Encryption with a Deck of Cards

Overview:

This is a team assignment.

For this assignment, we are going to encrypt a string of text using a deck of cards. This technique is considerably harder to break than some basic encryption techniques (such as the Caeser Cipher), yet it's still relatively strightforward. We'll be assuming that we have just two suits (say, hearts and spades) from a deck of cards, plus the two jokers, just to keep things simple. Further, let's assume that the values of the 26 suit cards are 1 to 26 (Ace through King of hearts, followed by Ace through King of spades), that the "A" joker is 27, and that the "B" joker is 28. Thus, 15 represents the 2 of spades. Now that you've got the idea, note that because we are doing this in a computer, note that the actual card names are irrelevant; really, we just need to track a list of numbers from 1 through 28. Well, we do need to know which ones are considered jokers (27 and 28).

The main part of this technique is the generation of the *keystream values.* (They will be used to encrypt or decrypt our messages.) Here are the steps used in our version of this algorithm, assuming that we start with a list of the values from 1–28 as described above:

1. Find the A joker (27). Exchange it with the card beneath (after) it in the deck, to move the card down the deck by one position. (What if the joker is the last card in the deck? Imagine that the deck of cards is continuous; the card following the bottom card is the top card of the deck, and you'd just exchange them.)
2. Find the B joker (28). Move it two cards down by performing two exchanges.
3. Swap the cards above the first joker (the one closest to the top of the deck) with the cards below the second joker. This is called a triple cut.
4. Take the bottom card from the deck. Count down from the top card by a quantity of cards equal to the value of that bottom card. (If the bottom card is a joker, let its value be 27, regardless of which joker it is.) Take that group of cards and move them to the bottom of the deck. Return the bottom card to the bottom of the deck.
5. (Last step!) Look at the top card's value (which is again 1-27, as it was in the previous step). Put the card back on top of the deck. Count down the deck by that many cards, starting with the top card. Record the value of the NEXT card after that in the deck, but don't remove it from the deck. If that next card happens to be a joker, don't record anything. Leave the deck the way it is, and start again from the first step, repeating until that next card is not a joker.

The value that you recorded in the last step is one value of the keystream, and will be in the range $1 - 26$, inclusive (to match with the number of letters in the alphabet). To generate another value, we take the deck as it is after the last step and repeat the algorithm. We need to generate as many keystream values as there are letters in the message being encrypted or decrypted.

As usual, an example will really help make sense of the algorithm. Let's say that this is the original ordering of our half–deck of cards:

```
1 4 7 10 13 16 19 22 25 28 3 6 9 12 15 18 21 24 27 2 5 8 11 14 17 20 23 26
```

Step 1: Swap 27 with the value following it. So, we swap 27 and 2:

```
1 4 7 10 13 16 19 22 25 28 3 6 9 12 15 18 21 24 2 27 5 8 11 14 17 20 23 26
                                                ^^^^
```

Step 2: Move 28 two places down the list. It ends up between 6 and 9:

```
1 4 7 10 13 16 19 22 25 3 6 28 9 12 15 18 21 24 2 27 5 8 11 14 17 20 23 26
                          ^^^^^^
```

Step 3: Do the triple cut. Everything above the first joker (28 in this case) goes to the bottom of the deck, and everything below the second (27) goes to the top:

```
5 8 11 14 17 20 23 26 28 9 12 15 18 21 24 2 27 1 4 7 10 13 16 19 22 25 3 6
^^^^^^^^^^^^^^^^^^^^^                          ^^^^^^^^^^^^^^^^^^^^^^^^^^^
```

Step 4: The bottom card is 6. The first 6 cards of the deck are 5, 8, 11, 14, 17, and 20. They go just ahead of 6 at the bottom end of the deck:

```
23 26 28 9 12 15 18 21 24 2 27 1 4 7 10 13 16 19 22 25 3 5 8 11 14 17 20 6
                                                         ^^^^^^^^^^^^^^^
```

Step 5: The top card is 23. Thus, our generated keystream value is the 24th card, which is 11.

Self Test: What is the next keystream value? The answer is provided at the end of this document.

OK, so what do you do with all of those keystream values? The answer depends on whether you are encoding a message or decoding one.

To encode a message, verify that all input characters are capital letters, and that there are no spaces. If they aren't, quit and issue an error message. Convert the letters to numbers (A=1, B=2, etc.). Use the above algorithm to generate the same number of values as are in the message. For each character in the message, take its number, and add to it the number in the keystream corresponding to its position. If the answer is greater than 26, subtract 26 from it. Convert the numbers back to letters, and you're done.

Decryption is just the reverse of encryption. Start by converting the message to be decoded to numbers, again doing error checking to make sure that it contains all capital letters and no spaces. Using the same card ordering as was used to encrypt the message originally, generate enough keystream values. (Because the same starting deck of cards was used, the same keystream will be generated.) Subtract the keystream values from the message numbers. If a number comes up less than 1, add 26 to it. Finally, convert the numbers to letters and read the message.

