COMP125 Semester 2 2014
Assignment 1 – Credit Card Verification and Security

August 12, 2014

Overview of Assignment 1

This assignment will give you practice in designing and implementing some algorithms. All the algorithms requested relate to credit card verification and security. The assignment will be marked out of 24, and will contribute 6% towards your final mark for this unit. For this assignment a template program is provided to you, and an electronic submission is required from you, as described below. Your electronic submission will be machine tested and looked over by a tutor. Your mark for the assignment will be given on the basis of both the machine testing and the quality of your program as determined by visual inspection. Feedback on both kinds of marking will be provided to you. Submit your work in file `CardSecurity.java` via iLearn by 11 pm on Friday 5 September. Do **not** submit your entire project folder.

Description of basic tasks (for 20 marks out of 24)

Credit card companies typically use features such as *check digits* and *card security codes (CSCs)* for cards issued. A check digit depends on the card number, is usually added to the card number, and helps to validate the card number. Its purpose is related to detection of errors rather than security. A card security code (CSC), on the other hand, helps to verify that the person wishing to make a transaction actually possesses the card itself. CSCs are particularly useful for "card-not-present" transactions. (CSCs are usually not stored by merchants for security reasons.) This assignment explores the computation of both check digits and CSCs.

Check digits are often computed using relatively simple algorithms published quite freely. A still popular choice is a method devised by H. P. Luhn in 1954. We will use a variation of Luhn's technique. We describe our method as follows. Suppose that a card number is stored as a string of $n \geq 0$ decimal digit characters in *cardNum*. Starting from the left, we divide the card number into $\lfloor n/2 \rfloor$ pairs of digits, plus a single digit at the end if n is odd. (Note that $\lfloor n/2 \rfloor$ denotes the integer part of $n/2$.) Now we obtain a new card number *cardNumNew* with the same length by reversing the first, third, fifth, etc. pairs. The second, fourth, sixth, etc. pairs are not changed. If n is odd the single digit at the end is not changed. Next we add up the integer values of the $\lfloor n/2 \rfloor$ digit pairs of *cardNumNew*, adding to the sum

the value of the single digit at the end if n is odd, obtaining *sum*. Finally we obtain our check digit by computing the remainder of *sum* on division by 10.

For example, consider the 16-digit card number "3215697843892165". We obtain the new card number by reversing the first, third, fifth, etc. pairs: we get "2315967834891265". Next we find the sum of the integers 23, 15, 96, 78, 34, 89, 12 and 65, which is 412. Finally we compute 412 modulo 10, which is 2. This is the check digit.

CSCs, on the other hand, are computed using algorithms which tend to be confidential to the card issuing company. In practice the company uses relatively advanced cryptographic techniques in designing such algorithms, including the use of secret cryptographic keys. We will compute CSCs by adapting and simplifying an approach outlined by L. Padilla in relation to VISA PINs. Again, suppose that *cardNum* stores a string of $n \geq 0$ decimal digit characters. Suppose also that cryptographic keys k_1 and k_2 having positive integer values less than 10 are given. Our method involves the following steps.

1. First, encrypt *cardNum* by applying the rule $E(d) = (k_1 * d + k_2) \bmod 10$ to each digit of *cardNum*. In the rule, d denotes the integer value of a digit character of *cardNum*, and $E(d)$ denotes the integer value of the encrypted character. Store the result of the encryption in the string *cryptogram*. For example, suppose *cardNum* stores "123", $k_1 = 3$ and $k_2 = 5$. To encrypt the card number we encrypt each digit in turn. Observe that $E(1) = 8$, $E(2) = 1$ and $E(3) = 4$. Thus *cryptogram* should store "814".

2. Second, convert *cryptogram* into hexadecimal (that is, base 16) notation. Store the result of the conversion in the string *cryptoHex*. Note that *cryptoHex* could consist of a mixture of both decimal (0..9) and nondecimal (a..f) digits. (The nondecimal digits are defined by a = 10, b = 11, etc.) For example, suppose *cryptogram* stores "814". Then *cryptoHex* should store "32e". (The reason is that $814 = 3 \times 16^2 + 2 \times 16 + 14$.) Fortunately, there is a Java method which we could use to take care of the conversion into hexadecimal for us. We could use it if our card numbers have length at most 16 decimal digits.

