CMPS 12A
Introduction to Programming
Winter 2013

Programming Assignment 3

In this assignment you will write a java program that prompts the user for two positive integers, then prints out the greatest common divisor of the two numbers. Your program will check its input, and will repeatedly prompt the user until appropriate values are entered. The main program control construct used in this program will be the loop (while, do-while, or for).

The *greatest common divisor* (GCD) of two positive integers is, as the name implies, the largest integer which evenly divides both numbers. For instance the GCD of 531 and 300 is 3. One way to see this is to list all divisors of both numbers, determine the set of common divisors, then determine the largest element in that set.

> Divisors of 531: {1, 3, 9, 59, 177, 531}
> Divisors of 300: {1, 2, 3, 4, 5, 6, 10, 12, 15, 20, 25, 30, 50, 60, 75, 100, 150, 300}
> Common Divisors of 531 and 300: {1, 3}

It follows that the GCD of 531 and 300 is 3. Another way to find the GCD is to simply compare the prime factorizations of the two numbers. In this case $531 = 3 \cdot 3 \cdot 59$ and $300 = 2 \cdot 2 \cdot 3 \cdot 5 \cdot 5$, so clearly 3 divides both numbers, while no number larger than 3 does. The problem with these techniques is that for large integers, it is a very time consuming and computationally intensive task to determine the prime factorization or the full set of divisors. Around 300 BC, Euclid of Alexandria published a very efficient algorithm for the determination of the GCD. This algorithm is known universally today as the Euclidean Algorithm. It uses the operation of integer division, which takes two positive integers as input, and produces two integers as output.

Recall that if *a* and *b* are positive integers, then the quotient *q* and remainder *r* of *a* upon division by *b* are uniquely determined by the two conditions: $a = b \cdot q + r$ and $0 \le r < b$. We call *a* the *dividend*, *b* the *divisor*, *q* the *quotient* and *r* the *remainder*. If $r = 0$ we say that *b divides evenly* into *a*, or simply that *b divides a*. (Note that the word "divisor" is used differently here than in the preceding paragraph, in which it meant a number which divides evenly into another. The remainder need not be zero in general.) For instance, dividing 531 by 300 yields a quotient of 1 and remainder of 231, since $531 = 300 \cdot 1 + 231$.

We now illustrate the Euclidean Algorithm on the two numbers 531 and 300.

- Divide the larger number by the smaller, getting a quotient and remainder: $531 = 300 \cdot 1 + 231$
- Divide the preceding divisor by the preceding remainder: $300 = 231 \cdot 1 + 69$
- Divide the preceding divisor by the preceding remainder: $231 = 69 \cdot 3 + 24$
- Divide the preceding divisor by the preceding remainder: $69 = 24 \cdot 2 + 21$
- Divide the preceding divisor by the preceding remainder: $24 = 21 \cdot 1 + 3$
- Divide the preceding divisor by the preceding remainder: $21 = 3 \cdot 7 + 0$

The process halts when we reach a remainder of 0. The GCD is then the last non-zero remainder, which is in this case is 3. Observe that on each iteration, we discard the dividend and quotient, and save only the divisor and remainder for the next step, on which they become dividend and divisor respectively. Note also that the process must eventually terminate, since on each iteration the remainder is less than the divisor ($0 \le r < b$), hence the sequence of remainders is strictly decreasing and must reach zero.

We illustrate on another example: find the GCD of 675 and 524. We begin by dividing the larger by the smaller.

- $675 = 524 \cdot 1 + 151$
- $524 = 151 \cdot 3 + 71$
- $151 = 71 \cdot 2 + 9$
- $71 = 9 \cdot 7 + 8$
- $9 = 8 \cdot 1 + 1$
- $8 = 1 \cdot 8 + 0$

We stop when we reach a remainder of 0, and conclude that the GCD of 675 and 524 is 1. (Check this result by working out and comparing the prime factorizations of 675 and 524.) If you'd like to see proof of the correctness of Euclid's Algorithm, take Discrete Mathematics CMPE 16. (I'm teaching it this Summer.)

In this project you are to write a Java program that implements the Euclidean Algorithm described above. It should be clear at this point that your program must use one of Java's iterative control structures (i.e. a while loop, do-while loop, or a for loop.) Recall that if *a* and *b* are *int* variables storing positive values, then the expression $a \% b$ evaluates to the integer remainder of *a* upon division by *b*. Recall also that the quotient is never used, so one need only declare *int* variables for the dividend, divisor, and remainder. On each iteration you should compute the remainder from the dividend and divisor, update the values of dividend and divisor, then go to the next iteration. The loop repetition condition should simply be that the remainder is greater than zero.

In addition, your program will carry out a robust interaction with the user to check for correct input. In particular, if the user enters anything other a positive integer, your program will continue to prompt for more input. You must design appropriate loops to carry out these checks. Format your program output to coincide with the sample runs below.

```
% java GCD
Enter a positive integer: 531
Enter another positive integer: 300
The GCD of 531 and 300 is 3

% java GCD
Enter a positive integer: 300
Enter another positive integer: 531
The GCD of 300 and 531 is 3

% java GCD
Enter a positive integer: -57
Please enter a positive integer: a;lsdkjf
Please enter a positive integer: 531
Enter another positive integer: qopweiru
Please enter a positive integer: 123.456
Please enter a positive integer: z.x,mvn
Please enter a positive integer: 300
The GCD of 531 and 300 is 3
```

Observe that the user may enter the numbers in any order, i.e. smaller—larger, or larger—smaller. Note also that the program prompts separately for the two inputs, and that non-positive integers, double values, and non-numeric strings are rejected until a positive integer is entered. It is recommended that you first

write a program that does no input checking of any kind, and just computes the GCD of two positive integers correctly. Only when this phase is complete, should you implement the input checks described above. See examples on the website illustrating use of the `break` and `continue` commands for hints on how to implement these checks.

What to turn in

Name your source file for this project `GCD.java` and submit it to the assignment name `pa3`.