# COMP 2140 Assignment 3: Stacks and Queues

Robert Guderian and Helen Cameron

Due: Friday, November 9, 2018 at 4:30 p.m.

## Hand-in Instructions

Go to COMP2140 in UM Learn; then, under "Assessments" in the navigation bar at the top, click on "Assignments". You will find an assignment folder called "Assignment 3". Click the link and follow the instructions. Please note the following:

- Submit ONE .java file file only. The .java file must contain all the source code that you wrote. The .java file must be named `A3<your last name><your first name>.java` (e.g., `A3CameronHelen.java`).
- Please do not submit anything else.
- We only accept homework submissions via UM Learn. Please DO NOT try to email your homework to the instructor or TA or markers — it will not be accepted.
- We reserve the right to refuse to grade the homework or to deduct marks if you do not follow instructions.
- **Assignments become late immediately after the posted due date and time.** Late assignments will be accepted up to 49 hours after that time, at a penalty of 2% per hour or portion thereof. After 49 hours, an assignment is worth 0 marks and will no longer be accepted.

## How to Get Help

**Your course instructor is helpful:** For our office hours and email addresses, see the course website on UM Learn (on the right side of the front page).

For email, please remember to put "[COMP2140]" in the subject and use a meaningful subject, and to send from your UofM email account.

**Course discussion groups on UM Learn:** A discussion group for this assignment is available in the COMP 2140 course site on UM Learn (click on "Discussions" under "Communications"). Post questions and comments related to Assignment 3 there, and we will respond. Please do not post solutions, not even snippets of solutions there, or anywhere else. We strongly suggest that you read the assignment discussion group for helpful information.

**Computer science help centre:** The staff in the help centre can help you (but not give you assignment solutions!). See their website at http://www.cs.umanitoba.ca/undergraduate/computer-science-help-centre.php for location and hours. You can also email them at helpctr@cs.umanitoba.ca.

## Programming Standards

When writing code for this course, follow the programming standards, available on this course's website on UM Learn. Failure to do so will result in the loss of marks.

## Question

**Remember:** You will need to read this assignment many times to understand all the details of the program you need to write.

**Goal:** The purpose of this assignment is to write a program that plays a simple card game called War, using stacks and queues.

**Code you can use:** You are permitted to use and modify the code your instructor gave you in class for linked lists, stacks and queues. You are NOT permitted to use code from any other source, nor should you show or give your code to any other student. Any other code you need for this assignment, you must write for yourself. We encourage you to write ALL the code for this assignment yourself, based on your understanding of stacks and queues — you will learn the material much better if you write the code yourself.

**The card game War:** This card game requires no skill from the two players — it is the kind of card game that a patient adult plays with a small child.

**Card deck:** The game is played with one or more ordinary decks of 52 cards. (You can assume there's only one deck.) Each card has one of the four suits (spades ♠, diamonds ♦, hearts ♥, or clubs ♣). Each suit contains 13 cards with the ranks Ace (A), 2, 3, 4, ..., 10, Jack (J), Queen (Q), King (K). The ranks of the cards are important in this game and the suits are ignored.

**Game set up:** The deck (or decks) is thoroughly shuffled (randomized). Then each player is dealt half the cards. The cards are left face down — the players are not allowed to look at their hands.

**Game play overview:** The game consists of a sequence of rounds. In a round, the players reveal some of their cards. One of the players wins the round (based on comparing the ranks of some of the revealed cards). The round winner gets all of the revealed cards, which are added (face down) to end of the round winner's hand.

**Game end:** The game ends when one player has all of the cards — that player is the winner of the game.

**One round:** The players flip over (reveal) the front card of their hands and compare the ranks of the revealed cards to see who wins the round:

- If the two revealed cards are not equal rank, then the player with the highest ranked card wins the round. (Note: The ranks, from lowest to highest rank, are Ace (A), 2, 3, 4, ..., 10, Jack (J), Queen (Q), King (K).)
- If the revealed cards are equal rank, then it is WAR! (See below to see how a war is handled.) The winner of the war is the winner of the round.

The winner of the round gets all of the revealed cards (including all cards revealed in a war), which are added (face down) to end of the round winner's hand.

**War:** The players repeat the following steps until one player wins the war or one player runs out of cards (the other player — the one with cards — wins the war):

Each player flips over (reveals) two more cards in a stack on top of the equal-ranked already revealed card. These two cards are called the "ante" (pronounced like the word "auntie") — they are just there to make the win bigger. Then each player flips over (reveals) one more card, which goes on the top of the stack. The ranks of the last revealed card from each player are compared to determine who wins the war:

- If the top two revealed cards are not equal rank, then the player with the highest ranked card wins the war. (Note: The ranks, from lowest to highest rank, are Ace (A), 2, 3, 4, ..., 10, Jack (J), Queen (Q), King (K).)

- If the top two revealed cards are equal rank, then it is still WAR! (The players repeat the war steps.)

**Implementation details:**

**A player's hand:** A player's hand should be implemented as an ordinary Queue. The queue should be implemented as an array with front and end indexes, where end indicates the next available position, and one position in the array is empty.

**A player's stack of revealed cards in a war:** A player's stack of revealed cards in a war should be implemented as an ordinary Stack. The stack should be implemented as an array with a top index.

**A deck of cards:** There are a number of ways of implementing a deck of cards. One way is to represent he 52 cards in a deck as the integers 0, 1, . . . , 51:

- The first suit is the first 13 integers 0, 1, . . . , 12, representing the ranks Ace (A), 2, 3, 4, . . . , 10, Jack (J), Queen (Q), King (K) in the first suit.
- The second suit is the second group of 13 integers 13, 14, . . . , 25, representing the ranks Ace (A), 2, 3, 4, . . . , 10, Jack (J), Queen (Q), King (K) in the second suit.
- The third suit is the third group of 13 integers 26, 27, . . . , 38, representing the ranks Ace (A), 2, 3, 4, . . . , 10, Jack (J), Queen (Q), King (K) in the third suit.
- The fourth suit is the fourth group of 13 integers 39, 40, . . . , 51, representing the ranks Ace (A), 2, 3, 4, . . . , 10, Jack (J), Queen (Q), King (K) in the fourth suit.

(The order of the suits doesn't matter, you can order them any way you like.)

Then the formula 1 + (`c` mod 13) computes the rank of the card represented by the integer `c`.

The formula `c / 13` (use integer arithmetic — throw away fractional parts) computes the suit of the card represented by the integer `c` (the formula gives either 0, 1, 2, or 3). If you use more than one deck of cards, then this formula becomes (`c / 13`) mod 4.

Examples: Suppose we decide that the order of the suits will be 0 = spades ♠, 1 = diamonds ♦, 2 = hearts ♥, 3 = clubs ♣.

- The card represented by 2 has rank 1 + (2 mod 13) = 3 and suit 2/13 = 0, so the card is 3 ♠.
- The card represented by 49 has rank 1 + (49 mod 13) = 1 + 10 = 11 and suit 47/13 = 3 + 8/13 (third suit), so the card is the jack♥.

**Printing suits:** It is fine to simply print S for spades, D for diamond, H for hearts, and C for clubs.

If you want to get fancier, the char constants 9824, 9829, 9830, and 9827 give you the filled symbols for spades, hearts, clubs, and diamonds, respectively. Be warned: these are Unicode characters — your computer may not handle them correctly.

**Sample output:** You need to print out the game play. Files `A3sampleOutput1.txt` and `A3sampleOutput2.txt` are example outputs showing two different game plays. (You'll get a different game every time because the deck is thoroughly shuffled before play starts.)