

Project 1 – The Outsourcers

Making a credit card checking program

Ahmed Abubakar, Sabih Ul Huda, Hassan and Sami Sheikh | CSIT111

14 June 2022

# Analysis

The problem in this project was to code a program with multiple methods that would accept 2 parameters (Total number of credit cards AND credit card numbers as long integers in new lines) where we had to then establish outputs in three key fields:

* To display the belonging of the cards and to which companies they belonged (Visa, MasterCard, American Express or Discover Card)
* To display whether the inputted credit card number was valid or invalid based on the provided Luhn Check criteria provided beforehand.
* To suggest fixes to invalid credit card numbers through modification of the final digit ONLY.

# Design

The emphasis of the program was concentrated on developing methods from top-to-bottom. The first step was to allocate each member of our team their tasks along with scopes:

* Mohammad Sabih Ul Huda was responsible for developing the method: “Luhn\_Check\_Test.java” where his output was critical for the implementation of suggested fixes to the credit card number.
* Ahmad Abubakar was in charge of developing the method: “Suggested\_Fix.java” that was responsible for providing and implementing suggestions for invalid numbers to pass the Luhn check.
* Hassan was responsible for developing the method: “card.java” where he created a conditional statement necessary to ensure that the card entry satisfied the minimum and maximum requirements (from 13 to 16 digit characters)
* Sami Sheikh was responsible for the development of the method: “credit\_card\_validation\_starting\_digits.java” which was essential to establish the categorisation of the cards based on their first two digits.

Diagram

Description automatically generatedAfter the development of the methods was completed, all methods were called exhaustively in “main\_executable.java” where everyones’ contributions were executed with success. The project was named: Project1\_The\_Outsourcers.

# Coding

Code for Length Validation

package src;  
  
import java.util.Scanner;  
  
public class card {  
  
 // Main exists to test code  
 public static void main(String[] args) {  
 Scanner scan = new Scanner (System.*in*);  
 long cardNumber = scan.nextLong();  
  
 }  
 // Function to analyse the length of the card number with a boolean return type  
 public static boolean authentication(long card\_number\_long) {  
  
 // conversion of card\_number\_long to string  
 String card\_number\_string = String.*valueOf*(card\_number\_long);  
  
 // Conditional statement to analyse if the length of the string is between 13 AND 16  
 if (card\_number\_string.length()>= 13 && card\_number\_string.length()<=16) {  
 return true;  
  
 }  
 else return false;  
  
 }  
  
}

Code for Luhn Check Test Method

package src;  
  
public class Luhn\_Check\_Test {  
 public static void main(String[] args) {  
 *correctCredit*(4388576018402626L);  
 }  
  
 // Function to conduct Luhn check  
 public static int correctCredit(long credit) {  
  
 // Initialized variables  
 int totalEven = 0;  
 int total;  
 int odd\_Iteration = 0;  
 int even\_Iteration = 0;  
 int total\_Odd=0;  
  
 // String called to convert number to string  
 String num\_To\_String = credit + "";  
  
 // Iteration Variables  
 int halved\_Even = num\_To\_String.length();  
 int even\_Positions = num\_To\_String.length();  
  
  
 // Iteration to traverse even positions by halving by 2  
 while (even\_Iteration < (halved\_Even / 2)) {  
  
 // Value of even positions are returned and multiplied by 2  
 int value\_Of\_Even\_Position = Integer.*parseInt*(String.*valueOf*(num\_To\_String.charAt(even\_Positions - 2)));  
 int even = 2 \* value\_Of\_Even\_Position;  
  
 String evenum\_To\_String = even + "";  
  
 // Conditional statements to split double digits to single digits for even numbers  
 if (evenum\_To\_String.length() > 1) {  
 char firstValue = evenum\_To\_String.charAt(0);  
 char secondValue = evenum\_To\_String.charAt(1);  
 int first\_Digit = Integer.*parseInt*(String.*valueOf*(firstValue));  
 int second\_Digit = Integer.*parseInt*(String.*valueOf*(secondValue));  
 int DoubleEvenTotal = first\_Digit + second\_Digit;  
 totalEven += DoubleEvenTotal;  
  
 } else {  
 totalEven += even;  
 }  
 even\_Positions -= 2;  
 even\_Iteration++;  
 }  
  
 // Iteration to get the sum of odd position values  
 int odd\_Positions = num\_To\_String.length();  
 while (num\_To\_String.length() > odd\_Iteration) {  
 if(odd\_Positions % 2 !=0){  
 int Odd= Integer.*parseInt*(String.*valueOf*(num\_To\_String.charAt(odd\_Positions)));  
 total\_Odd+=Odd;  
  
 }  
  
 odd\_Positions -= 1;  
 odd\_Iteration++;  
 }  
 // Sum of both odd and even computed and returned  
  
 total = totalEven + total\_Odd;  
 return total;  
 }  
}

Code for credit card type Method

package src;  
import java.util.\*;  
  
public class credit\_card\_validation\_starting\_digits {  
  
 // Method to categorise credit card type  
 public static String credit\_Card\_Category(long credit\_Card\_Number) {  
  
