null` later on, its value will be `null` wherever you did the `=` assignment.

Shuffling the game

If time allows, add a few lines of code in the `Belote` constructor to shuffle the cards. We provide here pseudo-code for a simple shuffling algorithm:

```
for (int i=0; i < cards.length; i++) { pick a random index j between 0 and cards.length swap the cards at index i and j }
```

To generate a random number between 0 and N, use `Math.Random()` which returns a float between 0.0 and 1.0 and multiply it by N.

Good luck!