Solitaire

- things to address – the main areas where there is room for improvement
  - comments
    - for classes and methods
    - Javadoc style
  - correctness/robustness
    - identifying and checking preconditions
    - checking class invariants for `SolitaireDeck` – `checkStructure()` and `checkContents()`
    - checking that assertions are on when `SolitaireTester` is run
    - error-checking user input (including menu choices)

Solitaire

- things to address – the main areas where there is room for improvement
  - tester
    - `SolitaireTester` class
    - coverage of all public methods in `SolitaireDeck`, `KeystreamGenerator`, `SolitaireEncoder` – including `getDeckSize()`, `getTopCard()`, `getBottomCard()`, `getNthCard(n)`
    - coverage of test cases
      - if the specifications for a method draw attention to particular situations – such as the joker being first, last, or next-to-last in the joker swaps – definitely make sure you have test cases for those!
      - typical cases with linked lists – head, tail, empty list, list with one thing
        - the deck will always have at least two nodes (the jokers), but also consider empty or short lists when you are manipulating sublists (such as in `tripleCut`)
    - tester output – it should be easy to tell what worked and what didn't
      - at a minimum, print name/description of test and passed/failed
      - useful to also print expected result and actual result
      - you should not have to work through what the correct answer should be when looking at tester output

Solitaire

- instance variables should be `private`
- linked lists and `DoubleListNodes` are internal to `SolitaireDeck`
  - no nodes as parameters or return values from methods
  - `KeystreamGenerator` should only call `SolitaireDeck` methods to manipulate the deck
- `checkContents()` should check for duplicate cards, missing cards, and extra cards
  - make sure that there are decksize+2 nodes in the list – only counting through decksize+2 nodes isn't enough to know there aren't any more

Solitaire

Keystream Generation

The central part of the Solitaire algorithm is the generation of a *keystream* - a sequence of numeric values which will then be used to encrypt or decrypt a message. To generate the next value in the keystream:

- Do a joker swap with joker A.
- Do a joker swap with joker B.
- Do a triple cut.
- Do a count cut, using the value of the card at the bottom of the deck as the number of cards to count.
- Look at the top card's value and count down that many cards from the top of the deck, counting the top card as 0. If the resulting card isn't a joker, return that card's value as the next value in the keystream. If that card is a joker, go back to step 1 and repeat the whole process until you get to a card that isn't a joker.

- use the top card's value to count down *n* cards and then check if *that* card is a joker
  - don't check the top card – use its value as is

Solitaire

- tester implementation
  - using `assert` to check if a test case worked has the drawback that the first test that fails will end the tester
    - assertions are meant as a backup layer for catching problems during normal running of the program
    - prefer writing `if` statements to check test cases so that one bad test case doesn't halt the entire tester
  - prefer testing specific cases whenever possible
    - checking only that a keystream value is within the right range is better than nothing – and will catch some problems – but when you can check for the actual correct value, do it!
    - handout provides examples that give you specific test cases for keying the deck, keystream values, and encryption and decryption
  - hardcode the expected result
    - generating the expected result using the same procedure as the operation you are testing doesn't tell you whether either chunk of code is correct