 // Two variables initialised to get the first and first-two digits  
 long credit\_card\_first\_digit = credit\_Card\_Number;  
 long credit\_card\_first\_two\_digits = credit\_Card\_Number;  
  
 // Iteration with division to get the first digit  
 while (credit\_card\_first\_digit >= 9) {  
 credit\_card\_first\_digit /= 10;  
 }  
  
 // Iteration with division to get the first-two digits  
 while (credit\_card\_first\_two\_digits >= 99) {  
 credit\_card\_first\_two\_digits /= 10;  
 }  
  
 // Conditional statements with return type String  
 if (credit\_card\_first\_digit == 4)  
 return "Visa";  
 else if (credit\_card\_first\_digit == 5)  
 return "MasterCard";  
 else if (credit\_card\_first\_digit == 6)  
 return "Discover Card";  
 else if (credit\_card\_first\_two\_digits == 37)  
 return "American Express";  
 else  
 return "Invalid Card";  
  
 }  
}

Code for suggested fix Method

package src;  
  
public class Suggested\_Fix {  
  
 // Method to correct last digit in case of invalidity  
 public static void correctCredit(long credit, long n) { //n is total of credit card numbers needed to confirm validation  
  
 String credits = credit + "";  
  
 if (n % 10 != 0) {  
 int change = (int) (10 - (n % 10)); //difference to make the total to be modded by 10  
  
 String last\_Digit\_From\_String = credits.substring((credits.length() - 1)); //last digit from original card number  
  
 int last\_Digit = Integer.*parseInt*(last\_Digit\_From\_String); // Parsing digit retrieved from lastDigitFromString  
 int fix\_Last = (((last\_Digit) + (change)) % 10); //correct last digit to make card valid  
  
 //if the last digit of the credit is greater than the number that should replace it  
  
 if (last\_Digit > fix\_Last || last\_Digit == fix\_Last) {  
 while (last\_Digit != fix\_Last) {  
 credit -= 1;  
 last\_Digit -= 1;  
 }  
 System.*out*.println("Suggested fix: " + credit); //output:suggest valid credit card number  
  
 // if the last digit of the credit is less than the number that should replace it or equal to it  
  
 } else if (last\_Digit < fix\_Last) {  
 while (last\_Digit != fix\_Last) {  
 credit += 1;  
 last\_Digit += 1;  
  
 }  
 System.*out*.println("Suggested fix: " + credit);//output:suggest valid credit card number  
 }  
 }  
 }  
}

Code for Main method

package src;  
import java.util.\*;  
  
// This class acts as the main executable.  
public class main\_executable {  
 public static void main(String[] args) {  
 Scanner input = new Scanner(System.*in*);  
  
 // User inputs requested for n-number of iterations along with initialisation of the counter "iteration"  
 System.*out*.println("Please enter your number of cards: ");  
 int no\_Of\_Cards = input.nextInt();  
 int iteration = 0;  
  
 // While loop to iterate for each card  
 while (iteration < no\_Of\_Cards) {  
  
 // Request for user input for a credit card number  
 System.*out*.println("\n" + "Please enter your credit card number: ");  
 long credit\_Card\_Number = input.nextLong();  
  
 // Conditional statement to return whether credit card number length is valid or not (Between 13 and 16)  
 if (card.*authentication*(credit\_Card\_Number)) {  
 System.*out*.print(credit\_Card\_Number);  
 } else {  
 System.*out*.print("Your credit card number length is invalid, please try again");  
 break;  
 }  
  
 // Conditional test to check if the card type is valid or not based off the first digit.  
 if (!credit\_card\_validation\_starting\_digits.*credit\_Card\_Category*(credit\_Card\_Number).equals("Invalid Card")) {  
 System.*out*.print(" is " + credit\_card\_validation\_starting\_digits.*credit\_Card\_Category*(credit\_Card\_Number));  
 } else {  
 System.*out*.println("Your credit card type is an invalid card, please try again.");  
 break;  
 }  
  
 // Conditional statement to check whether the card number passes the Luhn check or not.  
 // if not, the correction is implemented in the latter.  
 if (Luhn\_Check\_Test.*correctCredit*(credit\_Card\_Number) % 10 == 0) {  
 System.*out*.print(" & is valid ");  
  
 } else if (Luhn\_Check\_Test.*correctCredit*(credit\_Card\_Number) % 10 != 0) {  
 System.*out*.print(" & it is invalid " + "\n" + "\n");  
 Suggested\_Fix.*correctCredit*(credit\_Card\_Number, Luhn\_Check\_Test.*correctCredit*(credit\_Card\_Number));  
 }  
 iteration++;  
 }  
 }  
}

# Testing Snapshots

Text

Description automatically generatedIDE that was used was IntelliJ version 2022.1, in this text, two cards would be evaluated, the first card being of type VISA and invalid.

Text

Description automatically generated

Upon data entry, the execution of the code had correctly displayed the expected result and had also suggested a fix based on the last digit.

In our second test, we had decided to use a different number, in this instance, the expected output was a card of the type “MasterCard” and be invalid.

