Programming Assignment 1

Simplified Solitaire Encryption

In this assignment you will implement a slightly simplified version of the Solitaire Encryption algorithm designed by Bruce Schneier and used in Neal Stephenson's novel *Cryptonomicon*. Your implementation will use CIRCULAR linked lists

Worth 60 points = 6% of your course grade

Posted Fri, Feb 3

Due Fri, Feb 17, 11:00 PM (WARNING!! NO GRACE PERIOD)

Extended deadline (with ONE time free extension pass): Mon, Feb 20, 11:00 PM (NO GRACE PERIOD)

You get ONE free extension pass for assignments during the semester, no questions asked. There will be a total of 5 assignments this semester, and you may use this one free extension pass for any of the 5 assignments.

A separate Sakai assignment will be opened for extensions AFTER the deadline for the regular submission has passed. The regular submission deadline for all assignments will be on a Friday, 11 PM, and the deadline for the corresponding extensions will be on the following Monday, 11 PM.

- You will work **individually** on this assignment. Read the DCS Academic Integrity Policy for Programming Assignments - you are responsible for this. In particular, note that **"All Violations of the Academic Integrity Policy will be reported by the instructor to the appropriate Dean".**
- **IMPORTANT - READ THE FOLLOWING CAREFULLY!!!**

  Assignments emailed to the instructor or TAs will be ignored--they will NOT be accepted for grading.
  We will only grade submissions in Sakai.

  If your program does not compile, you will not get any credit.

  Most compilation errors occur for two reasons:

  1. You are programming outside Eclipse, and you delete the "package" statement at the top of the file. If you do this, you are changing the program structure, and it will not compile when we test it.
  2. You make some last minute changes, and submit without compiling.

  To avoid these issues, (a) START EARLY, and give yourself plenty of time to work through the assignment, and (b) Submit a version well before the deadline so there is at least something in Sakai for us to grade. And you can keep submitting later versions (up to 10) - we will accept the LATEST version.

- Solitaire Encryption Algorithm (Simplified)
- Generating the keystream
- Implementation
- Running the Program
- Submission
- Grading

Solitaire Encryption/Decryption Algorithm (Simplified)

Bruce Schneier designed what he calls the "Solitaire" system that field agents could use to encrypt their messages securely. All they needed was a full deck of 52 cards plus two jokers, and a whole lot of time. This system is used in Neal Stephenson's novel *Cryptonomicon*

In this assignment, you will implement a simplified version of the Solitaire encryption algorithm. Simplified because you will use a half deck of 26 cards, plus two jokers. **You will store the deck in a CIRCULAR linked list**. What follows is a description of the encryption/decryption algorithm.

The algorithm starts with a deck in some random order. It uses this deck to generate what is called a ***keystream***, which is a sequence of numbers called ***keys***. Each key will be a number between 1 and 26. Imagine that you want to encrypt the following message to send to your friend:

```
DUDE, WHERE'S MY CAR?
```

Imagine also that you start with the following half deck of cards, with the two jokers given values of 27 (Joker "A") and 28 (Joker "B"):

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

Starting with this deck, you will get the following keystream (you will see how), one key for every alphabetic character in the message to be encrypted. Here's the message again, with everything but the letters taken out of consideration, followed by a parallel list of integers which correspond to the positions of the message letters in the alphabet, and finally, the keystream, one key per character:

```
Message:     D   U   D   E   W   H   E   R   E   S   M   Y   C   A   R

Alphabet:    4   21  4   5   23  8   5   18  5   19  13  25  3   1   18
Position

Keystream:   7   16  5   8   8   15  26  9   14  23  12  15  25  3   1
```

Encryption is then done by simply adding each key of the keystream to the corresponding alphabetic position, and if this sum is greater than 26, subtracting 26 from the sum. Here's the resulting sequence of numbers:

```
           11   11   9   13  5   23   5   1   19   16   25   14   2   4   19
```

The numbers are converted back to letters, to get the following encrypted message.

```
Encrypted:   KKIMEWEASPYNBDS
```

Decryption follows a similar process.
When the decrypter gets the coded message, she generates the keystream in exactly the same way, using the same original deck as the encryption. Then, the keystream is **subtracted** from the alphabetic position values of the letters in the coded message. If a code value is equal or smaller than the corresponding decryption key, 26 is first added to it and then the key is subtracted:

```
Code:       11   11   9   13   5   23   5   1   19   16   25   14   2   4   19

Keystream:   7   16   5    8   8   15  26   9   14   23   12   15   25  3    1
            ---------------------------------------------------------------------
Message:     4   21   4    5   23   8   5  18    5   19   13   25   3   1   18
```

```
D   U   D   E   W   H   E   R   E   S   M   Y   C   A   R
```

Generating the keystream

Here is the algorithm to generate each key of the keystream, starting with the initial deck.