3. Third, starting from the left, extract the first three decimal digits which occur in *hexString*, or as many decimal digits as possible if fewer than three are present. Store these digits in the string *code*. This is the CSC. For example, suppose that *cryptoHex* stores "32e". Then *code* should store "32".

As a larger example, consider again the 16-digit card number "3215697843892165". Put $k_1 = 3$ and $k_2 = 5$. Then *cryptogram* should store "4180326974921830", *cryptoHex* should store "ed9fc38629c66", and *code* should store "938".

You are required to develop a Java program containing methods to compute and display both a check digit and a CSC for a card number entered by the user. You are given a code template to start with. This is a zipped archive project folder assign1Template.zip available from iLearn. Download and import this file, as described in the Week 1 Workshop notes. This project contains a template file named CardSecurity.java which you could use as a starting point for your work. The `main` method defines some testing strings and calls some of the requested helping

methods. However the answers returned by these helping methods are incorrect. You need to complete the file correctly so that the check digit and CSC values computed are always correct for all given allowable input values.

In particular you are required to complete four Java methods, and to suitably modify the `main` method to allow the user to enter data. You could complete the methods in any order you like. The basic principle is: after a method is completed, test it thoroughly. We suggest you could start with the method specified below.

```
/**
* Checks whether or not the given string cardNum is a valid numeric string.
* It should return true if every character is a decimal digit
* and false otherwise.
*/

public static boolean isNumeric(String cardNum)
```

Design your function `isNumeric` according to the specification given above. Add some more lines to the `main` function to further test `isNumeric`. Once you are sure that `isNumeric` is working correctly, remove the testing statements from `main`.

Next try to write an *input validation loop* in your `main` function. Have `main` prompt the user to enter a card number comprising decimal digits. You could write a `while` loop which checks (using `isNumeric`) the string entered, and makes the user enter a new string if something containing a non-decimal-digit was entered. Test your program thoroughly.

The three remaining methods are specified as follows.

```
/**
* Computes and returns the check digit for the given card number.
* Precondition: it can be assumed that cardNum is a valid numeric string.
* Postcondition: the value returned is the check digit for cardNum
*                according to the procedure outlined in the assignment description.
*/

public static int checkDigit(String cardNum)


/**
* Returns the encryption of the given card number using the given keys.
* Precondition: it can be assumed that cardNum is a valid numeric string,
*               k1 is a positive integer at most 9, and
*               k2 is a positive integer at most 9.
* Postcondition: the string returned is the encryption of cardNum using keys
*                 k1 and k2, according to the procedure outlined in the
*                 assignment description.
*/
```

```
public static String encrypt(String cardNum, int k1, int k2)

/**
* Calculates CSC for given card number and given keys.
* Precondition: it can be assumed that cardNum is a valid numeric string
                whose length is at most 16,
*               k1 is a positive integer at most 9, and
*               k2 is a positive integer at most 9.
* Postcondition: the value returned is the CSC for cardNum, using
*                keys k1 and k2.
*/

public static String cardSecCode(String cardNum, int k1, int k2)
```

Design your method `checkDigit` according to the specification given above. You don't have to build any error-checking into your method `checkDigit`. Test your method thoroughly. Next design your method `encrypt` according to the specification given above. You don't have to build any error-checking into your method `encrypt`. Test your method thoroughly. Finally design your method `cardSecCode` according to the specification given above. You don't have to build any error-checking into your function `cardSecCode`. Test your method thoroughly.

Your final main method should prompt for and read data entered by the user, and should display the output of the three methods `checkDigit`, `encrypt` and `cardSecCode`. Submit your final class in electronic form (as file `CardSecurity.java`). Do not submit your entire CSC project folder. Be sure to document your code with your name, student ID and brief purpose of your program (as comments). You could include the block comments (that is, the /** */ kind) for each method given in the template file. These comments provide good documentation for your methods.

Only the four required methods `isNumeric`, `checkDigit`, `encrypt` and `cardSecCode` will be machine tested. But the marker who inspects your work will read all your work. The marker will be looking for clear, simple code, good programming style, and adherence to the specifications set out above.

Remaining points to be advised ...

The precise marking scheme will be announced in due course. A small but relatively advanced addition to the above basic tasks (for the remaining 4 marks out of 24) will also be advised in due course. Hints and pointers to handy Java methods which you could use will also be offered as needed.