HW 7-2, Week 9.

1. Create and test a class ***RotateRight*** with a method *rotateRight()* that takes an *int* array and rotates the array's contents to the right in place (that is, the input array is changed) by the number of places passed as a parameter.
   - The *rotateRight()* method header is:
     ```
     public static void rotateRight(int[] a, int places)
     ```
   - Numbers that fall off the right should cycle back to the left. For example, if the input array *numbers* is **{ 1, 3, 5, 7 }** and the call of *rotateRight()* is `rotateRight(numbers, 2)` then the *numbers* array should contain **{ 5, 7, 1, 3 }** after that call.
   - Also write or copy a ***display()*** method to print the elements in an *int* array, and use it in *main* to test *rotateRight()* by displaying the elements of various arrays before and after calling the method (make it clear which is before and which is after, and what you expect the result to be).
   - ***Hints*:**
     - **If the length of the input array is 0 or 1 just return immediately (in that case the order of the array elements will not change).**
     - **Set `places = places % a.length;` If `places` is now 0, just return (rotating multiples of the length of the array, including rotating 0 places, does not change it).**
     - **If `places` is negative, set `places = a.length + places;` to turn it into a positive rotation. In the above *numbers* example, rotating right -1 positions is the same as rotating right +3.**
     - **Create a new *int* array *b* the same length as *a*, and in a *for* loop copy the elements of *a*, starting at 0, into the *b* array:**
       ```
       b[places] = a[i]; // i is the for loop index
       places = (places + 1) % a.length;
       // this cycles back to start
       ```
     - **At the end of this loop, run another *for* loop to copy all the elements in *b* back into *a*. Now *a*'s elements have been rotated.**
   - Some possible tests:

Array	places	result	comments
{ 1, 3, 5, 7 }	2	{ 5, 7, 1, 3 }	example above
{ 1, 3, 5, 7 }	18	{ 5, 7, 1, 3 }	18 % 4 == 2
{ 1, 3, 5, 7 }	-1	{ 3, 5, 7, 1 }	4 + -1 == 3
{ 1, 3, 5, 7 }	8	{ 1, 3, 5, 7 }	8 % 4 == 0
{ 1 }	2	{ 1 }	`places` is ignored

2. Do a **modified** version of Chapter 7 Programming Project **2**: Create and test a class ***Palindrome*** with a method *isPalindrome()* that takes a *char* array as its parameter and determines if the characters in that array represent a palindrome, that is, if they read the same forward and backward, ignoring upper and lower case and any blanks.
    - The *isPalindrome()* method header is:
      ```
      public static boolean isPalindrome(char[] a)
      ```
    - Write a program that will accept a sequence of characters ending in a period, read by *Scanner nextLine()*, and will decide whether the *String* without the period is a palindrome, using the *isPalindrome()* method. You can assume that the input contains only letters and blanks before the period, and that it does contain a period at the end of the characters to check.
    - Your *main* method should contain a loop that allows the user to check as many strings as they want until they type ***quit***.
    - You may convert the input *String* to lower case before calling *isPalindrome()*, or you may have *isPalindrome()* do the conversion by using static method ***Character.toLowerCase()***.
    - ***Hints*:**
        - **Convert the input *String* to an array of characters before calling *isPalindrome()* – you can use *String* method *toCharArray()*, but only convert the part of the input *String* before the period:**
          ```
          int period = input.indexOf(".");
          String toCheck = input.substring(0, period);
          char[] ca = toCheck.toCharArray();
          ```
        - **Write a helper method *removeBlanks()* with this header:**
          ```
          private static char[] removeBlanks(char[] a)
          ```
          that will return a new, possibly shorter, character array with any blanks in *a* removed. You can use static method *boolean Character.isWhiteSpace(char c)* to find blanks in *a*.
        - **Write another helper method *reverse()* with this header:**
          ```
          private static char[] reverse(char[] a)
          ```
          that will return a new character array with the original characters in *a* in reverse order.
        - **Finally, call *removeBlanks()* in *isPalindrome()*, then call *reverse()* and compare the two arrays – if they have all the same (lower case) characters, the input is a palindrome.**