The Fraction Class (continued from Lecture 14)

Recall that, last lecture, we were writing a class for rational numbers.

```java
public class Fraction {
    private long numerator;
    private long denominator;

    Besides the two constructors we wrote then, it’s also useful to have a
    constructor that copies a Fraction.

    public Fraction(Fraction original) {
        this(original.numerator, original.denominator);
    }

    Now, we can use Fractions to divide rational numbers exactly. If you divide a
    fraction a/b by another fraction c/d, the result is ad/bc.

    public void divide(Fraction f) {
        numerator = numerator * f.denominator;
        denominator = denominator * f.numerator;
        reduce();
    }

    This method operates on two Fractions: the Fraction "f", and another fraction
    called "this".

    The "this" Keyword
    ---------------------
    The method invocation frac1.divide(frac2) implicitly passes the object "frac1"
    as a parameter called "this". So we could rewrite the divide method as follows
    without changing its meaning.

    public void divide(Fraction f) {
        this.numerator = this.numerator * f.denominator;
        this.denominator = this.denominator * f.numerator;
        reduce();
    }

    In this case, "this" is optional.

    However, sometimes you need "this". For example, sometimes you want to divide
    two fractions without changing one of them. Let’s write a method called
    "quotient" that constructs a new Fraction instead of modifying an existing one.

    public Fraction quotient(Fraction f) {
        Fraction result = new Fraction(this);
        result.divide(f);
        return result;
    }
```

The first line creates a new Fraction that is a copy of the Fraction the method
was called on. For example, suppose we invoke this method by writing

```java
frac3 = frac1.quotient(frac2);
```

Just before the "quotient" method finishes, the references and objects look
like this.

```
<table>
<thead>
<tr>
<th>numerator</th>
<th>numerator</th>
<th>numerator</th>
</tr>
</thead>
<tbody>
<tr>
<td>frac1</td>
<td>frac2</td>
<td>frac3</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>9</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>denominator</th>
<th>denominator</th>
<th>denominator</th>
</tr>
</thead>
<tbody>
<tr>
<td>frac1</td>
<td>frac2</td>
<td>frac3</td>
</tr>
<tr>
<td>4</td>
<td>8</td>
<td></td>
</tr>
</tbody>
</table>
```

These variables are in two different scopes.
- The scope of "frac1" and "frac2" is the calling method, where the line
  "frac3 = frac1.quotient(frac2);" appears.
- The scope of "this", "f", and "result" is the method Fraction.quotient.

The KEY POINT is that when we call "frac1.quotient(frac2)", frac1 is passed to
the quotient method as a parameter called _this_. It doesn’t appear in the
parameter list, but it’s still being passed.

When you write "numerator" inside a method in the Fraction class, Java
interprets it as "this.numerator".

There are two very important things you must memorize about "this".
1. You cannot change the value of "this"!
   A statement like "this = frac2" will trigger a compile-time error.
2. In a _static_ method, THERE IS NO "this"!

Using "divide" and "quotient"
----------------------------

Observe that "divide" and "quotient" both do rational division, but they’re
used slightly differently.

```java
frac3 = frac1.quotient(frac2); // frac3 = frac1 / frac2;
frac1.divide(frac2);          // frac1 = frac1 / frac2;
```
Euclid's GCD algorithm
----------------------
During the last lecture, I mentioned a "reduce()" method that reduces a fraction like 6/9 to its simplest form. How can we do that? Observe that 6 and 9 are both divisible by 3, so we have

\[
\begin{array}{c|c|c}
6 & 6/3 & 2 \\
- & - & - \\
9 & 9/3 & 3 \\
\end{array}
\]

which is the simplest form of the fraction.

More generally, to reduce a fraction n/d to its simplest form, we find the \_greatest_common_divisor\_ (GCD) of n and d, and divide both of them by the GCD. How do we compute a GCD?

