CMSC B113 - Computer Science 1
Fall 2020
Homework Assignment #5

Overview

Credit card companies and banks use built-in numerical security measures when creating the account numbers on credit cards. This means that there are only certain valid credit card numbers, and validity can quickly be detected by using various mathematical operations.

In this assignment, you will write Java methods to implement a simple validation algorithm on a credit card number. Note that this algorithm is purely made up; don't try to use it to create fake credit card numbers! ☺

This assignment is broken up into smaller parts in which you will implement different components of the validation algorithm. For each part, first read the explanation, then read it again! Then devise a solution to that part by clearly identifying all the inputs and outputs and sketching out the steps and calculations that will need to be performed. Code the steps into a Java program, then compile and run it. Double-check the program's answers with the ones we have provided to confirm that the results are correct. Start early. Do not wait until it is almost due.

Learning Outcomes

In completing this assignment, you will learn to:

- Implement static methods in Java
- Assemble multiple methods into a single program
- Pass arrays as arguments to methods
- Write methods that use arrays as return values

Code Understandability and Readability

In addition to producing the correct output, in this assignment you are also asked to write code that is easy to read and understand. In particular, part of your score on this assignment will be determined by:

- **Variable naming:** Variables should have meaningful names that indicate what they represent, using full English words or common abbreviations, e.g. "wins" or "votes" instead of "w" or "vot". Variable names should start with a lowercase letter and use "camelCaps", e.g. "creditCardNumber" instead of "creditcardnumber" or "credit_card_number".

- **Method naming:** Methods should have meaningful names that indicate what they do. Like variable names, they should use full English words or common abbreviations, start with a lowercase letter, and use camelCaps.

- **Appearance:** The code should be formatted so that indentation and spacing make it easy to understand which parts of the code are within the bodies of if-statements and loops. Additionally, there should be spacing between variables and operators to make it easy to read each individual line of code.

For example, a method that determines the maximum value in an array should look something like this:

```
public static int findMax(int[] values) {
    int max = values[0];
    for (int i = 1; i < values.length; i++) {
        if (values[i] > max) {
            max = values[i];
        }
    }
    return max;
}
```

Not like this:

```
public static int method(int[] values) {
int a=values[0];
for(int i=1;i<values.length;i++) {
if(values[i]>a) {
a=values[i];
}}
return a;
}
```

Please speak with your Instructor if you have any questions about this requirement!

Before You Begin

Download **CreditCardValidation.java** and review the **`getCreditCardNumber()`** method, which reads a credit card number from StdIn as a **`String`**, and then converts each character to an **`int`** and stores it in an array, which is the return value of this method.

Be sure you are able to compile this code. Note that its **`main()`** method does not call **`getCreditCardNumber()`**, so nothing will happen when you run this program. You will address that in Part 1.

Part 1: Printing Credit Card Number to StdOut

In **CreditCardValidation.java**, implement a method called **`printCreditCard()`** that takes an **`int`** array as its parameter and prints all elements of the array one at a time, on the same line, using **`StdOut.print()`**.

The **`printCreditCard()`** method that you implement must be **`static`** and its return type should be **`void`**, since it is not returning any value.

Then, modify the program's **`main()`** method so that it calls the **`getCreditCardNumber()`** method that we provided to you, stores the return value of that method in an **`int`** array, and then passes that array to your **`printCreditCard()`** method so that it will be displayed in StdOut.

Compile your code to make sure that the **`main`** method is able to call the other two methods.

HINT: If you get a compiler error message that reads "non-static method printCreditCard cannot be referenced from a static context", that means your **`printCreditCard()`** method isn't **`static`**! All methods in this assignment need to be declared as **`static`**.

Once your code compiles, you can test your code as follows:

1. Run the program using the command **`java-introcs CreditCardValidation`**
2. Type a 16-digit number into the Terminal
3. Your program should display the number and terminate

For example, (underlining indicates your typing):

```
$ java-introcs CreditCardValidation
5723563670089094
5723563670089094
$
```

Be sure you complete this part before moving on to Part 2!

Part 2: Implementing Validation Checks

Now that your program is able to read a credit card number from stdin and store the digits in an array, you can implement the methods that will perform the checks to determine whether the credit card number is valid.

The rules that we will use for a valid credit card number are as follows:

1. If the third digit is even, the fourth must be odd; if the third digit is odd, the fourth must be even.
2. The second digit must either be a zero or equal to the sum of the ninth and tenth digits.
3. The sum of all digits must be evenly divisible by 4.
4. If you treat the first two digits as a two-digit number, and the last two digits as a two-digit number, their sum must be 100.

Note that the rules above refer to a "1-based index," so that the "first" digit would be the one at array index #0 and so on.

Implement each of the rules as a separate method:

- Each method must be declared as **`public`** and **`static`**
- The input to each method should be an **`int`** array
- The return value of each method should be a **`boolean`** indicating whether or not the values in the array conform to the rule

Be sure to use meaningful names when writing these methods!

Implement and test one method at a time. Think about how you would do this before proceeding.

Next, write a public, static method called **`validateCreditCard()`** that takes an **`int`** array as its parameter and returns **`true`** if all of the individual validation methods return **`true`**, and returns **`false`** if any of the individual validation methods returns **`false`**.

