Lab 3 Document Henan Mu

Problem 1

Problem description

We were given a credit card number, and we need to validate credit card numbers, whether it is valid or invalid. We validate it by its size, prefix, and numbers. For the numbers, we use the Luhn check or the Mod 10 check algorithms.

Analysis

main()

First of all, we use `Scanner` to read the input as a string.

And then, we use the function `Long.parseLong()` to convert the string to a long variable.

Use function `isValid(long number)` to validate the variable, and get the result. So the main work is on the function `isValid()`.

isValid()

We validate the credit card number in three aspects:

1. Size
   - Firstly, we use `getSize(long number)` to get the size and validate it. If it's between 13 and 16, we continue. Otherwise, we return false.
2. prefix
   - Secondly, we check its prefix by `prefixMatched(long number, int d)` . if its prefix is valid, we continue. Otherwise, we return false.
3. Numbers
   - Thirdly, we use the Luhn check algorithms to validate the numbers. According to the algorithms, we get a result by `sumOfDoubleEvenPlace(number) + sumOfOddPlace(number)`, if it is divisible by 10, the card number is valid; otherwise, it is invalid.

sumOfDoubleEvenPlace()

For this function we need to iterate every second digit from right to left. I thought about converting the number to a string to do this task, but finally, I chose to do that in a purely mathematical way, to be more specific, I used division and modulo.

For an integer, dividing it by ten means removing the last digit of this number, and modulo it by ten means getting the last digit of this number. So we can get a second digit from right to left by dividing the number by ten and modulo it by ten. Then, we divide it by 100 and modulo it by ten we get another second digit. By doing this process, again and again, we can iterate every second digit from right to left.

For each second digit, we double it. If doubling of a digit results in a two-digit number, add up the two digits to get a single-digit number (we do that by `getDigit(int number)`).

Finally, add all single-digit numbers from above. The key codes of this function are like that:

```
number /= 10;
while (number > 0) {
    resutl += getDigit(2 * ((int)number % 10));
    number /= 100;
}
```

getDigit()

Return this number if it is a single digit, otherwise, return the sum of the two digits.

I compare the number with 10, if it is less than 10, it is a single digit, return it, otherwise, there are two digits, return `(number % 10) + (number / 10)`.

sumOfOddPlace()

This function is similar to `sumOfDoubleEvenPlace()`, now we need to iterate digits in the odd places from right to left in the card number and sum these digits. Key codes are like that:

```
while (number > 10) {
    resutl += number % 10;
    number /= 100;
}
```

prefixMatched()

For prefix matching, I still chose to use a purely mathematical way. We can remove the last digits of the number and then compare it with the prefix, if they are equal, the prefix matched!! We iterate all the possible prefixes of the number by removing the last digit one by one. Key codes are like that:

```
while (number > 0) {
    if (number == d) return true;
    number /= 10;
}
return false;
```

getSize()

We remove the last digit of the number one by one until the number is 0, at the same time, we count the times. Finally, the times are size.

getPrefix()

Firstly, we get the size of the number by `getSize()`.

If the size is not more than k, return the number.

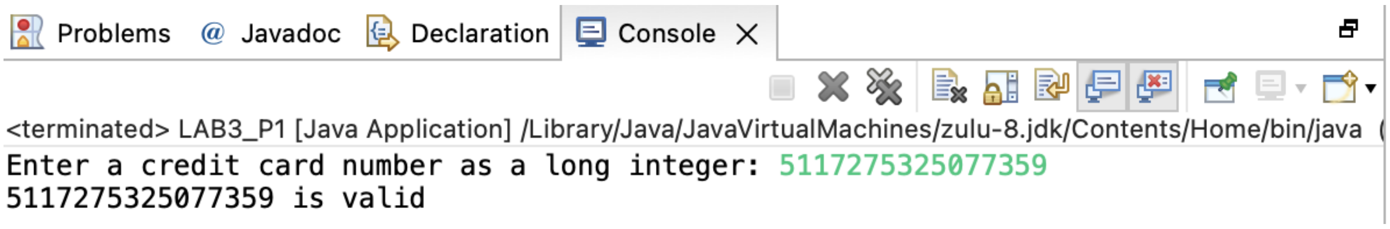
Otherwise, we remove the last `size - k` digits of the number and return it.

