**CSCI 470/680-E Assignment 2 – Classes, Objects, and Methods Summer 2016**

**50 points**

This programming assignment is to introduce you to object-oriented programming in Java. It is of medium difficulty but will hopefully solidify your knowledge about classes, objects, and methods before we move into much more complex object-oriented material.

**Part 1 – 25 points**

Create a USMoney class with two integer instance variables dollars and cents. Add a constructor with two parameters for initializing a USMoney object. The constructor should check that the cents value is between 0 and 99 and, if not, transfer some of the cents to the dollars variable to make it between 0 and 99.

Add a plus method to the class that takes a USMoney object as its parameter. It creates and returns a new USMoney object representing the sum of the object whose plus() method is being invoked and the parameter.

The plus method does not modify the values of the two existing objects. It should also ensure that the value of the cents instance variable of the new object is between 0 and 99.

For example, if x is a USMoney object with 5 dollars and 80 cents, and if y is a USMoney object with 1 dollar and 90 cents, then x.plus(y) will return a new USMoney object with 7 dollars and 70 cents.

Of course, you will need to create a USMoneyDemo class that tests the USMoney class. Your USMoneyDemo should prompt the user twice. The first to enter an integer representing dollars and the second to enter an integer representing cents. Use these values to create the first object named x. Do it again for the object named y. Do not worry about putting these prompts in a loop for now.

For Part 1, submit two files: USMoney.java and USMoneyDemo.java.

**Part 2 – 25 points**

Create a Date class with three integer instance variables named day, month, year. It has a constructor with three parameters for initializing the instance variables, and it has one method named daysSinceJan1().

Just as its identifier suggests, the daysSinceJan1() method computes and returns the number of days since January 1 of the same year, including January 1 and the day in the Date object itself.

Create a DateDemo class that tests the Date class. First prompt the user to enter an integer representing the day, a second prompt to enter an integer representing the month and a third to enter an integer representing the year. Do not worry about doing any validation of the values entered. Then, instantiate your Date object with these values. After that, call the method DaysSinceJan1() to display the answer.

An example would be that the user enters values pf day = 1, month = 3, and year = 2016, then the call day.daysSinceJan1() should return 61 since there are 61 days between the dates of January 1, 2016, and March 1, 2016. In non-leap years, it should return 60.

Note that, in a leap year like 2016, there will be one more day counted between January 1 and any date following February 28. What are the rules for leap years? If the year is evenly divisible by four, then it is a leap year...but not always! If it is a century year such as 1900 or 2000, it is only a leap year if evenly divisible by four. Therefore, the century year 2000 WAS a leap year but 1900 was NOT a leap year! Neither were 1700 or 1800 and neither will be 2100...but 1600 was!

For Part 2, submit two files: Date.java and DateDemo.java