Lab 3 - Assignment 1

Jasmine Chen #002980795

Problem 1:

- ### Problem Description:

This problem is to validate a credit card. Credit card validation has increased in its importance nowadays. According to the Federal Reserve Bank of San Francisco, about 28% of the daily transactions are made by credit card. Meanwhile, the world is relying on e-commercials more and more, as for the US, 14.8% of the retail sales are online which built up to 905 billion dollars last year.

Since credit card numbers are ruled in a specific pattern and many simple calculations are included, it is one of the best examples to have computers solve for us. Also, because the pattern of credit cards relies on the odd/ even digits, it is easy to transfer this problem solving skills to potentially similar problems.

- ### Analysis:

I found three details in this problem that I might need to think of a while. First, how to move two digits once and calculate it; second, how to get the first n digits left; third, how to know the total length without casting to a string.

According to the rules, I need to make sure the given credit card number is between 13 to 16 digits first as well as it's from a legal number before the Luhn check. Here I design two methods: getSize, to get the length of the credit card number, and prefixMatched, to check the legal starting number. To get the prefixMatched, I designed another method called getPrefix, because the starting can be either one number or two numbers to match different credit card service providers.

After the pre-check, I can get to the Luhn check algorithm. There are five steps of this algorithm, but in simple words, we group odd numbers and even numbers respectively, do some simple math calculation, and then combine to do another calculation to get to the final decision. So I designed a method, sumOddPlace, to sum up all odd numbers; and sumDoubleEvenPlace, to sum up all doubled even numbers. For odd numbers, it's simply just adding it together; whereas for even numbers, I need to check if it becomes larger than ten after timing two. Thus, I designed another method called getDigits, to make sure it adds up the two digits and returns a single digit number after doubling.

The problem here being, how to count digits one by one. I use mod to get the current unit digit, and use loop to divide the number by ten to get to the next digit.

Finally, I designed a method, isValid, to do the fourth and final steps. It calls the methods mention above, sum up, check the divisibility check, and return a Boolean. By this design, I can keep the main method clean and simple, and only handle the inputs and outputs.

- ### Source Code:

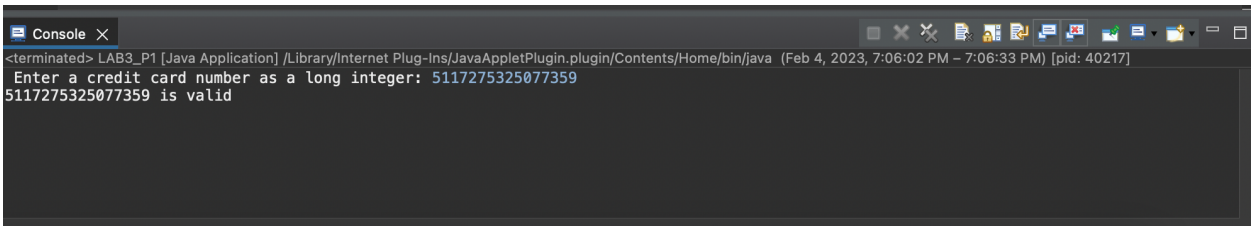
```
package edu.northeastern.csye6200;

import java.util.Scanner;
```