Get Key

- Execute the following four steps:
    - **Step 1 (Joker A):** Find Joker "A" (27) and move it ONE card down by swapping it with the card below (after) it. This results in the following, after swapping 27 with 7 in the starting deck:

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

      If the joker happens to be the last card in the deck, then loop around and swap it with the first. For example:

      ```
      5 ... 27
      ```

      Here 5 is the first card and 27 is the last card. Swapping them will give:

      ```
      27 ... 5
      ```

    - **Step 2 (Joker B):** Find Joker "B" (28) and move it TWO cards down by swapping it with the cards below (after) it. This results in the following, after moving 28 two cards down in the deck that resulted after step 1:

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

      If the joker happens to be the last (or second to last) card in the deck, then loop around and swap it with the card(s) in the front. For example:

      ```
      5 6 ... 10 28
      ```

      Here 28 is the last card. Moving it one card down gives:

      ```
      28 6 ... 10 5
      ```

      and moving it one more card down gives:

      ```
      6 28 ... 10 5
      ```

    - **Step 3 (Triple Cut):** Swap all the cards before the first (closest to the top/front) joker with the cards after the second joker:

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

      If there no cards before the first joker, then the second joker will become the last card in the modified deck. Similarly, if there are no cards after the second joker, then the first joker will become the first card in the modified deck.

    - **Step 4 (Count Cut):** Look at the value of the last card in the deck. Count down that many cards from the first card, and move those cards to just *before* the last card:

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

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

      If the last card happens to be Joker B (28), use 27 (instead of 28) as its value for this step.

- After these four steps are done, look at the value of the first card. If it is 28, then treat the value as 27. Count down by that many cards from the first. Look at the value of the **next** card. If it happens NOT to be 27 or 28, this is the key. Otherwise, repeat the whole process (Joker A through Count Cut) with the latest (current) deck (NOT the initial deck).

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

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

  In this example, the first card value is 28, so treat it as 27. The 27th card in the deck is 16, and the next card is 7, so 7 is the key.

Once a key is found, the algorithm is repeated to find the subsequent keys, starting every time with the current deck (NOT the initial deck).

Implementation

Download the `solitaire_project.zip` file attached in the Sakai assignment. DO NOT unzip it. Instead, follow the instructions on the Eclipse page under the section "Importing a Zipped Project into Eclipse" to get the entire project into your Eclipse workspace.

You will see a project called `Solitaire` with the classes `CardNode` (which implements a node of the linked list deck), `Solitaire` (which implements the encryption and decryption algorithms), and `Messenger` (the application driver), all in the package `solitaire`.

You will also see a sample input file, `deck.txt`, DIRECTLY UNDER THE PROJECT FOLDER (NOT under solitaire or src). This is where other input files must go when you test your program.

You need to fill in the implementation of the `Solitaire` class where indicated in the `Solitaire.java` source file. This includes the following:

Method	Points
`jokerA`	8
`jokerB`	8
`tripleCut`	16
`countCut`	12

getKey	5
encrypt	6
decrypt	5

The `printList` method has been implemented for your convenience - you may use it to print the deck for verification/debugging. (But it is NOT used by us when testing your program - see the Grading section at the end for details.)

Do NOT change `CardNode.java` in any way.

While working on **Solitaire.java**:

- You may NOT add any `import` statements to the file.

  Note: Sometimes Eclipse will automatically add import statements to the file, if you are using an unknown class other than the ones you are given. It is your responsibility to delete such automatically added import statements. At the time of grading, we will delete all such additional import statements we find, before compiling your code. If your code does not compile, we will not be able to test it, and you will get a zero.

