CSE 373 Section Handout #1

Numeric Data

1. `binaryZeros`

Write a method named **`binaryZeros`** that accepts an integer *n* as a parameter and returns the number of zeros that occur in the binary representation of *n*. For example, the call of `binaryZeros(44)` returns `3` because the binary representation of 44 is 101100, which contains three zeros.

2. `ones`

Given any integer $0 < n \leq 10000$ not divisible by 2 or 5, some multiple of *n* is a number which in decimal notation is a sequence of 1s. Write a method named **`ones`** that accepts an integer *n* as a parameter and returns how many digits are in the smallest such multiple of *n*. For example, the call of `ones(3)` returns `3` because the smallest multiple of 3 that consists entirely of 1s is 111, which is 3 digits long. Similarly, the call of `ones(7)` returns `6` because the smallest such multiple of 7 is 111111. Note that the sequence of 1s could be very long, too long to store in a standard Java `int` or `double`.

3. `reverseAndAdd`

Write a method named **`reverseAndAdd`** that accepts an integer parameter and repeatedly adds the integer to the reversal of itself (the integer constructed by reversing the order of the digits) until the result is a palindrome (an integer that is the same when the order of its digits is reversed). Your method should print a single line of output containing the number of adds that were needed, followed by a space, followed by the palindrome that was computed. The call of `reverseAndAdd(195);` should print: `4 9339` because:

```
   195 (initial number)
+  591  add #1
   786
+  687  add #2
  1473
+ 3741  add #3
  5214
+ 4125  add #4
  9339 (resulting palindrome)
```

It may help you to define one or more helper methods when solving this problem. Note that the resulting palindrome could be very long, too long to store in a standard Java `int` or `double`.

Text Data

4. `swapPairs`

Write a method named `swapPairs` that accepts a string as a parameter and returns that string with each pair of adjacent letters reversed. If the string has an odd number of letters, the last letter is unchanged. For example, the call of `swapPairs("example")` should return `"xemalpe"` and the call of `swapPairs("hello there")` should return `"ehll ohtree"`.

5. `crazyCaps`

Write a method named `crazyCaps` that accepts a string as a parameter and returns that string with the capitalization altered such that the even *letters* are all in lowercase and odd letters are all in uppercase. Note that non-alphabetic characters do not count as letters. For example, the call of `crazyCaps("Hey!! THERE!")` should return `"hEy!! ThErE!"`.

6. `mostCommon`

Write a method named `mostCommon` that accepts a string parameter and returns the word that occurs most frequently in the string. Words are separated by one or more spaces. Any duplicate words will occur consecutively. If there is a tie, use the word that occurred earlier. You may assume that the string contains at least one word. For example, the following call will return `"Kim"` because the word Kim occurs 3 times:

```
mostCommon("Ben Eric  Eric  Marty Kim Kim Kim  Jenny  Nancy Nancy  Nancy Paul  Paul")
```

CSE 373 Section Handout #1

Arrays

7. arrayMystery

Write the final array contents when the following method is passed each array below:

```
public static void mystery(int[] nums) {
    for (int i = 0; i < nums.length - 1; i++) {
        if (nums[i] > nums[i + 1]) {
            nums[i + 1]++;
        }
    }
}
```

Array	Final array contents
{42}	_____
{14, 7}	_____
{7, 1, 3, 2, 0, 4}	_____
{10, 8, 9, 5, 5}	_____
{12, 11, 10, 10, 8, 7}	_____

8. numUnique

Write a method named `numUnique` that accepts a sorted array of integers as a parameter and that returns the number of unique values in the array. The array is guaranteed to be in sorted order, which means that duplicates will be grouped together. For example, if a variable called `list` stores the following values: `{5, 7, 7, 7, 8, 22, 22, 23, 31, 35, 35, 40, 40, 40, 41}` then the following call: `numUnique(list)` should return `9` because this list has 9 unique values (5, 7, 8, 22, 23, 31, 35, 40 and 41). It is possible that the list might not have any duplicates. If `list` instead stored this sequence of values: `{1, 2, 11, 17, 19, 20, 23, 24, 25, 26, 31, 34, 37, 40, 41}` then a call on the method would return `15` because this list contains 15 different values.

9. stretch

Write a method named `stretch` that accepts an array of integers as a parameter and returns a new array twice as large as the original, replacing every integer from the original array with a pair of integers, each half the original. If a number in the original array is odd, then the first number in the new pair should be one higher than the second so that the sum equals the original number. For example, if a variable named `list` refers to an array storing the values `{18, 7, 4, 24, 11}`, the call of `stretch(list)` should return a new array containing `{9, 9, 4, 3, 2, 2, 12, 12, 6, 5}`. (The number 18 is stretched into the pair 9, 9, the number 7 is stretched into 4, 3, the number 4 is stretched into 2, 2, the number 24 is stretched into 12, 12 and the number 11 is stretched into 6, 5.)

