Lab 12: Recursion
CSE-021-01, Spr 2018 **Points: 20**

Overview
You will learn the basics of using recursion in methods to solve typical problems in CSE elegantly.

Before you get started, read chapters 12.1 – 12.6. Answer activities 12.2.2, 12.3.3, 12.4.1, 12.5.2 and 12.6.2 in a Word or text file. Also, answer the Assessment questions as you encounter them in the next section. The prompts for answering assessment questions are placed immediately following the material to which the listed questions relate.

Getting Started
After following the import instructions in the assignment page, you should have a Java project in Eclipse titled Lab 21_12. This PDF document is included in the project in the **doc** directory. The Java files you will use in this lab are in the **src/recursion** directory.

Part 1: Introduction to Recursion
We will learn about recursion by stepping through the code that we have supplied for you. Then you will fill in code to complete the behavior of other recursion programs.

Task 1: Modify `SimpleRecursion.java`
There are 6 different versions of `recur` in `SimpleRecursion`. Your first task is to run them and figure out how each of them behave.

[Answer assessment questions 1a – 1f]

Modify `SimpleRecursion.java` so all 6 versions can be run inside `main`. Also make sure that it is obvious which version is running by differentiating the outputs.

Part 2: Recursion Using Helper Methods
We are now going to design a recursive version of a method called `reverse` that returns a copy of the argument String with its characters reversed. Here are some examples:

Call to `reverse`	Returned value
`reverse("ABC")`	`"CBA"`
`reverse("120ESC")`	`"CSE021"`

One can easily write `reverse` iteratively, with a loop instead of recursion. In many problems, however, this is not so simple. Our intent here is to help you add another tool to your toolbox, so that when a problem shows up that requires recursion, you'll be ready.

We first will explore writing *separate* methods for different string length – `reverse1`, `reverse2`, `reverse3`, and so on. For some programmers, this is a useful intermediate design step towards the goal of coding a general recursive `reverse`.

Task 2: Fill-in `NaiveReverse.java`
Fill in the code inside `reverse6` method.

[Answer assessment question 2]

A fellow student thinks that using a helper method will make it easier to write `reverse9`, a method to reverse Strings of length 9. Your instructor suggests using `reverse8` as the helper (you can assume that `reverse8` works correctly; that is, it reverses a String of length 8 correctly). The student comes up with the following solution:

```
public String reverse9 (String s) {
   String partial = reverse8(s.substring(1, 9));
   String firstLetter = s.substring(0, 1);
   return (firstLetter + partial);
}
```

[Answer assessment questions 3a – 3c]

Task 3: Fill-in `Reverse17.java`
Fill in the code inside `reverse17` method.

Task 4: Fill-in `Reverse.java`
The usual way to write recursion is using a base case to return a default value, and using recursion otherwise (Zyante 12.1 and 12.3). Fill in the code inside `reverse` method to check for a base case and recursion case while returning the proper values or expressions.

Part 3: Working With Numbers

Task 5: Fill-in `Halved.java`
Fill in `Halved.java` so it uses recursion to return the result of a given number halved. It does not use any division or multiplication, but uses subtraction instead. The method ignores the fractional parts of halved numbers. So, half of `10` is `5` but it is also the same answer for `11`. Your task is to figure out the base case where the half of a small number is `0` (returning `0`). Figure out the range of small numbers where this is true and place your answer into the condition check of the `if`-statement.

Task 6: Fix `DigitCount.java`
The `digitCount` method is intended to return the number of digits in its (non-negative) integer argument. For example, with `314159` or `700000` as argument, the method should return `6`. Leading zeroes are not counted except in the case where the argument is `0`. The code for the method is as follows:

```
public static int digitCount(int value) {
   if(value == 0) {
      return 1;
   } else {
      return 1 + digitCount(value/10);
   }
}
```

[Answer assessment questions 4a – 4e]

Fix `digitCount` with the answers to the questions so it works properly.

Part 4: (Assessment) Logic Check and Level of Understanding

1) For the `recur` method in the `SimpleRecursion` class, answer the following:
   a. What is the output of version 1 called with argument `0`.
   b. What is the output of version 2 called with argument `0`.
   c. What is the output of version 3 called with argument `0`.
   d. What is the output of version 4 called with argument `0`.
   e. What is the output of version 5 called with argument `0`.

   f. What is the output of version 6 called with argument `0`. How can we run all 6 versions at once? (hint: name overloading does not work)

2) After filling-in the `reverse6` method in the `NaiveReverse` class, can you think of how to extend this algorithm for `reverse7` method? Explain.
3) Answer the following questions concerning the `reverse9` method given in Task 2:
   a. A student tests the `reverse9` method on the String `"confident"`. What will it return?
   b. What should be the proper reversed string for `"confident"`?
   c. Suggest how to fix the code so the method does it correctly.
4) Answer the following questions concerning the `digitCount` method in the `DigitCount` class:
   a. What is returned from the call `digitCount(0)`?
   b. What is returned from the call `digitCount(10)`?
   c. What is returned from the call `digitCount(314159)`?
   d. Should the digit count increase if the `value` is `0`? If not how should the code change
   e. What are other values of `value` for which digit count should not increase? Change the code to include those values.

What to hand in

When you are done with this lab assignment, submit all your work through CatCourses.

Before you submit, make sure you have done the following:

- Verified your solution with your TA or instructor
- Included answers to activities 12.2.2, 12.3.3, 12.4.1, 12.5.2 and 12.6.2 and Assessment questions (1 – 4) in a Word or text file named `Part4`
- Attached modified `SimpleRecursion.java`, filled in `NaiveReverse.java`, filled in `Reverse17.java`, filled in `Reverse.java`, filled in `Halved.java`, fixed `DigitCount.java`, and `Part4` files.
- Filled in your collaborator's name (if any) in the "Comments…" text-box at the submission page