Source code

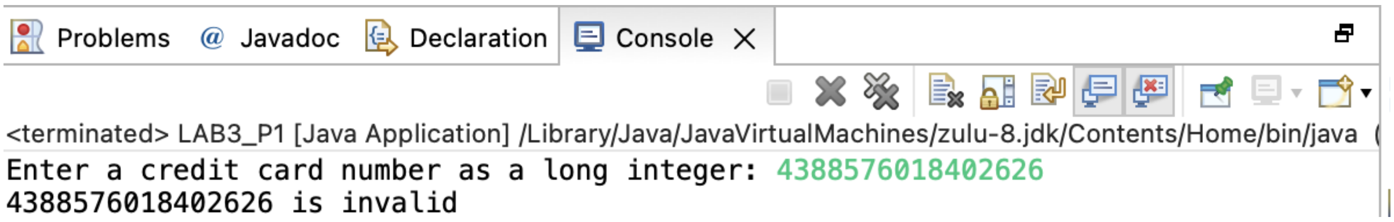
```
package edu.northeastern.csye6200;

import java.util.Scanner;

public class LAB3_P1 {
    public static void main(String[] args) {
        // TODO: write your code here
        Scanner my_sc = new Scanner(System.in);
        System.out.print("Enter a credit card number as a long integer: ");
        String cardNumberString = my_sc.nextLine();
        long cardNumberLong = Long.parseLong(cardNumberString);
        if (isValid(cardNumberLong)) {
            System.out.println(cardNumberString + " is valid");
        }
        else {
            System.out.println(cardNumberString + " is invalid");
        }
        my_sc.close();
    }

    /** Return true if the card number is valid */
    public static boolean isValid(long number) {
        // TODO: write your code here
        if (getSize(number) >= 13 && getSize(number) <= 16) {
            if (prefixMatched(number, 4) || prefixMatched(number, 5) ||
prefixMatched(number, 6) || prefixMatched(number, 37)) {
                int check = sumOfDoubleEvenPlace(number) + sumOfOddPlace(number);
                if (check % 10 == 0) return true;
            }
        }
        return false;
    }

    /** Get the result from Step 2 */
    public static int sumOfDoubleEvenPlace(long number) {
        // TODO: write your code here
        int resutl = 0;
        number /= 10;
        while (number > 0) {
            resutl += getDigit(2 * ((int)number % 10));
            number /= 100;
        }
        return resutl;
    }

    /**
     * Return this number if it is a single digit, otherwise, return the sum of
```

```java
     * the two digits
     */
    public static int getDigit(int number) {
        // TODO: write your code here
        if (number < 10) return number;
        else return ((number % 10) + (number / 10));
    }

    /** Return sum of odd place digits in number */
    public static int sumOfOddPlace(long number) {
        // TODO: write your code here
        int resutl = 0;
        while (number > 10) {
            resutl += number % 10;
            number /= 100;
        }
        return resutl;
    }

    /** Return true if the digit d is a prefix for number */
    public static boolean prefixMatched(long number, int d) {
        // TODO: write your code here
        while (number > 0) {
            if (number == d) return true;
            number /= 10;
        }
        return false;
    }

    /** Return the number of digits in d */
    public static int getSize(long d) {
        // TODO: write your code here
        int size = 0;
        while (d > 0) {
            size++;
            d /= 10;
        }
        return size;
    }

    /**
     * Return the first k number of digits from number. If the number of digits
     * in number is less than k, return number.
     */
    public static long getPrefix(long number, int k) {
        // TODO: write your code here
        int size = getSize(number);
        if (size <= k) return number;
        else {
            int numberBeRemoved = size - k;
            for (int i = 0; i < numberBeRemoved; i++) number /= 10;
            return number;
        }
    }
}
```

Screenshots of sample runs



```
<terminated> LAB3_P1 [Java Application] /Library/Java/JavaVirtualMachines/zulu-8.jdk/Contents/Home/bin/java (
Enter a credit card number as a long integer: 5117275325077359
5117275325077359 is valid
```



```
<terminated> LAB3_P1 [Java Application] /Library/Java/JavaVirtualMachines/zulu-8.jdk/Contents/Home/bin/java (
Enter a credit card number as a long integer: 4388576018402626
4388576018402626 is invalid
```

Problem 2

Problem description

We were given a number of values, and some values. we need to test whether the array has four consecutive numbers with the same value.

