Homework 1: User-defined objects

cs230
Spring 2002

Allen B. Downey
Computer Science Department

Due: February 11

This assignment has two parts: one exercises the built-in class `BigInteger`; the other develops a user-defined class called `Rational`. Please read Chapters 8 and 9, and the documentation for the BigInteger class.

1 BigIntegers

BigIntegers are built-in objects that can represent arbitrarily big integers. There is no upper bound except the limitations of memory size and processing speed.

Before you start, you should look over the documentation for BigIntegers:

`http://java.sun.com/j2se/1.4/docs/api/java/math/BigInteger.html`

1. Make a new directory called `hw01`. Make a copy of `Hello.java` and make sure you can compile and run it. Rename the file `Factorial.java` and rename the class `Factorial`. As always, you should start with a working program and proceed incrementally.

2. Write a method named `factorial` that takes an integer parameter and evaluates its factorial. It can be iterative or recursive.

3. In `main`, write a loop that prints a table of the integers from 0 to 30 along with their factorials. At some point around 15, you will probably see that the answers are not right any more. The reason is that there is a finite range of values that can be represented with an integer. In order to handle bigger numbers, you have to use an alternate representation.

   One option is the `BigInteger` class. To import this class, add `import java.math.BigInteger` to the beginning of your program.

4. There are several ways to create a new `BigInteger`, but the one I recommend uses `valueOf`. The following code converts an integer to a `BigInteger`:

   ```
   int x = 17;
   BigInteger big = BigInteger.valueOf (x);
   ```

   Type in this code and try out a few simple cases like creating a BigInteger and printing it. Notice that `println` knows how to print BigIntegers!

5. Unfortunately, because BigIntegers are not primitive types, we cannot use the usual math operators on them. Instead we have to use object methods like `add`. In order to add two BigIntegers, you have to invoke `add` on one of the objects and pass the other as an argument. For example:

```
BigInteger small = BigInteger.valueOf (17);
BigInteger big = BigInteger.valueOf (1700000000);
BigInteger total = small.add (big);
```

Try out some of the other methods, like `multiply` and `pow`.

6. Convert `factorial` so that it performs its calculation using `BigInteger`s, and then returns the `BigInteger` as a result. You can leave the parameter alone—it will still be an integer. You should perform this step gradually, using the development plan described in Appendix A.

7. Try printing the table again with your modified factorial function. Is it correct up to 30? How high can you make it go? On my machine, I calculated the factorial of all the numbers from 0 to 999, but it took over 2 minutes. The last number, 999!, has 2565 digits.

2 Rational numbers

A rational number is a number that can be represented as the ratio of two integers. For example, 2/3 is a rational number, and you can think of 7 as a rational number with an implicit 1 in the denominator. For this assignment, you are going to write a class definition for rational numbers.

1. Go to

   `http://rocky.wellesley.edu/cs230/code/hw01`

   and download the files `Test.java` and `Complex.java`. Compile and run them using a command like:

   ```
   javac Test.java; java Test
   ```

   Notice that the Java compiler figures out that it needs to compile `Complex.java` in order to get `Test` to work.

   Print a copy of both files and make sure you understand how they work. You should use this example as a template for what follows.

2. Create a new file named `Rational.java`. You might want to start with `Complex.java` and modify it, or you might want to start with a clean file.

3. Write a class definition for a `Rational` object that has two integers as instance variables; they will contain the numerator and denominator of a rational number.

4. Write a constructor that takes no arguments and that sets the two instance variables to zero.

5. Write a method called `printRational` that takes a Rational object as an argument and prints it in some reasonable format.

6. Add some lines to `Test` that create a new object with type Rational, set its instance variables to some values, and print the object.

7. At this stage, you have a minimal testable (debuggable) program. Debug it. Whenever I start writing a new class, I start with these three steps: instance variables, a simple constructor, and a print method. Then I add methods and test them one at a time.

2.1 Another constructor

1. Write a second constructor for your class that takes two arguments and that uses them to initalize the instance variables.

2. Add lines to `main` that test both constructors. You will probably want to add a print statement to the constructors so that you can tell exactly when each constructor gets invoked.

2.2 Modifers and functions

1. Write a new method called `negate` that reverses the sign of a rational number. This method should be a modifier, so it should return void. Add lines to `main` to test the new method.

2. Write a method called `invert` that inverts the number by swapping the numerator and denominator. Add lines to `main` to test the new method.

3. Write a method called `toDouble` that converts the rational number to a double (floating-point number) and returns the result. This method is a pure function; it does not modify the object. As always, test the new method.

4. Write a modifier named `reduce` that reduces a rational number to its lowest terms by finding the GCD of the numerator and denominator and then dividing top and bottom by the GCD. You can use the `gcd` method from the previous homework.

5. Write a method called `add` that takes two Rational numbers as arguments and returns a new Rational object. The return object, not surprisingly, should contain the sum of the arguments.

   There are several ways to add fractions. You can use any one you want, but you should make sure that the result of the operation is reduced so that the numerator and denominator have no common divisor (other than 1).

3 What to turn in

Please prepare and print your code following the instructions on the previous homework.