Programming Exercise 02

Authors: Vinayak, Eric, David
Topics: conditionals, iteration, printf, switch statements

Problem Description

The greatest common divisor (GCD) of two integers a and b is the greatest integer d that evenly divides both a and b; that is, $a \% d = 0$ and $b \% d = 0$. The GCD of two integers can be found using the Euclidean algorithm: a process that begins by dividing the two integers and then iteratively dividing the dividend by the divisor until the remainder is 0. The dividend is updated to the value of the previous divisor and the divisor is updated to the value of the previous remainder. The divisor that resulted in a remainder of 0 is the GCD. The GCD of two integers can also tell you if the two integers are relatively prime. Two integers are relatively prime if they share no common factors; that is, if the GCD of the two integers is 1.

To demonstrate, let's apply the Euclidean algorithm to find the GCD of 20 and 12.

1. **Let $dividend = 20$ and let $divisor = 12$. We then find the quotient and the remainder.**
   $\text{quotient} = 20 \div 12 = 1$
   $\text{remainder} = 20 \% 12 = 8$
   Thus, 20 = 12 * 1 + 8
2. **In the previous step, the divisor was 12 and the remainder was 8. The dividend is updated to the divisor (12), and the divisor is updated to the remainder (8).**
   $\text{quotient} = 12 \div 8 = 1$
   $\text{remainder} = 12 \% 8 = 4$
   Thus, 12 = 8 * 1 + 4
3. **In the previous step, the divisor was 8 and the remainder was 4. The dividend is updated to the divisor (8), and the divisor is updated to the remainder (4).**
   $\text{quotient} = 8 \div 4 = 2$
   $\text{remainder} = 8 \% 4 = 0$
   Thus, 8 = 4 * 2 + 0

   The remainder is 0, so the divisor that resulted in that is the GCD of 20 and 12, which is 4. This means that 4 is the largest number that evenly divides 20 and 12 without a remainder. Since the GCD of 20 and 12 is not 1, this means that 20 and 12 are not relatively prime.

Notice how this algorithm repeats the same few steps until a particular condition is met. This means we can implement it using iteration (i.e., loops)!

Solution Description

The Euclidean Algorithm

1. Create a class called `EuclideanAlgorithm`.
2. Create the `main` method inside the `EuclideanAlgorithm` class.
3. Inside the `main` method, create the following variables:
    a. Declare and initialize an `int` named `num1` with any integer in the range [1, 1000] of your choice.
    b. Declare and initialize an `int` named `num2` with any integer in the range [1, 1000] of your choice.
    c. Declare an `int` named `steps` and initialize it a value of 0.
    d. Declare an `int` named `dividend` and initialize it to the value of `num1` (do not hardcode using the literal value).
    e. Declare an `int` named `divisor` and initialize it to the value of `num2` (do not hardcode using the literal value).
    f. Declare an `int` named `gcd`.
    g. Declare an `int` named `quotient`.
    h. Declare an `int` named `remainder`.
4. Using `printf` with appropriate format specifiers, print the following on its own line:

    ```
    Finding the greatest common divisor of {num1} and {num2}.
    ```

5. Using an `if-else` statement:
    a. If `divisor` is greater than `dividend`:
        i. Print the following on its own line:

        ```
        The inputs would have caused an unnecessary step.
        ```

        ii. Swap the values of `divisor` and `dividend`.
    b. Otherwise:
        i. Print: `An extra step was avoided.`
6. Within a `do-while` loop, perform the following steps while `remainder` is not equal to zero:
    a. Increment `steps`.
    b. Assign `dividend / divisor` to `quotient`.
    c. Assign the remainder of `dividend / divisor` to `remainder`.
    d. Using printf with appropriate format specifiers, print the following on its own line:

    ```
    Step {steps}: {dividend} = {divisor} * {quotient} + {remainder}
    ```

    e. Update `gcd` to the value of `divisor`.
    f. Update `dividend` to the value of `divisor`.
    g. Update `divisor` to the value of `remainder`.
7. Using string concatenation and one `println` statement, print out the following:

