COMP 196ABL – Project #9
10 points with option of 8 extra credit points; due date - end of lab session you attend on April 15, 2019
Email your Java source file to: sgs@csun.edu

Recursive Change Maker

At many grocery and convenience stores a change machine is connected to the cash register. After the amount of cash tendered is entered, the cash register sends control signals to the change machine to vend an appropriate number of quarters, dimes, nickels and pennies. For purposes of this assignment, the control signals will be represented by the printing of the values of the coins to be released. So, for 17 cents in change, the output generated would be:

```
10
5
1
1
```

representing one dime, one nickel and two pennies. At the end of each transaction, generate a blank line.

Write code to complete the following change maker program.

```
public class Change
{
   public static void main(String[] args)
   {
      change(75);  // test cases
      change(42);
      change(1);
      change(5);
      change(93);
      change(17);
   }

// insert your method definition(s) here

}
```

For this assignment, you may NOT use division (/) or remainder (%) operators, nor any library methods that perform similar operations. Also, NO loop constructs other than recursion are permitted. You can add as many methods as you like to the above code base.

(continues on next page)

The output generated by the sample test cases in the main:

```
25
25
25

25
10
5
1
1

1

5

25
25
25
10
5
1
1
1

10
5
1
1
```

EXTRA CREDIT OPTIONS (you can implement either or both of these options; they are not dependent on each other):

1. (4 extra credit points) Design a recursive solution that requires no more than 18 lines of well-structured code with all { } braces appearing as standalone lines. This count does not include blank lines, comment lines, or the base code appearing at the start of this assignment. Method header declarations as well as the lines containing { or } braces count towards the 18 line limit. The instructor holds the right to make the final determination what is or is not well-structured code.

2. (4 extra credit points) Create a well-structured solution that permits the changing of the coin denominations and the number of different coin types through changes to just one line of code. For example, if we wanted this same program to function correctly in terms of European Euros, instead of US coins, we would have the following coin denominations: 20, 10, 5, 2, and 1. In both monetary currencies we are dealing with coin denominations less than 50 cents.