We need more programming. We want more programming. We love programming.

If you’ve been trying to do your programs all at the last minute stop being so stupid and spread out the work. Sheesh. Do as much as you can. You should probably put in at least 15 hours of work on this, unless you finish more quickly than that of course.

If you get stuck on something feel free to get some help or email me.

For this set of problems you can do them in any order you like. Read through the entire list and choose the ones that you think would benefit you the most, either because they are something you feel you can handle and therefore will give you good programming practice or because you think they look challenging and you are ready for a challenge. Each requires a separate report. Remember that a report includes your code, a driver if appropriate, sample output (and input too if appropriate), and perhaps a few comments about the problem and its solution. If the problem requires you to change code provided elsewhere then you do not need to include all of the code provided; just clearly indicate the code that you added/changed/removed in order to implement the change.

Print this sheet out and hand it in with your reports. For each problem that you attempt, write the number of minutes you worked on the problem on this sheet, just beside the problem number.

1. Write a program that repeatedly prompts the user to enter strings, using the string "x done" to indicate when finished. The user is assumed to only enter strings of the form "f name" or "m name". Output the names that had 'm' indicated in the same order they are entered and then do the same for the names that had 'f' indicated. Use two of our queues.

   Sample Run
   Input a gender and name (x done to quit) > m Fred
   Input a gender and name (x done to quit) > f Wilma
   Input a gender and name (x done to quit) > m Barney
   Input a gender and name (x done to quit) > m BamBam
   Input a gender and name (x done to quit) > f Betty
   Input a gender and name (x done to quit) > x done

   males: Fred Barney BamBam
   females: Wilma Betty

2. Create an interactive test driver for the ArrayBndQueue class. See pages 92 to 99 for more information about interactive test drivers.

3. Take a blank piece of paper. Put your name at the top. Divide the paper vertically into five sections by drawing horizontal lines. On the left side of section 1 write 'ArrayBndQueue q = new ArrayBndQueue(5);', of section 2 write 'q.enqueue("X")', of section 3 write 'q.enqueue("M")', of section 3 write 'q.dequeue( )', and of section 5 write 'q.enqueue("T")'. On the right side of each section draw the internal view of q, showing what the data "looks" like after each statement is executed. Do not just draw what you think it will look like but actually read and follow the code as you are creating the drawings.

4. Take a blank piece of paper. Put your name at the top. Divide the paper vertically into five sections by drawing horizontal lines. On the left side of section 1 write 'LinkedUnbndQueue q = new LinkedUnbndQueue( );', of section 2 write 'q.enqueue("X")', of section 3 write 'q.enqueue("M")', of section
3 write 'q.dequeue()', and of section 5 write 'q.enqueue("T")'. On the right side of each section draw the internal view of q, showing what the data "looks" like after each statement is executed. Do not just draw what you think it will look like but actually read and follow the code as you are creating the drawings.

5. A peek method returns the first element in a queue without removing it. It throws a QueueUnderflowException if it is called on an empty queue. Implement peek for the ArrayBndQueue class.

6. A peek method returns the first element in a queue without removing it. It throws a QueueUnderflowException if it is called on an empty queue. Implement peek for the LinkedUnbndQueue class.

7. Solitaire: For this console-based application you should use one of the queue implementations that includes the peek method. Use the RankCardDeck class from Section 5.5 to create a deck of cards. Also create three queues of cards (essentially integers); let’s call them queues A, B, and C. Shuffle the deck. Deal a card to queue A, then B, then C until 30 cards have been dealt. The user starts with $30. The card values at the front of queues A and C are displayed to the user, with a "?" in between, for example: 5 ? 0. The ? is supposed to represent the card value at the front of queue B. The user must bet anything between $10 and their total money that the ? will be "in-between" the other two cards (ties do not count so for example 3 is not in-between 3 and 5; if the two card displayed happen to have the same rank value then the loser is out of luck). If it is in-between they win (e.g. if they bet $20 they get the $20 back plus an additional $20). Play continues for 10 rounds or until the player runs out of money. During the game provide appropriate feedback to the user.

8. Do the previous problem as a GUI.

9. Chapter 5, number 16: create a toString method for the ArrayBndQueue class.

10. Chapter 5, number 26: create a toString method for the LinkedUnbndQueue class.

11. Create an ArrayListUnbndQueue class, using an ArrayList instead of an array as the underlying storage. See pages 194 – 195 for a similar class.

12. Chapter 5, number 23c: make a change to the War Game program (report the average number of wars).

13. Chapter 5, number 23d: make a change to the War Game program (report the number of wins for each player).

14. A “dequeue” is like a queue but it also allows you to insert into the front of the queue and to remove from the rear of the queue. Create an array based Dequeue class. Feel free to reuse code from the queue classes already provided in the book, if you can.

15. Create a linked based Dequeue class. Feel free to reuse code from the queue classes already provided in the book, if you can.

16. Study the Case Study in Section 5.7 and do number 33a (output largest number of customers).

17. Study the Case Study in Section 5.7 and do number 33b (use shortest finish time instead of shortest queue).

18. Chapter 5, number 23d: create a linked queue that uses only one reference to the queue.