    ```
    The GCD is {gcd}.
    ```

8. Using a `switch` statement, print one of the following statements on its own line depending on the value of `steps`:
    a. 1: `Only one step was needed!`
    b. 2: `Two steps were taken!`
    c. 3: `This process took three steps.`
    d. 4: `Wow! Four steps.`
    e. In all other cases: `{steps} steps is a lot of steps!`
9. Using a ternary expression (`?:`) and `printf`, print one of the following statements on its own line depending on whether `num1` and `num2` are relatively prime:
    a. If `num1` and `num2` are relatively prime:
        i. `{num1} and {num2} are relatively prime.`
    b. Otherwise:
        i. `{num1} and {num2} are not relatively prime.`
    - **Hint:** See the Problem Description section above for how to determine if two integers are relatively prime.

Turn-In Procedure

Submission

To submit, upload the files listed below to the corresponding assignment on Gradescope:

- `EuclideanAlgorithm.java`

Make sure you see the message stating the assignment was submitted successfully.

You can submit as many times as you want before the deadline, so feel free to resubmit as you make substantial progress on the assignment. We will only grade your latest submission. **Be sure to submit every file each time you resubmit**.

Gradescope Autograder

If an autograder is enabled for this assignment, you may be able to see the results of a few basic test cases on your code. Typically, tests will correspond to a rubric item, and the score returned represents the performance of your code on those rubric items only. If you fail a test, you can look at the output to determine what went wrong and resubmit once you have fixed the issue.

The Gradescope tests serve two main purposes:

- Prevent upload mistakes (e.g., non-compiling code)
- Provide basic formatting and usage validation

In other words, the test cases on Gradescope are by no means comprehensive.

Be sure to thoroughly test your code by considering edge cases and writing your own test files. You also should avoid using Gradescope to compile, run, or Checkstyle your code; you can do that locally on your machine.

Other portions of your assignment can also be graded by a TA once the submission deadline has passed, so the output on Gradescope may not necessarily reflect your grade for the assignment.

Burden of Testing

You are responsible for thoroughly testing your submission against the written requirements to ensure you have fulfilled the requirements of this assignment.

Be **very careful** to note the way in which text output is formatted and spelled. Minor discrepancies could result in failed autograder cases.

If you have questions about the requirements given, reach out to a TA or Professor via the class forum for clarification.

Allowed Imports

To prevent trivialization of the assignment, you may not import any classes for this assignment.

Feature Restrictions

There are a few features and methods in Java that overly simplify the concepts we are trying to teach or break our autograder. For that reason, do not use any of the following in your final submission:

- `var` (the reserved keyword)
- `System.exit`
- `System.arraycopy`

Collaboration

Only discussion of the assignment at a conceptual high level is allowed. You can discuss course concepts and assignments broadly; that is, at a conceptual level to increase your understanding. If you find yourself dropping to a level where specific Java code is being discussed, that is going too far. Those discussions should be reserved for the instructor and TAs. To be clear, you should never exchange code related to an assignment with anyone other than the instructor and TAs.

Important Notes (Don't Skip)

- Non-compiling files will receive a 0 for all associated rubric items.
- Do not submit `.class` files.
- Test your code in addition to the basic checks on Gradescope.
- Submit every file each time you resubmit.
- Read the "Allowed Imports" and "Restricted Features" to avoid losing points.
- **Check on Ed Discussion for a note containing all official clarifications and sample outputs.**

It is expected that everyone will follow the Student-Faculty Expectations document and the Student Code of Conduct. The professor expects a **positive, respectful, and engaged academic environment** inside the classroom, outside the classroom, in all electronic communications, on all file submissions, and on any document submitted throughout the duration of the course. **No inappropriate language is to be used, and any assignment deemed by the professor to contain inappropriate offensive language or threats will get a zero.** You are to use professionalism in your work. Violations of this conduct policy will be turned over to the Office of Student Integrity for misconduct.