CSS 735: HW 1

Due Date: *Sep 29 (F) 11:59 PM* ***Overdue submissions*****:** subject to penalties

Credit card numbers follow certain patterns. The number must have between 13 and 16 digits, and it must start with:

- 4 for Visa cards
- 5 for MasterCard credit cards
- 37 for American Express cards
- 6 for Discover cards

In 1954, Hans Luhn of IBM proposed an algorithm for validating credit card numbers. The algorithm is useful to determine whether a card number is entered correctly by a user or scanned correctly by a scanner. Credit card numbers are generated following this validity check, commonly known as the Luhn check or the Mod 10 check, which can be described as follows (for illustration, consider the credit card number 4388576018402626):

1. Double every second digit from right to left. If doubling of a digit results in a two-digit number, add up the two digits to get a single-digit number.

4	3	8	8	5	7	6	0	1	8	4	0	2	6	2	6
4*2=8		8*2=16 (1+6=7)		5*2=10 (1+0=1)		6*2=12 (1+2=3)		1*2=2		4*2=8		2*2=4		2*2=4	

2. Now add all single-digit numbers from step 1:

   4 + 4 + 8 + 2 + 3 + 1 + 7 + 8 = 37

3. Add all the digits in the odd places from right to left in the card number:

   6 + 6 + 0 + 8 + 0 + 7 + 8 + 3 = 38

4. Sum the results from Steps 2 and 3:

   37 + 38 = 75

5. If the result from Step 4 is divisible by 10, the card number is valid; otherwise, it is invalid. For example, the number 4388576018402626 is invalid, but the number 4388576018410707 is valid.

Write a Scala program that checks whether or not a number is a valid credit card number. The program reads input from an input file named **numbers.txt** and displays the output to standard output (the computer's screen). Each line in the file contains a single number. The output must be either the string valid or the string invalid with no extra leading or trailing spaces (this is important in case a script is used to check your output). Sample input and the corresponding output are as follows:

4388576018402626	invalid
4388576018410707	valid
4012888888881881	valid
4552720412345677	valid
4539992043491562	valid
4992739871600	valid
4992739870017	invalid
80840123456789	invalid
5588320123456789	invalid
5491946915444923	invalid
5490123456789128	valid
378282246310005	valid
371449635398431	valid
371449635398431	valid
378734493671000	valid
378734493671001	invalid
6041273990139424	invalid
6011111111111117	valid
6011000990139424	valid

Note: In order to compile your program, you need to provide the path to the input file unless it is in the same directory as the program.

Grading Scheme:

- **Working program [50%]** A working program which satisfies all of the requirements automatically receives 50% of the total assignment mark. Each element of non-compliance will be penalized with respect to its severity.
- **Program Structure [25%]** A program which follows principles of Object-Oriented Design and structured programming rules (procedural, modular, uses parameters) to perfection automatically receives 25% of the total assignment mark. Marks are deducted depending on severity and number of occurrences of non-compliant elements.
- **Program Documentation [20%]**
    - **Internal documentation [20%]:** Documentation should be complete and in a standard format. Every non trivial part of the code should have a *clear* comment that explains it. In addition, every method or function, including the main program should have an explicative comment header. This header includes:

module name, author, date of creation and purpose. A description of parameters and method output is mandatory. Marks are deducted according to the absence of these elements.

- **Program Style [5%]** Style refers to Occam's razor principle. Code that is needlessly tricky, obscure, or difficult to read will be marked accordingly. Program text indentation is also an element of style and must be present. Significant constant, variable and structure names must be used. Marks are deducted on the basis of the frequency of these errors.
- Use comments to document and explain your code where needed. Make sure to use functions to break the code into segments.
- Test each program with at least two different sets of input values.

Policies and procedures

- Assignments must be completed **individually and independently,** and **NOT** in a group.
- All files to be submitted should be placed in a single directory and **zipped** together into a single file identified by your name and student ID (**YourFamiliyName_StudentID.zip**) for uploading to Blackboard.
- Write your name and your student ID at the top of each file.
- Your **submission** for each programming question includes: (a) the source code files (.scala); (b) a Readme.txt file containing a brief explanation of each file; (c) To prove that your code works correctly, you will also submit screenshot of the output of your program in a file with an extension of **.png** or **.pdf**.
- Assignments are to be handed in by the **due date**; otherwise, **penalty per day** will be incurred: 1 day late 10% penalty; 2 days late 20% penalty; after 2 days no credit will be given for a late assignment. Homework that is habitually late will not be accepted.
- Any incomplete work that is submitted will be considered for partial marks.
- If exceptional circumstances have occurred that have kept you from submitting your assignment on time, you should contact the course instructor as soon as possible. If you are ill, a valid **document**, e.g., a doctor note, must be provided.