Due: Thursday, March 31st by the start of class.

Be sure to include the appropriate collaboration statement as comments at the top of your submitted program. Failure to do so will result in a 10 point deduction.

Move into your CS170 directory:
```
cd cs170
```
Make a directory for hw3:
```
mkdir hw4
```
And move into that directory:
```
cd hw4
```
You can now start up gedit:
```
  gedit &
```

Name your program `Homework4.java`. Write the following methods. Follow the specified naming conventions EXACTLY (capitalization, parameter order, etc).

Strategy: I HIGHLY recommend you write one method at a time, test it, and verify it works before moving on to another method. While it is not required, I also recommend you write some of these methods with while loops and some with for loops to get practice with both of them.

1. Write a method named `avgLength` which takes an array of Strings as an input parameter and returns a double. The method should calculate and return the average length of all the strings in the array.
   Examples:
   ```
   avgLength({“Hello”, “Q”}) returns 3.0
   avgLength({}) returns 0.0
   avgLength({“Hello”, “Goodbye”}) returns 6.0
   ```

2. Write a function named `sumOfDiffs` which takes an array of integers as input and returns an integer. The return value is calculated by summing up the values that occur when you subtract each element from the preceding element.
   Examples:
   ```
   sumOfDiffs({3, 4, 5}) returns -2
   sumOfDiffs({4, 1, 19, 6}) returns -2
   sumOfDiffs({}) returns 0
   sumOfDiffs({3, 0, -1}) returns 4
   ```
3. Write a function named `studentAverages` which takes a 2D array of integers as input. Each column in the 2d array is an assignment and each row is composed of grades for a particular student. (See below for an example). Your function should return an array of doubles representing the grades for each student. You may assume each assignment is scored out of 100 points.

<table>
<thead>
<tr>
<th></th>
<th>Quiz 1</th>
<th>Quiz 2</th>
<th>Quiz 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maggie Simpson</td>
<td>50</td>
<td>100</td>
<td>0</td>
</tr>
<tr>
<td>Lisa Simpson</td>
<td>100</td>
<td>100</td>
<td>80</td>
</tr>
</tbody>
</table>

`studentAverages( { {50,100,0}, {100,100,80} ) would return {50, 93.33333}

#4 below implements portions of the Luhn checksum algorithm. Most e-commerce websites these days take credit cards. Users must enter their credit card number and the merchant verifies that the number is valid. Then Visa, Mastercard, or AmEx process the payment to the merchant and pass the bill along to the user. However, users often mistype their credit card number by one or two digits. These common errors are why credit cards are designed with a secret. Using just the credit card number, we can detect (most) mistakes and errors caused by user mistakes/mistypes. The credit card number contains an error control code called a “checksum”. Specifically, the credit card number is formatted to comply with a Luhn-10 checking algorithm. In #4 you will write a method to calculate the checksum for a given credit card number.

The Luhn-10 algorithm is a weighted algorithm. Each digit in the credit card number is multiplied by a weight. These weights are then summed, forming the checksum. The checksum is divided by 10. If the remainder is 0, the credit card number is valid. If the remainder is NOT 0, the user made an error and can be prompted to re-enter their credit card data. The weighting for the Luhn-10 algorithm is as follows:

- Beginning with the first (ie leftmost) digit in the credit card, every other number is multiplied by 2.
  - If the product results in a 2 digit number (eg 6 x 2 = 12) then the individual digits (eg 1 and 2) are added to the checksum.
- The remaining digits of the credit card number are simply added to the checksum. That is, their weight is 1.
- Several (small) examples are given below, but this algorithm will work with your Visa or Mastercard number. Try it!:

4. Write a method named `luhnChecksum` which takes an array of integers as an input parameter and returns the integer checksum computed by the above algorithm.

Examples:

- `luhnChecksum({4,5,6,3,9,2})` returns 30 (see below for full calculation)
- `luhnChecksum({4,9,9,1,6,5, 7})` returns 40 (see below for full calculation)
Example Number 1: 456392

digit:  
\[
\begin{array}{cccccc}
4 & 5 & 6 & 3 & 9 & 2 \\
\end{array}
\]
multiplied by:  
\[
\begin{array}{cccccc}
2 & 1 & 2 & 1 & 2 & 1 \\
\end{array}
\]

product:  
\[
\begin{array}{cccccc}
8 & 5 & 12 & 3 & 18 & 2 \\
\end{array}
\]
checksum:  
\[
8 + 5 + 1+2 +3 + 1+8+ 2 = 30
\]
Conclusion: This is a valid number since 30 % 10 == 0!

Example Number 2: 4991657

digit:  
\[
\begin{array}{ccccccc}
4 & 9 & 9 & 1 & 6 & 5 & 7 \\
\end{array}
\]
multiplied by:  
\[
\begin{array}{ccccccc}
2 & 1 & 2 & 1 & 2 & 1 & 2 \\
\end{array}
\]

product:  
\[
\begin{array}{ccccccc}
8 & 9 & 18 & 1 & 12 & 5 & 14 \\
\end{array}
\]
checksum:  
\[
8 + 9 +1+8 + 1 + 1+2 +5 + 1+4 = 40
\]
Conclusion: This is a valid number since 40 % 10 == 0!

Submission;
Make sure to include comments and your collaboration statement. Then, from your cs170/hw4 directory, run the command:

```
/home/cs170001/turnin  Homework4.java  hw4
```

You will see a message containing “+++ALLOWED” if your submission was successful. If you had previously submitted, you will be asked to verify you wish to overwrite a previous submission.