- You MUST work with CIRCULAR LINKED LISTS only. You may NOT transfer the contents of the linked lists to arrays or other data structures for processing.
- You may NOT add any new classes (you will only be submitting `Solitaire.java`).
- You may NOT add any fields to the `Solitaire` class.
- You may NOT modify the headers of any of the given methods.
- You may NOT delete any methods.
- You MAY add helper methods if needed, as long as you make them `private`.

Notes on character manipulation

The `java.lang.Character` class has several useful methods for manipulating characters. Here are a few you may find useful.

- `Character.toUpperCase(ch)`
- `Character.toLowerCase(ch)`
- `Character.isLetter(ch)`

You can switch between character and integer values using casts. Here's an example that starts with the character 'D', gets its position in the alphabet (4), adds 1 to it, and gets the next character ('E'):

```
char ch = 'D';
System.out.println(ch);  // D
int c = ch-'A'+1
System.out.println(c);   // 4
c++;
ch = (char)(c-1+'A');
System.out.println(ch);  // E
```

Make sure you read the comments above each method in the code for additional details on what the method does. This information, in addition to the specification in this document, is essential to correctly implement the method.

Running the Program

The class `Messenger` has a `main` method, so it can be run as a Java application.

Here's a sample run to encrypt a message, with the deck supplied in the file `deck.txt` (which came with the project):

```
Enter deck file name => deck.txt
Encrypt or decrypt? (e/d), press return to quit => e
Enter message => DUDE, WHERE'S MY CAR?
Encrypted message: ODALGGPCLAVJNAP
```

And here's a sample to decrypt a message that was encrypted using `deck.txt`:

```
Enter deck file name => deck.txt
Encrypt or decrypt? (e/d), press return to quit => d
Enter message => ODALGGPCLAVJNAP
Decrypted message: DUDEWHERESMYCAR
```

You should look at the code in `Messenger` and understand that it reads the deck from whatever input file you specify, and sets up the Solitaire object's linked list for this deck, by calling the `makeDeck` method. It then calls the `encrypt` or `decrypt` methods.

Make sure you test your code comprehensively with other test cases of your own. Pay particular attention to the extreme or special cases in each of the steps of the algorithm. Remember, when you create other test input files, you should put them alongside `deck.txt`, DIRECTLY UNDER THE PROJECT FOLDER.

You may assume that all input decks will be legitimate: a deck will consist of (only) values from 1 through 28 in some order. So you don't need to do any check on input format.

You may also assume that all the letters in the string to be encrypted or decrypted will be uppercase, so you don't need to do any checking. But the string to be encrypted may have spaces, punctuation characters, etc. which should be ignored while encrypting. So the decrypted string will only have uppercase letters, even if the input string that was encrypted had characters other than letters.

Since there will no be no errors in input, you will not need to throw any exceptions.

Submission

Submit your **Solitaire.java** source file (NOT Solitaire.class) in Sakai.

You will ONLY submit this file, which means you should not make any changes to `CardNode.java` because we will be using the original `CardNode.java` to test your implementation.

Grading - IMPORTANT!!!

Your submission will be auto-graded by a grading script that will run several test cases on each graded method. For each test case, the result computed by your code will be compared with that computed by our correct code.

Result computed by your code means this:

- The actual configuration of the circular linked list in your Solitaire object (pointed to by `deckRear`) computed by the methods `jokerA`, `jokerB`, `tripleCut`, and `countCut`
- The integer returned by `getKey`
- The string returned by each of `encrypt` and `decrypt`.

Result does NOT refer to anything your program might print as "output". All printed output will be ignored, including any debugging statements you might have left lying around in your code - they will have no bearing on the tests we run.

When grading is done, your test report will be emailed, detailing the score on each test case. Test cases will be posted so you can run your program against them to verify the test report.

Your methods will be tested *independently*. This means the grade for a method is independent of the grade for any other method. So, for instance, when we grade `countCut`, we will *not* use your `jokerA`, `jokerB`, and `tripleCut` implementations for the calls that precede the call to `countCut`. Instead, we will use our correct implementations of these methods. This way, even if any of these other methods does not work correctly, you will still get credit for `countCut` if it works correctly by itself.