HOMEWORK PROBLEMS 9

1) Give a statement that creates an `int` array with 200 slots.

2) Give a `for` loop that initializes each slot of an `int` array `ia` to one more than its index.

3) Give the statement that creates a 5 by 10 `double` array.

4) Suppose an array contains the integers 1 to 10 in ascending order. How many probes will a binary search perform to find 3 in the array?

5) What does the program in Fig. 9.15 display?

6) Incorporate the code below in a program. Compile and run. What happens? Why?

```
int[] a;
a = new int[5];
a[5] = 10;
```

7) What is unusual about the following `for` loop (it is legal):

```
for (int i = 1, sum = 0; i < 10; i++)
   sum = sum + i;
```

You can have more than one statement in the startup or update actions in a `for` loop as long as each statement is separated from the next with a comma (the comma replaces the semicolon that normally terminates a statement).

8) Incorporate the following code in a program. Compile and run. What do the two `for` loops do?

```
int ia[] = {1, 2, 3};
for (int z, i = 0; i < ia.length; i++)
{
   z = ia[i];
   System.out.println(z);
}
for (int z: ia)              // enhanced for loop
   System.out.println(z);
```

The startup action in the first `for` loop declares `z` and `i`, and initializes `i` to 0. It then displays all the elements in the `ia` array. The second `for` loop is an example of an **enhanced for loop**. `z` in this loop is not the index used to access the array. Instead, `z` receives the successive values in the array. This loop is equivalent to the first loop.

9) Do enhanced `for` loops (see homework problem 8) work for an `ArrayList`? Incorporate the following code in a program and run it to check your answer:

```
ArrayList<Integer> ial = new ArrayList<Integer>();
ial.add(100);
ial.add(200);
ial.add(300);
ial.add(400);

for (int z: ial)
   System.out.println(z);
```

10) When you create an array with the `new` operator, can you use an expression for the size? For example, is the following code legal:

```
int x = 3, y = 4;
int[] a = new int[x + y];
```

Run a test program to check your answer.

11) Try to assign a new value to the `length` field for an array. What happens?

12) Write a program that includes the following loop that displays the contents of the `z` array:

```
i = 0;
while (i < z.length)
{
   System.out.println(z[i]);
   i++;
}
```

Then replace the entire body of the loop (including the braces) with the following single statement:

```
System.out.println(z[i++]);
```

Does the loop work the same way as the original version? What would happen if the `++` operator preceded `i`?

13) Write a program that reads in 20 strings from the keyboard and then displays them in reverse order. Read the strings into an array. Then write a second program like the first, but use an `ArrayList` instead of an array.

14) In the program below, we have an instance `int` array `nsia` and a static `int` array `sia`. The natural place to initialize the instance array is in the constructor. However, although it is legal to initialize the static array in the constructor, you should not do so for two reasons:

    1) We can use a static field before creating any objects—that is, before any constructor is executed. Thus, if a constructor performs the initialization, it would be possible to use the static field before it has been initialized.

    2) Each time we create an object, the static array would be re-initialized. We want the static array initialized only once when the class containing it is first loaded into memory.

    To initialize the static array, use a **static initializer** (see lines 6 to 11 below). It is executed only once when the class is first loaded into memory. Use it to initialize static fields when the initialization is too involved to be handled in the declaration of the static field. For example, if we have a static field `x` whose initial value is supposed to be 1, we can simply declare it with

```
private static int x = 1;
```

    We would not need a static initializer. We could similarly declare and simultaneously initialize an array:

```
private static int[] a = {1, 2, 3, 4};
```

    However, this approach is not practical if the array is large.

