PA 3
Worth: 40 points + possible 5 points extra credit!

Brief Introduction:

In this assignment you will perform number conversion between different bases. As discussed in lecture early in the semester, in our every day life we use base 10 to represent our numbers. This resulted from the fact that we have 2 hands and 5 fingers on each hand. In base 10, we use ten different symbols: the digits 0 through 9. Computers, on the other hand (pun intended), use binary or base 2 and only two symbols: 0 and 1. Because a long string of zeros and ones is hard to remember, we often use octal (base 8) or hexadecimal (base 16) to represent values stored in a computer since these representations make the numbers easier for us to remember. Octal uses the digits 0-7 while hexadecimal uses 0-9 plus the characters A (for 10) through F (for 15).

Your task on this assignment is to write a series of recursive methods that convert numbers from one base to another. To illustrate the process, consider the number 13 (in base 10) and suppose we wish to represent this in octal. The process involves repeating the following step:

Divide the number by the new base to obtain the quotient and the remainder. 13 divided by 8 gives a quotient of 1 and a remainder of 5. The 5 becomes our rightmost digit of the new number we are creating and we repeat the process with the quotient. 1 divided by 8 gives a quotient of 0 and a remainder of 1. This remainder becomes the next digit of our result. Because the quotient is 0, we have completed the conversion process. The result is 15 base 8. The following is a table of the number 13 converted into different bases. Before you continue reading this assignment, see if you can produce each of these results:

Original Number	New Base	Resulting number
13	2	1101
13	3	111
13	4	31
13	5	23
13	6	21
13	7	16
13	8	15
13	16	D

Problem statement:

Create the *DecimalToNewBase* class. It contains a single int state variable, number, and supports several instance behaviors as described below (you should determine if you need additional methods):

decToBaseTwo()
This recursive method returns the base 10 number converted into base 2 (binary). The “digits” used in this base are: 0, 1.

decToBaseThree()
This recursive method returns the base 10 number converted into base 3. The “digits” used in this base are: 0, 1, 2.

decToBaseFive()
This recursive method returns the base 10 number converted into base 5. The "digits" used in this base are: 0, 1, 2, 3, 4.

decToBaseEight()
This recursive method returns the base 10 number converted into base 8 (octal). The "digits" used in this base are: 0, 1, 2, 3, 4, 5, 6, 7.

decToBaseNine()
This recursive method returns the base 10 number converted into base 9. The "digits" used in this base are: 0, 1, 2, 3, 4, 5, 6, 7, 8.

decToBaseTwelve()
This recursive method returns the base 10 number converted into base 12. The "digits" used in this base are: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A for 10, B for 11.

decToBaseSixteen()
This recursive method returns the base 10 number converted into base 16 (hexadecimal). The "digits" used in this base are: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E.

Note: while the first five of these methods could return a numeric value as a result, base 12 numbers will include the digits 0-9 and the letters A (for 10) and B (for 11) and the base 16 numbers will include the digits 0-9 and the letters A (for 10) through F (for 15). As a result, these last two methods must return string results. So, program your methods so ALL return a string result.

Hint: you might find it useful to develop some "helper" methods that do all or some of the work.

Driver program:

Write a *NumberConversion* class. This driver program creates a base 10 number and converts this number to each of the bases given above. This method should produce output similar to the following:

```
Decimal 13 = 1101 base 2
Decimal 13 = 111 base 3
Decimal 13 = 23 base 5
Decimal 13 = 15 base 8
Decimal 13 = 14 base 9
Decimal 13 = 11 base 12
Decimal 13 = D base 16
```

Execute this driver program with 5, 57, 63, and 129.

Extra Credit! (5 points):

Develop a single, additional method for the *DecimalToNewBase* class called **DecToN(base *int*)** that could replace all of the other methods that you have written (note: this is an *additional* method. You must develop ***all*** of the other methods as well). **DecToN** will convert the decimal state variable, number, into a string representing number in any base between 2 and 36 inclusively (the upper limit is due to the alphabet having 26 characters).

Deliverables:

You are required to submit electronically a single ZIP file containing:

1. The Java source code for the classes used to solve this problem.
2. The implementation of your test plan.
3. An analysis and design of each of the methods developed.
4. A UML specification of all of the developed classes.
5. A description of your test plan. The test plan should consist of *an adequate number* of tests to show that your methods have been properly implemented (you need to determine what number is appropriate and justify it). Each test case should consist of a statement of the test case's objective, a data set for the test case, and the expected results from executing this test case.

Grading:

Analysis and design:

1. UML Specification _____/4
2. Analysis for each method _____/6
3. Design for each method _____/5

Testing and implementation:

4. Description of test plan _____/4
5. Test plan implemented _____/4
6. Java source code submitted and works _____/10
7. Code is commented and has proper structure (white space/layout) _____/4

Submitting all of the material as directed: _____/3

Extra Credit for implementing DecToN _____/5