Last, modify your program's **`main()`** method so that it calls **`validateCreditCard()`** using the **`int`** array that was returned by **`getCreditCardNumber()`**, and then displays the card number using your **`printCreditCard()`** method from Part 1, followed by "**`Valid`**" if the number is valid and "**`Not valid`**" otherwise.

Test your program by running it as described above, and then entering a 16-digit number into the Terminal. A valid credit card number that you can use for testing is 2581198805442775.

HINT: If your program does not appear to be working correctly, e.g. if it says that valid credit card numbers are invalid or vice-versa, try to debug it by displaying the return value from each of the individual validation methods to see which one is producing the wrong result!

Part 3: Validating Multiple Credit Card Numbers

Now that your program is able to validate a single credit card number, modify the main method so that it continues to loop as long as there is more to read from stdin (review your Homework #4 solution if you don't remember how to do this!) and continues to call **`getCreditCardNumber()`**, **`validateCreditCard()`**, and **`printCreditCard()`** in order to indicate whether each credit card number it reads is valid or not.

Download two data files:

1. **`numbers.txt`**
   contains 10,000 credit card numbers, mostly valid, but some invalid.
2. **`shortlist.txt`**
   contains ten credit card numbers, again mostly valid, but a few invalid.

First, test your program on **`shortlist.txt`** file as follows:

```
$ java-introcs CreditCardValidation < shortlist.txt
8643043906544314  Valid
9014279557929210  Valid
4117390134769160  Invalid
4056577086181460  Valid
4090318344221160  Valid
...and five more...
```

Observe and check your program's output to make sure it is correct. Once done, you can try your program on the larger file, **`numbers.txt`**:

```
$ java-introcs CreditCardValidation < shortlist.txt
```

The output will produce 10,000 lines! Do not include this output in your submission. Instead, for this file we may just be interested in seeing how many of them were valid (or invalid). We can do this using a combination of Bash commands: **`grep`** and **`wc`**.

First try **`wc`**:

```
$ wc shortlist.txt
 10  10 170 shortlist.txt
```

`wc` stands for "word count". When run on a file, it counts the number of lines (first number), the number of words (the second number), and the number of bytes (third number), followed by the name of the file. As you can see from above, there are 10 lines, 10 words (each number is one word), and 170 bytes in the **`shortlist.txt`** file. Next, you can try the command on **`numbers.txt`**.

grep is a very useful command to quickly search through a file for a given pattern (**grep** stands for “get regular expression pattern”). For example, we can search for the pattern “6181” in the **shortlist.txt** file:

```
$ grep 6181 shortlist.txt
4056577086181460
```

As you can see, it finds a line that has that pattern. You can try another pattern (see the five numbers from shortlist above). Try finding the pattern, “00000”.

Now, to see the invalid credit card numbers in a file, we can use the output from our **CreditCardValidation** program and then pipe its output through a **grep** command to search for the pattern: **Valid** as shown below:

```
$ java-introcs CreditCardValidation < shortlist.txt | grep Valid
8643043906544314  Valid
9014279557929210  Valid
4056577086181460  Valid
4090318344221160  Valid
…plus some more…
```

Try the same command as above to find all the invalid numbers.

Finally, to count the number of valid/invalid numbers, you pipe the output of **grep** into **wc**:

```
$ java-introcs CreditCardValidation < shortlist.txt | grep Valid | wc
      8      16     192
```

That is, there are 8 valid numbers in **shortlist.txt**. Try to find the number of invalid numbers in **shortlist.txt**. Now, you are ready to count the number of valid and invalid numbers in the large **numbers.txt** file.

In your submission, you will (1) include output from **shortlist.txt** and (2) ONLY the number of invalid numbers in the file, **numbers.txt**.

Part 4: Generating a Valid Credit Card Number

Now that you have a program that will validate credit card numbers, we'll add functionality to generate a valid number.

Write a **`public`**, **`static`** method called **`generateCreditCardNumber()`** that randomly generates a credit card number as a 16-element **`int`** array and passes it to the **`validateCreditCard()`** method to determine whether it is valid.

The method should continue generating numbers until it creates one that is valid. Once it generates a valid credit card number, the **`generateCreditCardNumber()`** method should return the **`int`** array to the **`main()`** method, where it should be displayed using the **`printCreditCard()`** method you wrote in Part 1.

Last, modify your **`main()`** method so that it uses the first runtime argument to determine whether to validate credit card numbers (Parts 1-3) or to generate a credit card number (Part 4). So if the program is run like this, with **0** as the runtime argument:

```
java-introcs CreditCardValidation 0 < shortlist.txt
```

then the program should validate all the credit card numbers in the input file

However, if it is like this, with **1** as the runtime argument:

```
java-introcs CreditCardValidation 1
```

then it should generate a credit card number.

Your program should display an error message and terminate if there is no runtime argument (think about how you would detect this!) or if the first runtime argument is neither 0 nor 1. You can assume that, if it exists, the first runtime argument is an integer.

Submitting Your Solution

To submit your work, copy/paste the source code of your Java program, as well as the output produced in Part 3, into a *single* PDF, and submit the PDF in the "Assignment #5" subfolder of the Dropbox folder that your instructor created for you.

Only submit the final version of **CreditCardValidation.java**; you do not need to submit intermediate solutions for each of the separate parts.

Please be sure to put all of your code into a *single* document and please only submit a PDF, not a Microsoft Word .docx file or the .java source files. Your work will not be graded if you submit anything other than a PDF file.