Let's give it a try. The message to be sent is this:

```
INSANE
```

Next, convert the letters to numbers:

```
I  N  S  A  N  E
9 14 19  1 14  5
```

Rather than actually generating a sequence of 6 keystream values for this example, let's just pretend that we did:

```
1 7 14 6 13 1
```

Just add the two groups together pairwise, subtracting if over 26: (Note that this isn't quite the result that the operator `%` would give you in Java.)

```
   9 14 19  1 14  5
 + 1  7 14  6 13  1
--------------------
  10 21  7  7  1  6
```

And convert back to letters:

```
JUGGAF
```

Here's how the recipient would decrypt this message. Convert the encrypted message's letters to numbers, generate the same keystream (by starting with the same deck ordering as was used for the encryption), and subtract the keystream values from the message numbers. Again, to deal with values less than 1, just add 26 to the answer if it is negative.

```
  10 21  7  7  1  6
 - 1  7 14  6 13  1
--------------------
   9 14 19  1 14  5
```

Finally, convert the numbers to letters, and viola!

```
9 14 19  1 14  5
I  N  S  A  N  E
```

Assignment: To get this all working, we're going to break this assignment up into four different parts. None of the individual pieces are all that hard, it's just a bunch to think about at once.

Parts 1 and 2: the deck

Create a class called `Deck` that maintains the list of numbers (1–28) in a **circular linked list**. A circular linked list is one where the very last item in the list points to the first one. It turns out the deck rearranging algorithm described above is more straightforward if your deck can be thought of as one big circle of cards. The actual ordering of the numbers itself should be read from a file named `deck.txt`, which contains the numbers in some given order, on a single line, separated by spaces. Your class should contain a method called `print(n)`, which prints out `n` numbers from your deck. `n` is an integer greater than or equal to 1, though it can be larger than 28. If it is, your deck should start over from the top and keep printing. We should be able to test your code as follows:

```
Deck cards = new Deck();
cards.print(3);
cards.print(30);
```

If `deck.txt` looks like this:

```
    23 26 28 9 12 15 18 21 24 2 27 1 4 7 10 13 16 19 22 25 3 5 8 11 14 17 20 6
```

then the above code should print out

```
    23 26 28
    23 26 28 9 12 15 18 21 24 2 27 1 4 7 10 13 16 19 22 25 3 5 8 11 14 17 20 6 23 26
```

Make sure to print out your items efficiently: you shouldn't be doing something awful like "find the first item in the list, print it, then start at the beginning and find the second item, print it, then start at the beginning and find the third item, print it, etc."

You must use a circular linked list. Do not use any pre-existing linked list code, apart from what I or the textbook provide.

Parts 1 and 2 are the same exercise, though for part 1, your code only needs to work for printing out 28 or fewer cards. In other words, for part 1, the circular part of your linked list does not need to work yet if you're still working on that piece.

Part 3: Encrypting and decrypting text, using an easier algorithm

Write a separate class called `Encrypt`, with a `main` that reads a string of text from a file called `input.txt`. (It contains all capital letters, no spaces.) Encrypt the text using the above algorithm, but without yet doing any of the fancy deck cutting and rearranging. Instead, generate your keystream using a really easy technique: the first number in the deck is the first number in the keystream, the second number in the deck is the second number in the keystream, etc. You can decide for yourself how to structure this, but I recommend creating a method called `keystream` that generates an array of keystream values of a specified size. `Encrypt` can then use that array to encrypt the given message. Print the encrypted message to the screen.

Likewise, create a class called `Decrypt`, that works in exactly the same way as described above, except that it decrypts a string instead of encrypting it.

(Note added on 4/20): When doing the arithmetic on the letters, use ASCII values to keep your code brief. Don't do a massive 26-way if-statement converting A to 1, B to 2, etc.)

Part 4: Doing the deck rearranging

Create a method in your `Deck` class called `rearrange` to actually rearrange the deck according to the rules given at the beginning of this assignment. Modify your code from part 2 so that it rearranges the deck as necessary to generate the keystream values.

This is the part where your circular linked list really pays off! I want you to write your code efficiently and directly, taking advantage of the fact that this is a linked list. For example, when you move a chunk of cards from one portion of the deck to another, you should only be moving the minimum number of pointers necessary to do the job. Don't move the cards one-by-one, or something inefficient like that.

Answer to the Self Test: After Step 1:

```
23 26 28 9 12 15 18 21 24 2 1 27 4 7 10 13 16 19 22 25 3 5 8 11 14 17 20 6
```

After Step 2:

```
23 26 9 12 28 15 18 21 24 2 1 27 4 7 10 13 16 19 22 25 3 5 8 11 14 17 20 6
```

After Step 3:

```
4 7 10 13 16 19 22 25 3 5 8 11 14 17 20 6 28 15 18 21 24 2 1 27 23 26 9 12
```

After Step 4:

```
14 17 20 6 28 15 18 21 24 2 1 27 23 26 9 4 7 10 13 16 19 22 25 3 5 8 11 12
```

After Step 5:

The deck is the same as it was after step 4. The 15^{th} card, the next keystream value, is 9.