Homework 1

EE 363 (Fall 2022)

Department of Electrical and Computer Engineering
Clarkson University

Instructions

Note 1: Please read all instructions carefully before submitting your work. There is 1 question in this homework for a total of **40** points.

Note 2: Solve all problems and upload your answers to Moodle using the instructions below. Make sure any code you write works on Polaris before uploading your files. You will likely lose many points if your code doesn't compile. Whenever you write solutions on paper, you need to scan all documents before submitting.

Note 3: `USR` stands for your login ID on Polaris (`polaris.clarkson.edu`). `MJR` represents your major, i.e. CE, SE, EE, CS, etc.

Note 4: In program listings and shell interactions, any **`bold, underlined, constant-width`** text indicates that it is user input.

Note 5: In program listings and shell interactions, any *`italicized, underlined, constant-width`* text has to be explicitly replaced by an appropriate user-supplied value.

Note 6: Do not upload any executable or intermediate files as answers to problems, unless specifically asked to do so.

Note 7: You should upload a single file named `USR_HW1_MJR.zip` to Moodle. (If you have multiple majors, please mention all of them in the file name, e.g. `USR_HW1_MJR1_MJR2.zip`.) In this ZIP file, you should have a directory named `hw1`, inside which you should have directories for each problem, named to clearly indicate the problem number, i.e. `p1`, `p2`, etc, each of which should contain the appropriate deliverable. (To verify the directory structure before creating the zip file, the `tree` command on Polaris can be used to display the directory contents in a hierarchical fashion, e.g. `tree dirName`.)

Note 8: Please read the "Note on academic honesty" in the syllabus. In the directory for *each* problem, please include a `readme.txt` file containing the following:

- It should state whether you worked alone on that probem or list the full names of all the people you discussed it with. This includes the names of the course staff, if any of them helped you with the problem.
- If your answer includes or is inpired by a non-trivial amount of content (including code) from specific sources, please cite all of them prominently.

Note 9: On Polaris, the following command can be used to zip the contents of directory `hw1` into a file: `dest.zip`: `zip -9 -yrq dest.zip hw1`

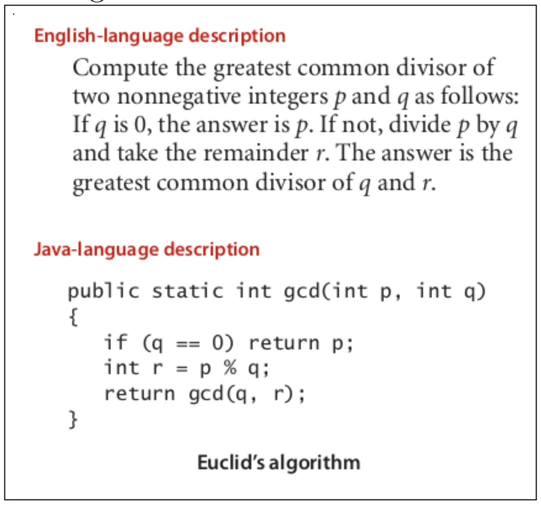
Note that you will need to rename the ZIP file as per the instructions described earlier. Please test the generated zip file by listing its contents (`unzip -l dest.zip`) and also by extracting and viewing its contents in a separate (temporary) directory (`unzip dest.zip`).

1. [40 points] You need to design the immutable `Rational` type with overflow validation having the interface shown in Listing 1. The class should be in package `hw1.p1`. Sample interactions are shown in Figure 3 and Figure 2.

   You should always keep a `Rational` in its lowest form. For this, a helper function to compute the GCD of two numbers might be useful Figure 1.

   In Figure 3, the lines enclosed in square brackets show the part of the source code required for understanding the output.

Figure 1: The greatest common divisor of two numbers.



English-language description

Compute the greatest common divisor of two nonnegative integers *p* and *q* as follows: If *q* is 0, the answer is *p*. If not, divide *p* by *q* and take the remainder *r*. The answer is the greatest common divisor of *q* and *r*.

Java-language description

```
public static int gcd(int p, int q)
{
   if (q == 0) return p;
   int r = p % q;
   return gcd(q, r);
}
```

Euclid's algorithm

Listing 1: API of the immutable Rational class.

```
 Rational(int numerator, int denominator)
Rational plus(Rational b)      //sum of this and b
Rational minus(Rational b)     //difference of this and b
Rational times(Rational b)     //product of this and b
Rational divides(Rational b)   //quotient of this and b
boolean equals(Rational b) //is this equal to b?
String toString()
```

Figure 2: Sample interaction with `Rational` with assertion checking disabled. The end of the output is shown here. Compare with the behavior when assertion checking is enabled, shown in Figure 3.

```
Overflow test:
Integer.MAX_VALUE: 2147483647
r3: 2147483647/3
r4: 10/1
Will attempt r3*r4 ...
r3*r4 = -10/3
```

Make sure that your program can be compiled and executed on Polaris, using the driver provided (`ClientRational.java`), as shown in Figure 3 and Figure 2.

As you can see in Figure 3, an assertion check causes an overflow that gets detected dynamically (i.e. at runtime). The flag `-ea` is used to enable assertion checking. On the other hand, when runtime assertion checking is disabled, the program should cause an (undetected) overflow, as shown in Figure 2.

Figure 3: Sample interaction with `Rational`.

```
$ ls -R hw1/
hw1/:
p1

hw1/p1:
ClientRational.java  Rational.java

$ javac hw1/p1/*java
$ java -ea hw1.p1.ClientRational
r1: 5/6
r2: 3/4
r1 + r2: 19/12
r1 - r2: 1/12
r1 * r2: 5/8
r1 / r2: 10/9

[Rational cr1 = r1;]
cr1: 5/6
[Rational n5 = new Rational(7, 8);]
r1: 5/6
n5: 7/8
r1 equals cr1: true
r1 equals n5: false

[Rational cr2 = new Rational(5, 6);]
r1: 5/6
cr2: 5/6
r1 equals cr2: true

r2 + n5: 13/8
res_r2plusn5: 13/8

Overflow test:
Integer.MAX_VALUE: 2147483647
r3: 2147483647/3
r4: 10/1
Will attempt r3*r4 ...
Exception in thread "main" java.lang.AssertionError: Overflow detected
        at hw1.p1.Rational.times(Rational.java:83)
        at hw1.p1.ClientRational.main(ClientRational.java:62)
```

Note: In `readme.txt`, also write how to compile and run your code with `ClientRational.java` on Polaris.

Deliverable: `Rational.java`, `readme.txt`.

End