    A static initializer has no name. It consists of the reserved world `static` followed by a block of initialization code. Run the program below. From the output generated, determine the order in which `main`, the static initializer, and the constructor are executed.

```
class C9h14
{
   public static void main(String[] args)
   {
      System.out.println("Start of main");
      Initializer r = new Initializer();
      System.out.println("End of main");
   }
}
//==================================================
class Initializer
{
   private int[] nsia = new int[100];
   private static int[] sia = new int[100];
   //----------------------------------
   static               // static initializer
   {
      System.out.println("In static initializer");
      for (int i = 0; i < sia.length; i++)
         sia[i] = i;
   }
   //----------------------------------
   public Initializer() // constructor
   {
      System.out.println("In constructor");
      for (int i = 0; i < nsia.length; i++)
         nsia[i] = i;
   }
}
```

15) Write a program that reads in 15 integers into a 3-by-5 array. Display the last row. Display the last column. Display the entire array with each row on a separate line.

16) Is the following code legal:

```
int[] a = new int[3];
a[0] = a[1] = a[2] = 5;
System.out.println(a.toString());
System.out.println(a);
```

Does an array have a `toString` method? Run a test program to check your answer. Try to decipher the output displayed.

17) Use the selection sort to sort `n` numbers. Determine the value of `n` for which the elapsed time or your sort is about 10 seconds. Double `n` (i.e., double the number of numbers to sort) and sort. What is the sorting time for the new value of `n`? Double `n` again and determine the sorting time. What happens to sorting time each time you double `n`? Estimate how long sorting 1 billion numbers would take.

18) Use the selection sort to sort the numbers in the `int` array `ia`. Use calls to `System.currentTimeMillis` before and after the sort to determine the elapsed time:

```
long start = System.currentTimeMillis();
// do sort
long stop = System.currentTimeMillis();
```

`System.currentTimeMillis` returns the current time in milliseconds. Thus, to get elapsed time in seconds, divide the difference in the start and stop times by 1000.0. Then round the result to the nearest long value by calling `Math.round`:

```
long elapsed = Math.round((stop – start)/1000.0);
```

Determine the size of `ia` for which the elapsed time of the sort is about 100 seconds. Now perform the same sort using the static sort method in the `Arrays` class in the `java.util` package. Call this method with

```
      Arrays.sort(ia);
```

What is the elapsed time for this sort? Is this sort significantly faster that the selection sort?

19) Each row in a two-dimensional array does not have to have the same length. Arrays in which rows have differing lengths are called **ragged arrays**. The following code creates a ragged array `m` with three rows:

```
int m[][] = new int[3][];
m[0] = new int[3];
m[1] = new int[5];
m[2] = new int[2];
```

Write a test program in which you create a ragged array using the code above. Assign each slot the sum of its row and column indices. Use nested `for` loops. Draw a picture of the jagged array.

20) Does the `ArrayList` class have an `equals` method? For example, does the following code work as you would expect:

```
String s1 = new String("hello");
String s2 = new String("hello");
ArrayList<String> q1 = new ArrayList<String>();
ArrayList<String> q2 = new ArrayList<String>();
q1.add(s1); q1.add(s1);
q2.add(s2); q2.add(s2);
// compare two ArrayLists with equals
System.out.println(q1.equals(q2));
```

21) Try out the code below. When `remove` is passed `i2` does it remove the `i1` object (see the comments in the code)?

```
ArrayList<Integer> ial = new ArrayList<Integer>();
Integer i1 = new Integer(1);
Integer i2 = new Integer(1);     // construct identical object
if (i1 == i2)
   System.out.println("identical references, identical objects");
else
   System.out.println("different references, identical objects");
ial.add(i1);
ial.remove(i1);                  // remove what was just added
System.out.println(ial.size());  // should be zero
ial.add(i1);
ial.remove(i2);                  // will passing i2 remove i1?
System.out.println(ial.size());  // is it zero?
```

22) What does `remove` return if the item to be removed is not in the `ArrayList`? Try this code at the end of the `main` method in `C9h22.java` (a copy of Fig. 9.15):

```
String t = sal.remove("not in sal");
System.out.println(t);
```

23) The complexity of the binary search algorithm is O($\log_2 n$). What is the complexity of the sequential search algorithm?