10. rotateRight

Write a method named `rotateRight` that accepts an array of integers as a parameter and rotates the values in the array to the right (i.e., forward in position) by one. Each element moves right by one, except the last element, which moves to the front. For example, if a variable named `list` refers to an array containing the values `{3, 8, 19, 7}`, the call of `rotateRight(list)` should modify it to store `{7, 3, 8, 19}`. A subsequent call of `rotateRight(list)` would leave the array as follows: `{19, 7, 3, 8}`

CSE 373 Section Handout #1
Solutions

1.

```
public static int binaryZeros(int n) {
    String binary = Integer.toString(n, 2);
    int zeros = 0;
    for (int i = 0; i < binary.length(); i++) {
        if (binary.charAt(i) == '0') {
            zeros++;
        }
    }
    return zeros;
}
```

2.

```
public static int ones(int n) {
    BigInteger bn = BigInteger.valueOf(n);
    BigInteger ones = BigInteger.ONE;
    String onesStr = "1";
    while (!ones.mod(bn).equals(BigInteger.ZERO)) {
        onesStr += "1";
        ones = new BigInteger(onesStr);
    }
    return onesStr.length();
}
```

3.

```
public static void reverseAndAdd(int n) {
    BigInteger bi = BigInteger.valueOf(n);
    int iter = 0;
    while (!isPalindrome(bi)) {
        BigInteger rev = reverseDigits(bi);
        bi = bi.add(rev);
        iter++;
    }
    System.out.println(iter + " " + bi);
}

private static BigInteger reverseDigits(BigInteger bi) {
    StringBuilder sb = new StringBuilder(bi.toString());
    sb.reverse();
    return new BigInteger(sb.toString());
}
private static boolean isPalindrome(BigInteger bi) {
    String s = bi.toString();
    for (int i = 0; i < len / 2; i++) {
        if (s.charAt(i) != s.charAt(s.length() - 1 - i)) {
            return false;
        }
    }
    return true;
}
```

4.

```
public static String swapPairs(String s) {
    String result = "";
    for (int i = 0; i < s.length() - 1; i += 2) {
        result += s.charAt(i + 1);
        result += s.charAt(i);
    }
    if (s.length() % 2 != 0) {
        result += s.charAt(s.length() - 1);
    }
    return result;
}
```

CSE 373 Section Handout #1

5.

```
public static String crazyCaps(String s) {
    String result = "";
    int letterCount = 0;
    for (int i = 0; i < s.length(); i++) {
        char c = s.charAt(i);
        if (Character.isLetter(c)) {
            if (letterCount % 2 == 0) {
                result += Character.toLowerCase(c);
            } else {
                result += Character.toUpperCase(c);
            }
            letterCount++;
        } else {
            result += c;
        }
    }
    return result;
}
```

6.

```
public static String mostCommon(String s) {
    Scanner words = new Scanner(s);
    String mostWord = words.next();
    int most = 1;
    String current = mostWord;
    int count = 1;
    while (words.hasNext()) {
        String next = words.next();
        if (next.equals(current)) {
            count++;
            if (count > most) {
                most = count;
                mostWord = current;
            }
        } else {
            current = next;
            count = 1;
        }
    }
    return mostWord;
}
```

7.

Array	Final array contents
{42}	{42}
{14, 7}	{14, 8}
{7, 1, 3, 2, 0, 4}	{7, 2, 3, 3, 1, 4}
{10, 8, 9, 5, 5}	{10, 9, 9, 6, 6}
{12, 11, 10, 10, 8, 7}	{12, 12, 11, 11, 9, 8}

8.

```
public static int numUnique(int[] list){
    if (list.length == 0) {
        return 0;
    } else {
        int count = 1;
        for (int i = 1; i < list.length; i++) {
            if (list[i] != list[i - 1]) {
                count++;
            }
        }
        return count;
    }
}
```

9.

```
public static int[] stretch(int[] list) {
    int[] result = new int[2 * list.length];
    for (int i = 0; i < list.length; i++) {
        result[2 * i] = list[i] / 2 + list[i] % 2;
        result[2 * i + 1] = list[i] / 2;
    }
    return result;
}
```

10.

```
public static void rotateRight(int[] list) {
    int last = list[list.length - 1];
    for (int j = list.length - 1; j > 0; j--) {
        list[j] = list[j - 1];
    }
    list[0] = last;
}
```