Euclid of Alexandria described an algorithm for computing GCDs about 2,300 years ago, in Book VII of the _Elements_, the most famous math textbook ever. It's one of the oldest known algorithms. I don't know of any earlier formal algorithm except the algorithms for doing simple arithmetic by hand (like you learned in grade school). Before we see Euclid's algorithm, let's look at the mathematics behind it.

Let GCD(a, b) be the GCD of a and b. Find GCD(a, b) is easy if one of the numbers is zero.

\[\text{GCD}(a, 0) = a. \] (Zero is divisible by everything.)

What if neither a nor b is zero? Let

\[ q = \text{floor}(a / b), \] which means \( a / b \) rounded down to the nearest integer, and
\[ r = \text{remainder}(a / b). \]

By the definition of "remainder,"

\[ a = qb + r. \]

This formula allows us to "simplify" GCD(a, b). Let's prove it.

Theorem: If a and b are divisible by some integer g, then so is r.

Proof: Because a and b are divisible by g, there are integers x and y such that \( a = xg \) and \( b = yg \).

So, \( r = a - qb = xg - qyg = (x - qy)g \).

Since \( q, x, \) and \( y \) are integers, so is \( x - qy \), so \( r \) is divisible by \( g \). QED

Theorem: GCD(a, b) = GCD(b, r).

Proof: Because a and b are divisible by GCD(a, b), so is r (see proof above).

Because b and r are divisible by GCD(b, r), so is qb + r = a.

Therefore, a, b, and r are all divisible by both GCD(a, b) and GCD(b, r).

If GCD(a, b) > GCD(b, r), then GCD(b, r) isn't really the \_greatest_common_divisor\_ of b and r, because GCD(a, b) is greater. --> GCD(a, b) <= GCD(b, r).

If GCD(a, b) < GCD(b, r), then GCD(a, b) isn't really the \_greatest_common_divisor\_ of a and b, because GCD(b, r) is greater. --> GCD(a, b) >= GCD(b, r).

Therefore, GCD(a, b) = GCD(b, r). QED

The algorithm
-------------
The idea of the algorithm is to replace "GCD(a, b)" with "GCD(b, r)", and do it over and over. The key is that \( r \) is always smaller than \( b \). Because we keep replacing \( b \) with a smaller number, eventually we will whittle it down to zero, and we can use "GCD(a, 0) = a" to finish.

Let's compute GCD(21, 78) by hand.

Dividing 21 by 78 gives 0 with remainder 21. So GCD(21, 78) = GCD(78, 21).

Dividing 78 by 21 gives 3 with remainder 15. So GCD(78, 21) = GCD(21, 15).

Continuing this way, we have

\[
\begin{align*}
\text{GCD}(21, 78) &= \text{GCD}(78, 21) \\
&= \text{GCD}(21, 15) \\
&= \text{GCD}(15, 6) \\
&= \text{GCD}(6, 3) \\
&= \text{GCD}(3, 0) \\
&= 3.
\end{align*}
\]

Observe that the second operand of GCD gets smaller on every iteration. That gives us a guarantee that the algorithm won't run forever.

Now let's write a method that computes a GCD.

```java
static private long gcd(long a, long b) {
    while (b > 0) {
        long remainder = a % b;
        a = b;
        b = remainder;
    }
    return a;
}
```

The method is static because it doesn't operate on a Fraction; it just computes the GCD of two parameters. Note that the code doesn't even need to compute \( a / b \); just \( a \) % \( b \). It's short and very fast.

Now we can use the gcd method to reduce Fractions and make sure the denominator is positive. (This method is NOT static.)

```java
private void reduce() {
    long divisor = gcd(numerator, denominator);
    numerator = numerator / divisor;
    denominator = denominator / divisor;
    if (denominator < 0) {
        numerator = -numerator;
        denominator = -denominator;
    }
}
```

A random closing thought: Euclid's algorithm requires the most iterations when you run it on two large successive Fibonacci numbers.