24) Write a program that displays the sum of the command line arguments. For example, if you enter

```
java C9h24 1 3.5 7
```

your `C9h24.java` program will display

```
Sum = 11.5
```

Your program should work with any number of command line arguments. *Hint*: Use `Double.parseDouble` (which converts a string to type `double`).

25) Write a program that prompts for and reads in 10 `int` numbers. Your program should then display every number read in that is bigger than the average of all the numbers read in. Test your program by entering the numbers 1, 2, 3, 4, 5, 6, 7, 8, 9, and 30.

26) Write a program in which you create a 5x10 `int` array. Using nested `for` loops, assign row 0 all zeros, row 1 all 1's, and so on. Display the final contents of the array in rectangular format using a second set of nested loops.

27) Write a program that prompts for and reads in three test grades for each of 5 students. Your program should then display the average for each student and the average for each test. Be sure to label your output appropriately. Use a two-dimensional array. Use the following test data:

```
Student 1: 100, 100, 100
Student 2:  90,  90,  90
Student 3:  80,  80,  80
Student 4:  30,  40,  50
Student 5:   0,   1,   2
```

28) Write a program in which `main` creates two `int` arrays each with 10 slots and then initializes the first array with random integers. `main` then calls `copyArray`, passing it the two arrays. `copyArray` should copy the values from the first array to the second and then return to `main`. `main` should then display the contents of both arrays side by side.

29) Create an `ExpandableArray` class. It should have methods `add`, `get`, `set`, and `size`. These methods should work like the identically named methods in an `ArrayList`. Internally, `ExpandableArray` should have an `int` array with 2 slots initially. The `size` method should return the net number of integers added—not the number of slots in the internal array. Thus, initially size should be zero. If on a call of add the internal array is full, the `add` method should

    1) Create a new array with twice the number of slots as the current array.
    2) Copy the contents of the original array to the new array, and thereafter use the new array (until it too is not big enough).

    Write a program that uses your `ExpandableArray` class. It should

    1) create an `ExpandableArray` object
    2) add 1, 2, 99, 4, 6, 7, 8, 9, and 10, in that order

3) set 99 to 3
4) insert 5 after the number 4
5) display the current size
6) display the integers stored by calling `get` within a `for` loop.

30) Implement and test a **bubble sort**. To sort *n* numbers, make *n* - 1 passes. On each pass, compare every pair of adjacent numbers. If any pair is not in ascending order and not equal, switch the pair. On the first pass, start from the first element (compare the first with the second, the second with the third, and so on). On the second pass, again start from the first element but stop your compares one slot from the end of the array. In each successive pass, stop one slot before the stopping point of the previous pass. Use your bubble sort to sort 100, -3, 15, -3, 7, -3, 2, 311, 5, 8, and 3. Display the sorted numbers.

31) Modify the program in Fig. 9.11 so that is uses an `ArrayList` in place of an array. *Hint*: Replace line 26 with

```
if (a.get(j).compareTo(min) < 0)
```

32) Create a `Table` class according to the following specifications:

Data fields:	`ArrayList<Integer> id, ArrayList<String> name`
Constructor:	`public Table()` creates the id and name ArrayLists.

Methods:	`public void add(int i, String n)`
	adds `i` to `id`, adds n to `name`
	`public String getName(int searchID)`
	returns the string in `name` whose id is `searchID.`
	If `searchId` is not in `id`, `getName` should return `null`.

Write a program that uses your `Table` class. Your program should execute the following loop:

1) Prompt for and read in an ID number (a non-negative integer).
2) If the number is negative, exit the loop.
3) Prompt for and read in a name. Call the `add` method in a `Table` object, passing the ID number and name read in.
4) Go to step 1.