Text

Description automatically generated

As expected, the result had met our expectations and had correctly established the category and its invalidity along with the fix.

# Addendum

During the interview, we identified an issue in the original code where we hard-coded the length of the code, and hence it wouldn’t work well with odd length inputs (13-digits, 15-digits).

As such, we have made an addition of a conditional statement that analyses the length of the credit card number such that if the length is odd or even in length, the appropriate string bounds will be used to account for the difference. The code below is the code **Post-Interview** that resolves the identified problem.

// This file contains the fix to the problem regarding 13-digit input.  
// The fix involved the use of a conditional statement that checked whether the length of the string was odd or even, thereby alleviating the input error.  
  
// THIS WAS DONE POST INTERVIEW  
  
  
  
package src;  
  
public class Luhn\_Check\_Test\_Fixed\_After\_Interview {  
 public static void main(String[] args) {  
 *correctCredit*(4388576018402626L);  
 }  
  
 // Function to conduct Luhn check  
 public static int correctCredit(long credit) {  
  
 // Initialized variables  
 int totalEven = 0;  
 int total;  
 int odd\_Iteration = 0;  
 int even\_Iteration = 0;  
 int total\_Odd = 0;  
  
 // String called to convert number to string  
 String num\_To\_String = credit + "";  
  
 if (num\_To\_String.length() % 2 == 0) {  
  
  
 // Iteration Variables  
 int halved\_Even = num\_To\_String.length();  
 int even\_Positions = num\_To\_String.length();  
  
  
 // Iteration to traverse even positions by halving by 2  
 while (even\_Iteration < (halved\_Even / 2)) {  
  
 // Value of even positions are returned and multiplied by 2  
 int value\_Of\_Even\_Position = Integer.*parseInt*(String.*valueOf*(num\_To\_String.charAt(even\_Positions - 2)));  
 int even = 2 \* value\_Of\_Even\_Position;  
  
 String evenum\_To\_String = even + "";  
  
 // Conditional statements to split double digits to single digits for even numbers  
 if (evenum\_To\_String.length() > 1) {  
 char firstValue = evenum\_To\_String.charAt(0);  
 char secondValue = evenum\_To\_String.charAt(1);  
 int first\_Digit = Integer.*parseInt*(String.*valueOf*(firstValue));  
 int second\_Digit = Integer.*parseInt*(String.*valueOf*(secondValue));  
 int DoubleEvenTotal = first\_Digit + second\_Digit;  
 totalEven += DoubleEvenTotal;  
  
 } else {  
 totalEven += even;  
 }  
 even\_Positions -= 2;  
 even\_Iteration++;  
 }  
  
 // Iteration to get the sum of odd position values  
 int odd\_Positions = num\_To\_String.length();  
 while (num\_To\_String.length() > odd\_Iteration) {  
 if (odd\_Positions % 2 != 0) {  
 int Odd = Integer.*parseInt*(String.*valueOf*(num\_To\_String.charAt(odd\_Positions)));  
 total\_Odd += Odd;  
  
 }  
  
 odd\_Positions -= 1;  
 odd\_Iteration++;  
 }  
 // Sum of both odd and even computed and returned  
 } else if (num\_To\_String.length() % 2 != 0) {  
  
  
 // Iteration Variables  
 int halved\_Even = num\_To\_String.length();  
 int even\_Positions = num\_To\_String.length();  
  
  
 // Iteration to traverse even positions by halving by 2  
 while (even\_Iteration < halved\_Even) {  
  
 if (even\_Positions % 2 != 0) {  
  
 // Value of even positions are returned and multiplied by 2  
 int value\_Of\_Even\_Position = Integer.*parseInt*(String.*valueOf*(num\_To\_String.charAt(even\_Positions - 1)));  
 int even = 2 \* value\_Of\_Even\_Position;  
  
 String evenum\_To\_String = even + "";  
  
 // Conditional statements to split double digits to single digits for even numbers  
 if (evenum\_To\_String.length() > 1) {  
 char firstValue = evenum\_To\_String.charAt(0);  
 char secondValue = evenum\_To\_String.charAt(1);  
 int first\_Digit = Integer.*parseInt*(String.*valueOf*(firstValue));  
 int second\_Digit = Integer.*parseInt*(String.*valueOf*(secondValue));  
 int DoubleEvenTotal = first\_Digit + second\_Digit;  
 totalEven += DoubleEvenTotal;  
  
 } else {  
 totalEven += even;  
 }  
 even\_Positions -= 1;  
 even\_Iteration++;  
 }  
  
 // Iteration to get the sum of odd position values  
 int odd\_Positions = num\_To\_String.length();  
 while (num\_To\_String.length() > odd\_Iteration) {  
 if (odd\_Positions % 2 != 0) {  
 int Odd = Integer.*parseInt*(String.*valueOf*(num\_To\_String.charAt(odd\_Positions - 1)));  
 total\_Odd += Odd;  
  
 }  
  
 odd\_Positions -= 1;  
 odd\_Iteration++;  
 }  
 // Sum of both odd and even computed and returned  
  
 break;  
 }  
 }  
 total = totalEven + total\_Odd;  
 return total;  
 }  
}