CSCI-UA 9102 DATA STRUCTURES
Assignment 2

Augustin Cosse

June 6, 2025

Given date: June 6
Due date: June 15 (Midnight Any Time Frame)
Total: 20pts

In this second assignment, you will implement a war game through singly linked lists and generics. You will also get to extend the Caesar cipher into a new code based on a random mixing of the alphabet.

Question 1 (15pts)

We want to implement a simple War Game between 2 players. The game is organized with a regular deck of 52 cards which are distributed evenly among the two players. At each turn, the two players reveal the top card of their deck. The one with the strongest card wins the point.

For this assignment you will be asked to design two classes:

- The first class `MyList.java` should encode a simple (generic) LinkedList implementing the interface `ListInterface` provided in the file `ListInterface.java`
- The second file `Deck.java` should use the `MyList` class to represent a deck of cards.

To implement those two classes you are also given the class `Card.java` which implements a simple card. Concretely the `card` class contains the following methods

- A `getRank()` method which returns the rank of a card
- A `getSuit()` method which returns the suit (Clover, Diamond, Spade, Heart) of the card
- A `whichCard()` method which outputs a single String description of the card.
- Two comparison methods `isStrongerThan()` and `isEqual()` which can be used to compare two cards with one another.

Question 1.1. `MyList` (5pts)

If you open the file `ListInterface.java` you will see that your class `MyList` must implement a series of methods given below. You should therefore start your `myList.java` file with the line `public class myList<E> implements ListInterface<E>`

```
public interface ListInterface<E>{

        public boolean isEmpty(); // returns True if the List is empty
        public int numElem(); // return the number of elements in the list
        public void addFirst(E e); // Add an element on top of the list
        public void addLast(E e); // Add an element at the end of the list
        public E removeFirst(); // remove and return first element
        public void insert(int position, E e); // insert element at given
                        //position in the list
                       // position 0 is head of the list
        public E remove(int position, E e); // remove and return element
                                // at given position

        public E checkFirst(); // return (but does not remove) first element
        public E checkLast(); // return (but does not remove) last element
        public E checkElement(int position); // return (but does not remove)
                  //element at any given position
}
```

Your class `myList.java` must be a generic class so that each element in the list should contain two components: a pointer to the next element and a reference to an object of type `E`.

Before moving to Question 1.2, you should check that your `MyList` class will work as expected by creating a simple list of cards. You should instantiate this list of cards using the proper syntax for generics (see below) and make sure you can display the suit and rank of each of the elements in the list.

```
MyList<Card> cardList;
cardList = new myList<>();
```

Question 1.2. (5pts)

In this second part, you will implement the class `Deck.java`. If you open the corresponding file, you will notice that this second class should have the following form:

- It should contain an instance of the class `myList` which will be used to maintain a representation of the deck
- Besides the constructor, which will initialize the deck to an empty set of cards, it should contain a method `Shuffle` which will randomly reorganize the cards in the deck. It should also contain a method `initFullDeck` which will initialize the deck with a full set of 52 cards (the four suits of 13 cards)
- Finally it should include a method `numCards` which will return the number of cards in the deck and a method `isEmpty` which will return true once the deck is empty and false otherwise.

For the `shuffle` function, you might want to use the functions from the `java.util.Random` class (hint: you can generate a random shuffle of the deck by scaning each element of the list and swapping it with another element located at the random position returned by the function `nextInt(myList.size())` from the `Random` class.)

Question 1.3. (5pts)

Once you are done with the `Deck` class, you are left with the implementation of the War game itself.

Implementation of the Game should be done in a separate class file with a `main` method. The method should start with the sentence "`Start game: [y]/[n]`". If a `y` is entered, then a full deck of cards should be evenly distributed between the two players and the game should start until one player (you or the computer) runs out of cards. The game should then conclude with a sentence displaying the winner and the final result. The program should finally display the sentence "`Game finished, Start new game? [y]/[n]`". A new game should be started until a '`n`' is entered on the command line.

Bonus (5pts)

Try to design a more elegant program that can play the War game and display the two cards that are revealed as well as the decks. For the card images you can use the links

- https://www.iro.umontreal.ca/~reid/ift1146/E06/tp2/jeuxDeCartes.html

or

- https://commondatastorage.googleapis.com/codeskulptor-assets/cards.jfitz.png

If you use the second link, you might want to use the `BufferedImage class` to crop the set of cards. Moreover, for any of the two links, you might also want to use `JFrame` and `JPannel`.

Question 2 (5pts)

During the lectures, we have discussed one of the earliest encryption scheme: The Caesar cipher. The Caesar cipher can be used to encrypt a message by mapping each character to another character corresponding to a constant shift of the first one. To refresh your memory about that class you can check the slides of week 4 on the course website. You can also open the file `CaesarCipher.java` which is provided on github. As an example of an ecrypted message, see the example below. Here the encryption code indicates the mapping between the original alphabet and the code. the letter `'A'` corresponds to the position 0 in the array and is therefore mapped to the letter `'D'` during the encryption. Correspondingly, `'B'` will be mapped to the letter `'E'` and so on. Once the encoder is defined, the decoder is simply given by reverting the encoder. That is to say since `'A'` is mapped to `'D'` in the encoder, `'D'` (position 4) will be mapped to `'A'` in the decoder.

```
Encryption code = DEFGHIJKLMNOPQRSTUVWXYZABC
Decryption code = XYZABCDEFGHIJKLMNOPQRSTUVW
Secret: WKH HDJOH LV LQ SODB; PHHW DW MRH'V
Message : THE EAGLE IS IN PLAY; MEET AT JOE'S
```

Modify the `CaesarCipher class` to design a class `RandomCipher` so that the encoder and decoder rely on a random permutation of the alphabet (limit yourself to the 26 uppercase letters and the symbols `#, $, &,` as well as `=`)