When the loop exits, execute a second loop:

1) Prompt for and read in an ID number.
2) If the number is negative, exit the loop.
3) Display the name that corresponds to the number entered. If no name corresponds to the number entered, display `INVALID NUMBER`.
4) Go to step 1.

Test your program by entering the following data: For the first loop, enter

```
151 Tom 22 Dick 16 Harriet -1
```

For the second loop, enter

```
16 151 22 55 -1
```

33) Same as homework problem 30 in Chapter 8 except replace the *entire* `switch` statement in `countIt` with a *single* simple statement (not an `if` statement) that increments one of the slots of a counter array `c`. The `c` array should have five slots, each slot serving as a one counter. For example, `c[0]` should count the number of 0's, `c[1]` should count the number of 1's, and so on. Design your program so that it can be modified to handle a

different number of counts with almost no changes. For example, changing your program to count the occurrences of the numbers from 0 to 100 instead of from 0 to 4 should require a change of only one line of your program.

34) Create a `BinarySearch` class. It should have an `add` method which adds the integer it is passed to an internal array, and a `search` method that determines using a binary search the index of the number it is passed. Create a `BinarySearch` object. Add the following integers: 2, 44, 98, 2020, 4412, 12345, 21345, 34343, 73737, and 99999, in that order. Then call `search` for each of these arguments: 2, 4412, 99999, and 5000. For each call, display the value passed and the value returned.

35) Write your own `StringBuffer` class. Call it `MyStringBuffer`. Include all the `StringBuffer` methods in your class that are described in Section 8.4. Use a `char` array whose initial size is 2 to hold the string. If on any operation the array is not big enough, a new larger array should be created to which the old array is copied. The new array should then be used. Replace `StringBuffer` in `C9h12.java` with `MyStringBuffer`. Compile and run.

36) Same as homework problem 37 in Chapter 8 except use the following strategy: Create four "cups" numbered 0, 1, 2, 3, each containing the moves 1, 2, and 3. When it is the program's turn, it should select its next move randomly from the cup whose number is given by (number of straws left)%4. If the selected cup is empty, your program should pick up one straw. When a move in a cup is selected, it should not be removed from the cup except in the following two situations:

    a) If the move results in an immediate loss (i.e., if it is greater than or equal to the number of straws).
    b) If the selected cup is cup number 1, and in the previous move made by the program, the selected cup was not empty. In this case, the *previous* move should be removed from the cup from which it was selected.

    Write a class named `Cup` that models a cup as described. It should have a `select` method that selects a move in a cup and a `remove` method that removes a move from a cup. At the end of each game, your program should display the current score (how many games won by the human, how many games won by the machine). It should then ask to user to play another game. On an affirmative response, your program should initiate another game *without reinitializing the cups*. Play 10 or more games with your program. Does your program seem to learn how to play better with each game?

37) Do the slots of arrays have initial values like instance and class variables? Or do they initially contain garbage like local variables? Run a test program to check your answer. Test arrays declared within a method, arrays that are instance variables, and arrays that are class variables.

38) Write a `IntHasher` class according the following specifications:

Data fields:
```
private int[] ia;
private boolean[] available;
private int entryCount;
private long probeCount;
```
Constructors: `public IntHasher()`

Creates `ia` and `available` arrays each with 1000 slots.
Initializes each slot of `available` to `true`.
Sets `entryCount` and `probeCount` to 0.

Methods: `public boolean enter(int x)`

Stores `x` in `ia[Math.abs(x)%1000]` and `false` in `available[Math.abs(x)%1000]` unless these slots are not available (as indicated by the `available` array). If the slots are not available, then `enter` should search sequentially starting with the next slot. If the search is unsuccessful by the end of the array, the search should wrap around to the beginning of the array. If the array is full so that `x` cannot be stored, `enter` should return `false` to its caller. Otherwise, it should increment `entryCount` and return `true`.