Analysis

main()

First of all, we read all the input. And then we need to create a array to save all the values. We pass this array as a parameter into `isConsecutiveFour()` to test this array.

isConsecutiveFour()

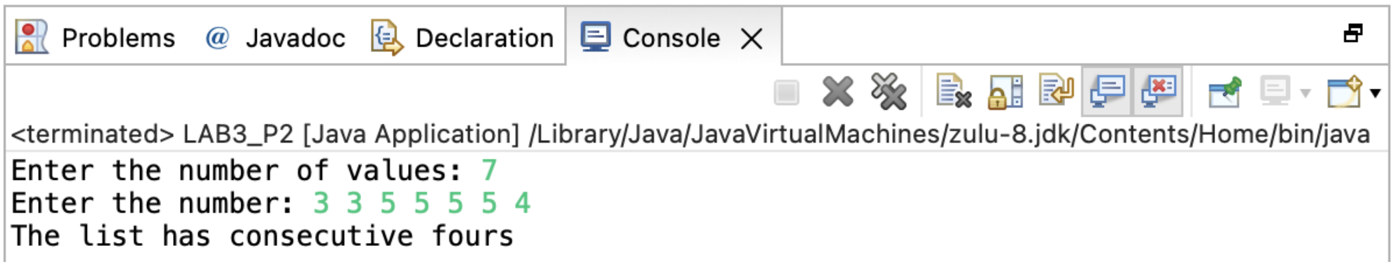
Firstly, I check the size of the array, if the size is less than 4, the array definitely doesn't have four consecutive numbers with the same value.

Secondly, my algorithm is to iterate the array once. While iterating the array, compare the current the element with the prior one, if they are equal, count the consecutive numbers, otherwise, reset the counter.

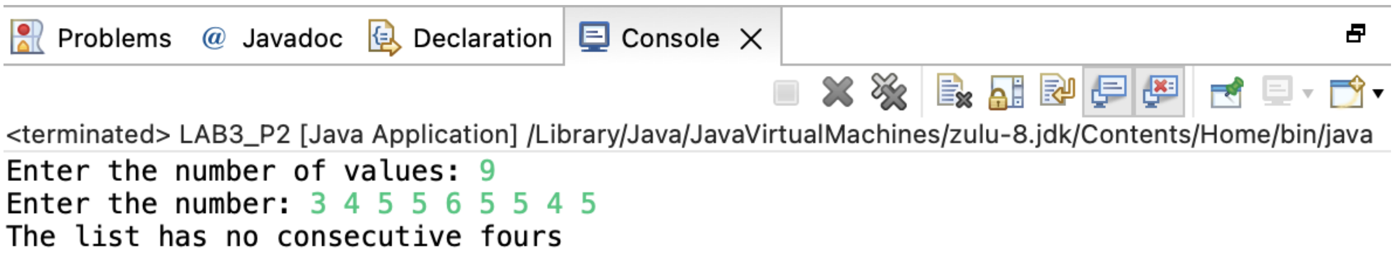
Source code

```
package edu.northeastern.csye6200;

import java.util.Scanner;

public class LAB3_P2 {
    public static void main(String[] args) {
        // TODO: write your code here
        Scanner my_sc = new Scanner(System.in);
        System.out.print("Enter the number of values: ");
        int size = my_sc.nextInt();
        int[] values = new int[size];
        System.out.print("Enter the number: ");
        for (int i = 0; i < size; i++) {
            values[i] = my_sc.nextInt();
        }
        if (isConsecutiveFour(values)) {
            System.out.println("The list has consecutive fours");
        } else {

            System.out.println("The list has no consecutive fours");
        }
        my_sc.close();
    }

    public static boolean isConsecutiveFour(int[] values) {
        // TODO: write your code here
        int size = values.length;
        if (size < 4) return false;
        int prior = values[0];
        int numberOfConsecutive = 1;
        for (int i = 1; i < size; i++) {
            if (prior == values[i]) {
                numberOfConsecutive++;
                prior = values[i];
                if (numberOfConsecutive == 4) return true;
            } else {
                numberOfConsecutive = 1;
                prior = values[i];
            }
        }
        return false;
    }
}
```

Screenshots of sample runs



```
<terminated> LAB3_P2 [Java Application] /Library/Java/JavaVirtualMachines/zulu-8.jdk/Contents/Home/bin/java
Enter the number of values: 7
Enter the number: 3 3 5 5 5 5 4
The list has consecutive fours
```



```
<terminated> LAB3_P2 [Java Application] /Library/Java/JavaVirtualMachines/zulu-8.jdk/Contents/Home/bin/java
Enter the number of values: 9
Enter the number: 3 4 5 5 6 5 5 4 5
The list has no consecutive fours
```