ASSIGNMENT 1

COMP-202, Summer 2017, Section 001

Due: May **14st**, 2017 (23:59)

Please read the entire PDF before starting. You must do this assignment individually.

Question 1:	50 points
Question 2:	50 points
	100 points total

It is very important that you follow the directions as closely as possible. The directions, while perhaps tedious, are designed to make it as easy as possible for the TAs to mark the assignments by letting them run your assignment, in some cases through automated tests. While these tests will never be used to determine your entire grade, they speed up the process significantly, which allows the TAs to provide better feedback and not waste time on administrative details. Plus, if the TA is in a good mood while he or she is grading, then that increases the chance of them giving out partial marks. :)

To get full marks, you must:

- Follow all directions below
- Make sure that your code compiles
  - Non-compiling code will receive a very low mark
- Write your name and student number in a comment in each .java file you hand in
- Indent your code properly
- Name your variables appropriately
  - The purpose of each variable should be obvious from the name
- Comment your work
  - A comment every line is not needed, but there should be enough comments to fully understand your program

Part 1 (0 points): Warm-up

Do **NOT** *submit this part, as it will not be graded. However, doing these exercises might help you to do the second part of the assignment, which will be graded. If you have difficulties with the questions of Part 1, then we suggest that you consult the TAs during their office hours; they can help you and work with you through the warm-up questions. You are responsible for knowing all of the material in these questions.*

Warm-up Question 1 (0 points)

Create a file called `HelloWorld.java`, and in this file, declare a class called `HelloWorld`. This class should define only one method called `main()`. In the body of this method, use `System.out.println()` to display "`Hello world!`". You can find such a class in the lecture slides; make sure you can compile and run it properly.

Warm-up Question 2 (0 points)

Create a file called `Diagram.java`, and in this file, declare a class called `Diagram`. This class should define only one method called `main()`. In the body of this method, use five statements of `System.out.println()` to display the following pattern:

```
  22
2   2
   2
 2
22222
```

Warm-up Question 3 (0 points)

1. What does the following logical expression evaluate to?

   ```
   (false or false) and (true and (not false))
   ```

2. Let a and b be boolean variables. Is it possible to set values for a and b to have the following expression evaluate as *false*?

   ```
   b or (((not a) or (not a)) or (a or (not b)))
   ```

Warm-up Question 4 (0 points)

The following code is designed to put the value of the variable x into the variable y and the value of the variable y into x. What is wrong with it?

```
public class BadSwap {
     public static void main(String[] args) {
          int x = 1;
          int y = 2;
          x = y;
          y = x;
     }
}
```

Part 2

The questions in this part of the assignment will be graded.

Question 1: Encryption Program (50 points)

Have you checked the list of the most frequently-used passwords on internet? Then, you will not be surprised at the fact that 'password' and 'qwerty' are at the top. You, as a student of Comp202 would never use something so indiscreet (right?). But internet passwords are not the only passcodes that give access to very sensitive information. For example, the PIN number of your debit card gives you access to your bank account. This number is composed by just four digits. Then, there are only 10000 possible combinations to decipher your pin code (making a lot easier for a thief to guess at it). Even if there are several possible combinations, according to cyber-security analysis, 27% of debit card clients use the top 20 most common PIN numbers for ATM cards. More surprisingly (or depressingly) is the fact that the code '1234' is used by almost 11% of card holders. Your birthday should also be avoided as PIN because nearly all users carry documentation of it on their wallet. Statistical studies have shown that a competent thief is able to gain use of an ATM card once for every 11-18 stolen wallets. Users try to use memorable passwords because they will not forget about it. However, it makes the work of thieves easier. As a Comp202 student, you have been commissioned to write a java program which takes a memorable PIN number and encrypts it according to a set of instruction provided to you. The number produced at the end of the encryption process is the one that you will use as your new ATM PIN.

INPUT:
The input to your program is a set of four integer numbers that represent the ATM PIN access code to encrypt.

ENCRYPTION:
The list of encryption steps is as follows:

1. Take the input number and reverse it.
2. If the reversed number (i.e., the number produced by step 1.) is greater or equal than 4987, then subtract the sum of its digits. Otherwise, add the sum of its digits.
3. If the number produced by step 2 is even, then add the biggest digit. Otherwise, subtract the smallest digit.

OUTPUT:
For your input ATM PIN, encrypt the number with the instructions given, and output each of the encrypted steps performed on your number.

Lets see an example to make clear what your program must output:

SAMPLE INPUT 1:
1234
SAMPLE OUTPUT 1:
Step 1: 4321
Step 2: 4331
Step 3: 4330

NOTES (for INPUT 1):
i) The Step 1 produces as result '4321' because this number represents the reverse of '1234'.
ii) Given that '4321' is less than '4987', Step 2 should add the sum of the digits of '4321'. In particular, the sum of its digits is equal to 10 (4+3+2+1=10). Then, the result produced by Step 2 is '4331' (4321 + 10 = 4331).

iii) Given that '4331' is not even, then, we must subtract to it the smallest digit. In this case, the smallest digit is 1 and the output of Step 3 is '4330' (4331 - 1 = 4330).

Lets see another example:
SAMPLE INPUT 2:
0987
SAMPLE OUTPUT 2:
Step 1: 7890
Step 2: 7866
Step 3: 7874

NOTES (for INPUT 2):
i) The Step 1 produces as result '7890' because this number represents the reverse of '0987'.
ii) Given that '7890' is greater or equal than '4987', Step 2 should subtract the sum of the digits of '7890'. In particular, the sum of its digits is equal to 24 (7+8+9+0=24). Then, the result produced by Step 2 is '7866' (7890 - 24 = 7866).
iii) Given that '7866' is even, then, we must add to it the biggest digit. In this case, the biggest digit is 8 and the output of Step 3 is '7874' (7866 + 8 = 7874).

OK, just lets see the last example:
SAMPLE INPUT 3:
4890
SAMPLE OUTPUT 3:
Step 1: 0984
Step 2: 1005
Step 3: 1005

NOTES (for INPUT 3):
i) The Step 1 produces as result '0984' because this number represents the reverse of '4890'. Note that it is important to print the leading digit '0' (you still need four digits to enter in the ATM machine)
ii) Given that '0984' (nine hundred eighty four) is smaller than '4987', Step 2 should add the sum of the digits of '0984'. In particular, the sum of its digits is equal to 21 (0+9+8+4=21). Then, the result produced by Step 2 is '1005' (0984 + 21 = 1005).
iii) Given that '1005' is not even, then, we must subtract to it the smallest digit. In this case, the smallest digit is 0 and the output of Step 3 is '1005' (1005 - 0 = 1005).

GENERAL NOTES:
i) Attached to this assignment is a file called `Encryption.java`. Note that for each encryption step, there is a marked section where your code must go. The code outside of this area must not be modified (that includes the name of the package). You should create a package in Eclipse with the same name (i.e., Assignment1). ii) Note that this program is run by providing *input arguments*. If you are using Eclipse, please read this:
`http://www.cs.colostate.edu/helpdocs/eclipseCommLineArgs.html` and/or go to the first tutorial.

Question 2: Grade Calculator (50 points)

Based on University regulations, the official grade in each McGill course is a letter grade. Grades A through C represent satisfactory passes, D a conditional (non-continuation) pass, and F a failure. You must obtain a grade of C or better in courses that you take to fulfil program requirements. Table 1 shows the possible Letter Grades and the Numerical Scale of Grades.

Table 1: McGill Grade Regulations

Text Grades	Numerical Scale
A	80 - 100%
B	65 - 79%
C	55 - 64%
D	50 - 54%
F	0 - 49%

As a Comp202 student your grade for the course will be computed based on the scores of your assignments, midterm and final examinations. In particular, you will automatically get the better mark of two possible schemas. Those two scenarios are shown in Table 2 and 3.

Table 2: Comp202 Marking Scheme 1

Task	Worth Value
Assignments	40%
Midterm	20%
Final	40%

Table 3: Comp202 Marking Scheme 2

Task	Worth Value
Assignments	40%
Final	60%

As a Comp202 student, you have been commissioned to write a java program to determine what grade (as numerical scale) you need on your final exam in order to get a certain letter grade in Comp202.

INPUT:
The input to your program is a set of two integer numbers that represent (i) the percentage value obtained by your assignments [an integer value between 0 and 100]. (ii) the percentage value achieved in your midterm [an integer value between 0 and 100]. Additionally, you will get as input a character that represents the desired text grade.

OUTPUT:
For the specific set of inputs, you must compute the minimum percentage value [an integer value between 0 and 100] needed to achieve the desired text grade.

Lets see an example to make clear what your program must output:

SAMPLE INPUT 1:
60 100 A
SAMPLE OUTPUT 1:
Final: 90

NOTES (for INPUT 1):
i) For the given input (i.e. Assignments = 60%, Midterm = 100%, TextGrade = 'A'), the minimum

score needed in the Final exam is computed by the schema 1 and it is equal to 90. You can verify this result by computing ($60 \times 0.4 + 100 \times 0.2 + 90 \times 0.4 = 80$ [where 80 is the minimum score to obtain an 'A', and 0.2, 0.4 and 0.4 correspond to the weights given in Table 2]).

Lets see another example:
SAMPLE INPUT 2:
60 90 A
SAMPLE OUTPUT 2:
Final: 94

NOTES (for INPUT 2):
i) For the given input (i.e. Assignments = 60%, Midterm = 90%, TextGrade = 'A'), the minimum score needed in the Final exam is computed by the schema 2 and it is equal to 94. You can verify this result by computing ($60 \times 0.4 + 94 \times 0.6 = 80.4$ [where 80.4 is greater than 80 [which is the minimum score to obtain an 'A'], and 0.4 and 0.6 correspond to the weights given in Table 3]). The exact score needed to obtain an A is 93.3, but given that your answer must be an integer, the right answer to display is 94. You should also check that 93 is not a correct answer because with that score you will not get an 'A' ($60 \times 0.4 + 93 \times 0.6 = 79.8$ [where 79.8 is NOT greater than 80 [which is the minimum score to obtain an 'A']).

OK, just lets see the last example:
SAMPLE INPUT 3:
80 100 D
SAMPLE OUTPUT 3:
Final: 0

NOTES (for INPUT 3):
i) For the given input (i.e. Assignments = 80%, Midterm = 100%, TextGrade = 'D'), the minimum score needed in the Final exam is computed by the schema 1 and it is equal to 0. You can verify this result by computing ($80 \times 0.4 + 100 \times 0.2 + 0 \times 0.4 = 52$ [where 52 is greater than 50 [which is the minimum score to obtain an 'D'], and 0.4, 0.2 and 0.6 correspond to the weights given in Table 2]). The exact score needed to obtain an A is -5, but given that your answer must be an integer value between 0 and 100, the right answer to display is 0 (you can not get a negative value in your final exam, the instructor is not so strict).
ii) In a similar way, if you need more than 100 to get a desired text grade, you should output as result 100 (the instructor is not strict, but he will not give you marks either).

GENERAL NOTES:
i) Attached to this assignment is a file called `Grade.java`. Note that in this file there is a marked section where your code must go. The code outside of this area must not be modified (that includes the name of the package).

What To Submit

You have to submit the following files. Please DO NOT zip (or rar) your files, and do not submit any other files (except from the ones listed below) especially .class files.

`Encryption.java`
`Grade.java`
`Confession.txt` (optional) In this file, you can tell the TA about any issues you ran into doing this assignment. If you point out an error that you know occurs in your problem, it may lead

the TA to give you more partial credit. On the other hand, it also may lead the TA to notice something that otherwise they would not.