`public int getIndex(int x)`
Returns the index at which `x` is stored. Increments `probeCount` on each probe—that is, each time `getIndex` accesses a slot of the `ia` array. Returns `-1` if x is not in the array. `getIndex` should use the same search sequence as `enter`.

`public long getProbeCount()`
returns `probeCount`.

`public void clear()`
Sets `entryCount` and `probeCount` to 0. Sets each slot in `available` to `true`.

Write a program that creates a `IntHasher` object. Your program should generate 100 random integers (call `nextInt` in `Random` without any argument), and enter each one into the `IntHasher` object. After all 100 integers have been entered, call `getIndex` for each integer previously entered. Regenerate the same sequence of integers by setting the seed of the random number generator to its original value by calling `setSeed` (see Fig. 8.6). Then display the average number of probes per integer and the total number of probes. Call `clear` and repeat with 200 integers. Makes addition runs for 300, 400, 500, 600, 700, 800, 900, and 1000 integers.

The number of integers stored divided by the array size is called the **loading factor**. As the loading factor increases, the likelihood of a **collision**—two integers having the same initial index—increases. If all the integers are loaded into the array without any collisions, then the number of probes per integer is one. But if there are collisions, then accessing some of the integers stored in the table requires a sequential search, necessitating additional probes. By keeping the loading factor low, we can keep the average number of probes to a small value. From the data generated by your program, determine the largest loading factor for which the average number of probes is low. How does the average number of probes for this loading factor compare with the number of probes for a serial search? For a binary search?

The technique of storing and retrieving data that is illustrated by this problem is called **hash coding**. Each data item has to be "hashed" to a hash code (the array index). The mathematical function that describes the hashing mechanism is called the **hash function**. For our program, the hash function is h(x) = x%1000.

39) Extend your `IntHasher` class in homework problem 38 so that it keeps track of the maximum number of probes required to access any single integer in the table. Display this maximum along with the average and total number of probes. Run the program with different seeds to see how the statistics produced vary with different sets of random numbers.

40) Write a program that creates a 3-dimensional array and assigns each element the sum of its three indices. Display the contents of the array by the row and column values at successive depths.

41) Write a program that prompts the user for a non-negative integer n. It should then compute and display the value of *n*! (*n* factorial). Your program should be able to compute *n*! for any value of *n* up to 100. To compute such large factorials, use an `int` array `factorial` to hold the successive factorials your program computes. Each slot of the `factorial` array should hold a *single* digit of a factorial. For example, 720 (6!) would be represented with the digits 0, 2, and 7 in `factorial[0]`, `factorial[1]`, and `factorial[2]`, respectively, with 0 in all the remaining slots. Suppose the `factorial` array contains k! To compute (k+1)!

a) Multiply each slot in `factorial` by (k+1).

b) Propagate the carries so that each slot holds a single digit. For example, suppose after the multiplication in step a, 49 is in some slot. Your program should add 4 to the next higher slot and assign 9 to the slot with 49.

Make your program as efficient as possible. For example, avoid unnecessary multiplications in step a. Using your program, determine the values of 0! (by definition, its value is 1), 1!, 2!, 48!, 49!, 50!, 98!, 99!, and 100! Before you run your program, can you predict how many trailing zeros are in each of these factorials?

42) What is displayed when the following program is executed:

```
class C9h42
{
   public static void main(String[] args)
   {
      int[]  a = {1, 2, 3};
      int[]  b = {1};
      char[] c = {'x', 'y', 'z');
      System.out.println(a);
      System.out.println(b);
      System.out.println(c);
   }
}
```

Why is the `c` array displayed differently from the `a` and `b` arrays? *Hint*: See Fig. 8.13.