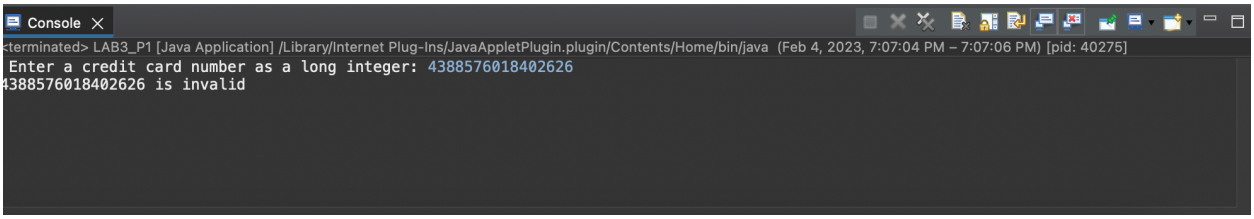
```
public class LAB3_P1 {
	public static void main(String[] args) {
		System.out.print(" Enter a credit card number as a long
integer: ");
		Scanner input = new Scanner(System.in);
		long number = input.nextLong();
		if(isValid(number)) {
			System.out.println(number+" is valid");
		}else {
			System.out.println(number+" is invalid");
		}

	}

	/** Return true if the card number is valid */
	public static boolean isValid(long number) {
		if(getSize(number)<13 || getSize(number)>16) return false;
		if(prefixMatched(getPrefix(number, 1), 4) ||
prefixMatched(getPrefix(number, 1), 5) ||
				prefixMatched(getPrefix(number, 1), 6) ||
prefixMatched(getPrefix(number, 2), 37)) {
				int k =
sumOfDoubleEvenPlace(number)+sumOfOddPlace(number);
				if(k%10==0) return true;
		}
		return false;
	}

	/** Get the result from Step 2 */
	public static int sumOfDoubleEvenPlace(long number) {
		long n=number;
		int i=0,sum=0;

		while(n>0) {
			if(i%2!=0) {
				sum+=getDigit((int)(n%10));
			}
			i++;
			n=n/10;
		}

		return sum;

	}

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

```
	 */
	public static int getDigit(int number) {

		number=number*2;
		if(number<10) return number;
		else return number%10+1;

	}

	/** Return sum of odd place digits in number */
	public static int sumOfOddPlace(long number) {
		long n=number;
		int i=0,sum=0;

		while(n>0) {
			if(i%2==0) {
				sum+=n%10;
			}
			i++;
			n=n/10;
		}

		return sum;
	}

	/** Return true if the digit d is a prefix for number */
	public static boolean prefixMatched(long number, int d) {
		if(number==(long)d) return true;
		return false;
	}

	/** Return the number of digits in d */
	public static int getSize(long d) {
		int l=0; //String.valueOf(d).length();
		while(d>9) {
			d=d/10;
			l++;
		}
		return l;
	}

	/**
	 * 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
		long n=number;
		while(String.valueOf(n).length()>k) {
```

```
            n=n/10;
        }
        return n;
```

- Screenshots:





Problem 2:

- Problem Description:

The problem is to check if an array has consecutive values for four or more. This problem is important because array being a basic data structure, it is easy to apply to real world cases. For example, to avoid typos or to verify a series of numbers.

Also, when the data follows a certain pattern, it is easy to adjust the code by changing the logic check in the method. For example, to check if a series of number keeps increasing, or if a series of numbers differ by the same amount.

- Analysis:

In this problem, I only design one extra method other than main. Similar to problem one, the main method only handles the inputs and outputs to keep it clean and readable. Because the inputs contain multiple integers, I used a for loop to put each integer into an array one by one to make it executable.

The isConsecutiveFour method allows the user to pass in an array as a parameter and returns a Boolean of true or false. I use a for loop to scan through the array and use condition statements to check or to reset to default counting.

- Source Code:

```
package edu.northeastern.csye6200;

import java.util.Scanner;

public class LAB3_P2 {
     public static void main(String[] args) {
          Scanner input = new Scanner(System.in);
          System.out.print("Enter the number of values: ");
          int length = input.nextInt();
          int[] array = new int[length];
          System.out.print("Enter the number: ");
          for(int i=0; i<length; i++) {
               array[i]=input.nextInt();
          }

          if(isConsecutiveFour(array)) {
               System.out.println("The list has consecutive
fours.");
          }else {
               System.out.println("The list does not have
consecutive fours.");
          }
     }
```

```
	public static boolean isConsecutiveFour(int[] values) {
		int count=0;
		for(int i=0; i<values.length-1; i++) {
			if(values[i]==values[i+1]) {
				count++;
				if(count>=3) return true;
			}else {
				count=0;
			}

		}
		return false;
	}

}
```

- Screenshots: