CSC 110: Fundamentals of Programming I
Assignment #4: If-statements, loops

Due date

Friday, October 21st, 2016 at 5:00 pm via submission to conneX.

How to hand in your work

Submit the requested files for part (b) (see below) through the Assignment #4 link on the CSC 110 conneX site. Please make sure you follow all the required steps for submission (including confirming your submission). Part (a) is not marked.

Learning outcomes

When you have completed this assignment, you should understand:

- How to use if-statements to describe the control flow we want in our program.
- Using loops to process through strings.
- Use method parameters and method return values to direct the computation performed in your solution.

Part (a): Problems from the Textbook

Complete Chapter 4 self-check problems 1 to 17; complete Chapter 5 self-check problems 1 to 9 and 12; and compare your answers with those available at:

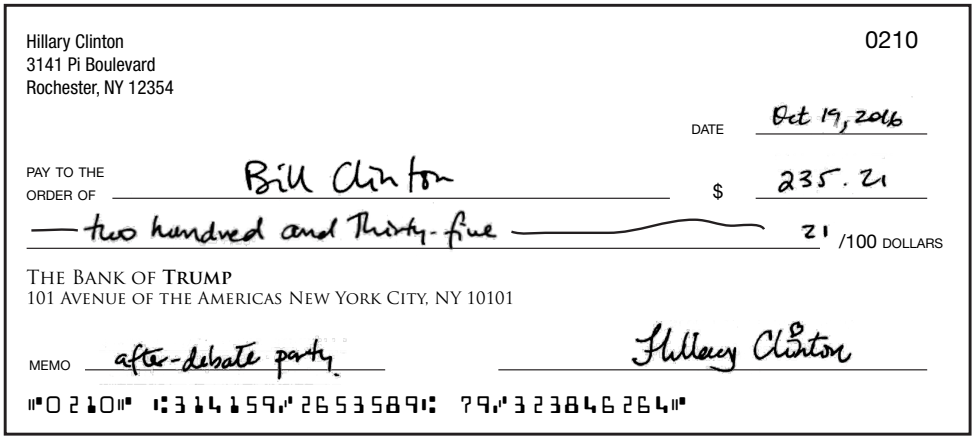
`http://www.buildingjavaprograms.com/self-check-solutions-4ed.html`

Part (b): Converting currency amounts into text

In this assignment you will write a program that converts a monetary amount up to $10000.00 and from its digit format to its spelled-out string format. The basic idea behind this algorithm lies behind any computer program that prints paper cheques (e.g., for paycheques, refunds from stores, government payments, etc.).

One reason cheques are written with amounts in both numerals and in text is to reduce some kinds of fraud. It is much harder to alter cheques by adding digits to the amount as both the numerals and the text must agree.

Here is an example cheque made out to the amount of $235.21.


Hillary Clinton
3141 Pi Boulevard
Rochester, NY 12354

0210

DATE Oct 19, 2016

PAY TO THE ORDER OF Bill Clinton $ 235.21

two hundred and Thirty-five 21 /100 DOLLARS

THE BANK OF TRUMP
101 AVENUE OF THE AMERICAS NEW YORK CITY, NY 10101

MEMO after-debate party Hillary Clinton

⑈0210⑈ ⑆314159⑆ 265358979⑈ 79⑆ 323846264⑈

The amount in numbers (235.21) matches the textual version of the amount ("two hundred and thirty-five ---- 21/100 dollars"). And here is an interaction example with your program—called *CurrencyToWords.java*—using the amount shown above:

```
$ java CurrencyToWords
What is the monetary amount? 235.21
two hundred and thirty-five ---------------------------- 21/100 dollars
```

The following must be hold for your solution to this problem.

- Your program must accept amounts as large as 10000.00 and as small as 0.01, and any other values in between. (When we test your submission we will never use fractional cents.)
- The output of your program must format the text version of the amount in a string 70 characters long. Note the dashes sit between the dollar amount and the cents amount with a space on either side of the dashed line. Notice that the "s" of dollars must be the 70th character of the string.
- The word *and* appears at most once—before the text for the part of the value that is less than 100. (Please look at the last page of this assignment description for some sample runs of the program.) Also notice the dash that is needed to hyphenate the text of numbers less than 100 (i.e., "thirty-five" but never "thirty five").

With respect to program structure, your solution must have at least three (3) methods:

- *public static String baseCardinalToString(int value):* Your whole program will be based upon this method as a building block. When the method is passed *value* belonging to the set of integers {0-9, 10-19, 20, 30, 40, 50, 60, 70, 80, 90}, then the method will use *value* to create the spelled-out text string version of

that number. All other input to the method is ignored. The method will return the created string. For example: if value=9 then it returns "nine"; if value = 18 it returns "eighteen"; if value = 30, it returns "thirty"; etc.

- *public static String convertToWords(double value):* This method uses the behavior of *baseCardinalToString* to build a string corresponding to the spelled-out version of *value*. Amongst other things, the method will need to separate the thousands from the hundreds, and then the hundreds from the values less than 100, and then those values from the cents (e.g., values less than 1.00). The methods statements will involve expressions using modulo arithmetic, multiplication, division and truncation.
- *public static double getValueFrom User():* This method interacts with the user at the console to obtain the monetary amount as a number. However, this method needs to be *robust*—that is, if the user does not enter a numeric amount, the program must re-prompt the user for the numeric amount. You can use the Scanner method *hasNextDouble()* to determine if the user has entered a suitable value (and then use *nextLine()* to discard the user's input if it isn't suitable). If a numeric amount has fractional cents, then the fractional cents can be ignored by your solution. Hint: You'll need a *while* loop to implement this method.

Your main method will therefore look something like:

```
double value = getValueFromUser();
String text = convertToWords(value);
System.out.println(text);
```

You are welcome to introduce more of your own methods as you think they are needed, but please do this wisely and carefully comment each such method (as you should for all methods declared in code you write).

File to submit: CurrencyToWords.java

A potential "gotcha"

Throughout the semester we have tried to emphasize the difference between the concept of a real number and its computer approximation (a variable of type *double*). This approximation can produce frustration when working with monetary amounts. For example, consider the following lines of Java:

```
double value    = 4398.44;
double value100 = value * 100.0;
int    cents    = ((int)(value * 100.0)) % 100;
System.out.println(value100);
System.out.println(cents);
```

These lines are an attempt to extract that part of *value* less than 1.00, and to store this part as an *int* (i.e., 0.44 becomes 44, 0.02 becomes 2, etc.). The second line

assigning to *value100* is not really needed but will help reveal a problem. Here is the output of the two *println* statements in that code:

```
439843.99999999994
43
```

Curiouser and curiouser! For some reason a Java *double* results in something odd when 4398.44 is multiplied by 100.0.

This kind of situation is one reason many commercial programs written in Java represent monetary amounts using the class *BigDecimal*. The approximations that occur when representing real numbers as doubles are never a factor with a *BigDecimal*. However, arithmetic with this class is not as straightforward as with doubles and therefore we will not use *BigDecimal*s in this assignment.

When we examine the output of the example code, it appears that the approximate value stored in *value100* is very, very close to what we need. The cast to *(int)* truncates the decimal portion; if we add 0.5 to the number before truncation we can get the value needed in *cents* (i.e., 44). Even if the approximation is in the other direction (e.g., 439844.000000000004), adding 0.5 and truncating via *(int)* will still result in 44 being stored in *cents*. The third line below is therefore modified accordingly:

```
double value    = 4398.44;
double value100 = value * 100.0;
int    cents    = ((int)(value * 100.0 + 0.5)) % 100;
System.out.println(value100);
System.out.println(cents);
```

and the output is:

```
439843.99999999994
44
```

Grading: What the marker will look for

- *Documentation*: Documentation in the implemented code must include the author's identification and the purpose of the program. Each group of instructions must be preceded with a comment describing their common purpose. Each method must be preceded with a comment describing its purpose as well as an indication of the input to and the output from the method.
- *White Space and Indentation* in the code and adherence to Java naming conventions.
- Your methods should *accept input parameter values and return an output* as described above. Your program should use the methods wherever it is appropriate. Other suitable methods can be created, of course, or no other methods, if that is appropriate.

- *Compiles and produces correct output:* The code constitutes a valid Java program (i.e. compiles *and* runs without error on the ECS 250 lab machines). It must accept input and print output as described above.
- Prompts are provided as described in this document.

Grading scheme

- "A" grade: An exceptional submission demonstrating creativity and initiative going above and beyond the assignment requirements. The program runs without any problems using the specified method. Any extra work appears in the file named *CurrencyToWordsExtra.java*, and identified within the file (i.e., Class comment) is how you have extended the assignment to demonstrate creativity and initiative.

- "B" grade: A submission completing the requirements of the assignment. The program runs without any problems, uses the specified methods, and interacts with the user as requested by this assignment.

- "C" grade: A submission completing most of the requirements of the assignment. The program runs with some problems but has the specified method, and yet might not have the expected output.

- "D" grade: A serious attempt at completing requirements for the assignment. The program runs with major problems, or is missing the specified method.

- "F" grade: Either no submission given, the submission does not compile, or submission represents very little work.

Appendix

```
$ java CurrencyToWords
What is the monetary amount? 305.00
three hundred and five ---------------------------------------- 0/100

$ java CurrencyToWords
What is the monetary amount? 9813.51
nine thousand eight hundred and thirteen ---------------------- 51/100

$ java CurrencyToWords
What is the monetary amount? 0.20
zero ---------------------------------------------------------- 20/100

$ java CurrencyToWords
What is the monetary amount? 8888.88
eight thousand eight hundred and eighty-eight ----------------- 88/100

$ java CurrencyToWords
What is the monetary amount? 1212.12
one thousand two hundred and twelve --------------------------- 12/100

$ java CurrencyToWords
What is the monetary amount? 3000.00
three thousand ------------------------------------------------ 0/100
```