43) Why do the two programs below produce slightly different answers? Which one is the more accurate answer? *Hint*: See homework problem C4h14 in Chapter 4.

```
class C9h43a
{
   public static void main(String[] s)
   {
      double sum = 0.0;
      for (int i = 1; i <= 100; i++)
         sum = sum + 1.0 / i;
      System.out.println(sum);
   }
}

class C9h43b
{
   public static void main(String[] args)
   {
      double sum = 0.0;
      for (int i = 100; i >= 1; i--)
         sum = sum + 1.0 / i;
      System.out.println(sum);
   }
}
```

44) Write a program that reads in a positive integer followed by that number of additional integers. For example, if the first number is 3, then your program should read in 3 more numbers. Your program should create an array whose length is equal to the first number. It should then read in the remaining numbers into the array. It should then display the numbers in the array from the last one to the first one. Test by entering

```
3 10 20 30
```

45) Write a program in which the `main` method creates an array with10 slots of type `int`. Assign to each slot a randomly-generated integer. Call a function, passing it the array. The called function should *return* the largest integer in the array to your `main` method. Your `main` method should display the number returned. Use a `Random` object to generate integers. Create it with

```
Random r = new Random(7);
```

Generate a random integer with

```
x = r.nextInt();
```

46) Same as question 45, but the called function should return the last number in the array. Do not assume the passed array has ten slots. It should work for an array of any size.

47) Run the following program (it sums up 0.01 one hundred times). Is the sum exactly 1.0? If not, why not?

```
class C9h47
{
   public static void main(String[] args)
   {
      double sum = 0.0;
      for (int i = 1; i <= 100; i++)
         sum = sum + 0.01;
      System.out.println("Sum = " + sum);
   }
}
```

48) Write a program in which `main` creates an `int` array, initializes it with random integers between 100 and 200, then calls a function `reverse`, passing it your array. `reverse` should create an array that has the same size as the array it is passed, copy the passed array into to the new array, but in reverse order, and then return the new array to `main`. `main` should then display the original array and the returned array. `reverse` should handle an `int` array of any size.

49) Write a program that reads in 10 integers from the keyboard into an `ArrayList` and then displays them in reverse order.

50) Write a program in which `main` reads in 10 integers from the keyboard into an `ArrayList`, and then calls `getMax`, passing it the `ArrayList`. Your `getMax` method should determine the maximum element in the `ArrayList` and then return it to `main`. `main` should then display the maximum.

51) Write a program in which `main` creates three `ArrayList`s with the following sizes and contents:

`a1`: size = 11, contains 1, 2, ... 10, 11
`a2`: size = 10, contains 1, 2, ... 9, 10
`a3`: size = 10, contains 1, 2, ..., 9, 11

`main` then calls `compAL` three times with the following arguments:

`a1` and `a2`
`a1` and `a3`
`a1` and `a1`

Your `compAL` method should return true if the two `ArrayList`s it is passed are identical, and false otherwise. On return to `main`, `main` should display `EQUAL` or `NOT EQUAL` according to the value returned by `compAL`. Beware: the `get` method returns an object—not a primitive time. How do you compare objects? With `==` and `!=`? NO!!! But you can use `==` and `!=` if you first auto unbox:

```
int x, y;
...
x = a1.get(i);  // auto unbox
y = a2.get(i);  // auto unbox
if (x != y)
   return false;
```

Warning: Be careful to not use an invalid index into the `ArrayList`. Your program might be doing this when the two `ArrayList`s have difference sizes.

52) Write a program in which `main` creates a `Double ArrayList` containing 10 randomly generated doubles. `main` should then call `makeArray`, passing it the `ArrayList`. `makeArray` should create an array just big enough to accommodate the numbers in the `ArrayList`. It should then copy the numbers in the `ArrayList` to the array. Finally, `makeArray` should return the array. On return, `main` should display the array.

53) Write a program that creates an `ArrayList` containing the numbers 1 to 10. Then read in an integer from the keyboard and insert it in the `ArrayList` so that the numbers are still in ascending order. Display the modified `ArrayList`. Test your program by inserting 5, -1, and 11.

54) Write a complete Java program. When you execute this program you provide two command line arguments that are integer numbers. Your program should display `less`, `equal`, or `greate"` if the first number is less than, equal to, or larger than the second number, respectively. For example, if you invoke the program with

```
java Q5 3 5
```

the program will display `less`. If you invoke the program with

```
java Q5 3 3
```

the program will display `equal`. If you invoke the program with

```
java Q5 5 3
```

the program will display `greater`.

55) What is displayed when the following program is executed?

```
class C9h55
{
   public static void main(String[] args)
   {
      int i, x = 1;
      for (i = 5; i > x ; i++)
      {
         System.out.println("i = " + i);
         System.out.println("x = " + x);
         x = x + 2;
      }
   }
}
```

56) Write a program that reads in a file and then displays the number of occurrences of the letter `e`. Your program should be case insensitive. For example, it should treat `e` and `E` as the same letter. Test your program with the file `gettingstarted.txt`.

57) Same as homework problem 56, but for every letter in the alphabet. Use an array for hold the counts for each letter.

58) Write a program the inputs a file and displays only the last 5 lines of the file. Test your program with the file `gettingstarted.txt`.

59) Write a program that prompts for and reads in non-negative integers from the keyboard. When the user enters a negative number, your program should display all the numbers read in except for the negative number and then terminate. Use an array of length 5 initially. If the array is full and another integer has to be saved, create a new array of twice the length as the current array, copy the old array to the new array, then continue using the new array. Thus, there is no upper limit on the integers that can be saved even though you are using an array to save them. Test your program by entering the integers 1 to 22 followed by -1.

60) A `queue` is a list structure which is processed in a FIFO (first-in-first-out) order. An example of a queue is the line of people waiting to get on a bus. The person at the head of the queue (who was the first to get on the line) is the first person to get onto the bus. Then the next person on the queue gets on the bus, and so on. New arrivals get on at the end of the line. A queue can be simulated in a computer program with an array. To create a queue of integers, use

```
int[] q = new int[100];
int first = 0, last = -1; // are these the best init values for first, last?
```

To add an item `x` to the queue, use

```
q[++last] = x;            // is q[last++] = x better?
// also handle any special cases
```

To remove an integer from the head of the queue and assign it to `x`, use

```
x = q[first++];           // is x = q[++first]; better?
// also handle any special cases
```

Use a queue to save 10 integers from the keyboard. Display the integers as you read them in. Then display the integers a second time by removing them from the queue in the order they were added to the queue. Include a method `isEmpty` that returns `true` when the queue is empty, and `false` otherwise. Use `isEmpty` to determine when the loop that removes and displays the items on the queue should exit.

61) In your program for homework problem 60, each time you do an add and remove, the queue moves up in the array. Thus, eventually, the queue will overflow if you add enough items, even if you remove items as you add new ones so the number of items on the queue never exceeds the size of the array. Fix this problem by moving all the items in the queue down one slot every time a remove operation is performed. Then the first item on the queue will also be in the array slot with index 0.

62) Same a problem 61, but use an `ArrayList`. Note that you do not have to reposition the queue to index 0 of the `Arraylist` every time you do a remove operation because it is done for you by the `ArrayList` mechanism.

63) A better solution than provided by homework problems 61 and 62 that solves the problem of the queue migrating to the end of the array and then overflowing is to wrap around the end of the queue to the beginning of the array when the index in `last` is the index of the last slot in the array. Then, there will be never be an overflow on an add operation unless the number of items on the queue is already at the capacity of the array. Since the last index in an array with 100 slots is 99, when `last` is 99 and an add operation is to be performed, then `last` wraps around to 0, and the add is to the slot at index 0 (assuming the item originally at index 0 has been removed by a previous remove operation). Redo problem 60 use this wrap around technique.

64) Write a method `isFull` for your program in homework problem 63 that returns `true` if the queue is at its maximum size, and `false` otherwise. Can `isFull` determine if the queue size is at its maximum from `first` and `last` alone?