COLBY COLLEGE
Computer Science Department

CS115 **Allen B. Downey**
Spring 1998

Assignment 8: Strings, series, and objects

Due: Wednesday 3 November

Series from heck

One way to evaluate e^x is to use the infinite series expansion

$$e^x = 1 + x + x^2/2! + x^3/3! + x^4/4! + ... \tag{1}$$

If the loop variable is named `i`, then the ith term is equal to $x^i/i!$.

1. Write a method called `myexp` that adds up the first `n` terms of the series shown above. You may use the `factorial` method from the book.
2. You can make this method much more efficient if you realize that in each iteration the numerator of the term is the same as its predecessor multiplied by `x` and the denominator is the same as its predecessor multiplied by `i`. Use this observation to eliminate the calls to `Math.pow` and `factorial`, and check that you still get the same result.
3. Write a method called `check` that takes a single parameter, `x`, and that prints the values of `x`, `Math.exp(x)` and `myexp(x)` for various values of `x`. The output should look something like:

```
1.0      2.708333333333333        2.718281828459045
```

HINT: you can use the String `"`
`t"` to print a tab character between columns of a table.

4. Vary the number of terms in the series (the second argument that `check` sends to `myexp`) and see the effect on the accuracy of the result. Adjust this value until the estimated value agrees with the "correct" answer when `x` is 1.
5. Write a loop in `main` that invokes `check` with the values 0.1, 1.0, 10.0, and 100.0. How does the accuracy of the result vary as `x` varies? Compare the number of digits of agreement rather than the difference between the actual and estimated values.
6. Add a loop in `main` that checks `myexp` with the values -0.1, -1.0, -10.0, and -100.0. Comment on the accuracy.

Captain Crunch

1. The Captain Crunch decoder ring works by taking each letter in a string and adding 13 to it. For example, 'a' becomes 'n' and 'b' becomes 'o'. The letters "wrap around" at the end, so 'z' becomes 'm'.

   Write a method that takes a String and that returns a new String containing the encoded version. You should assume that the String contains upper and lower case letters, and spaces, but no other punctuation. Lower case letters should be tranformed into other lower case letters; upper into upper. You should not encode the spaces.

2. Generalize the Captain Crunch method so that instead of adding 13 to the letters, it adds any given amount. Now you should be able to encode things by adding 13 and decode them by adding -13. Try it.

Using objects as parameters

1. Start with a clean copy of the Snowperson assignment. You can use my solutions if you want or you can use your own.

   `http://rocky.colby.edu/cs151/solutions/hw4/`

   If you use your solution, you might want to take a minute to review it, make sure you remember how it works, and clean it up a little.

2. Add a few lines of code to the program and experiment with Rectangle and Point objects a bit. Try out one or two of the examples from the book or Quiz 12. Try creating an object variable (but no object) and then access one of the instance variables. What happens? Look at the output from the exception handler and see if you can find the line of the program that caused the error.

3. Modify the Snowperson code so that every method that receives a bounding box as an argument gets a single Rectangle object instead of four integers. For example, the declaration of `paintSnowperson` would look like

   ```
   public void paintSnowperson (Graphics g, Rectangle bb)
   ```

4. Please turn in a paragraph that compares the two versions, discussing things like readability, debuggability, code length and encapsulation.

   In the Java library, they had the option of using Rectangles to pass parameters to methods like drawRect and drawOval, but they chose to pass four integers instead. In fact, there are only a couple of methods in the whole Java library that use Rectangles at all. What do you think of this decision? Would you prefer a version of drawOval that takes a single Rectangle as an argument? What if you could have both?

Using objects as return values

1. You are probably getting sick of the factorial method by now, but just for practice, write an iterative version of `factorial`. Put it in a new project named `Big.prj`.

2. Print 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. Why not?

3. Go to the Sun web page that contains all the Java documentation; there is a link to it from the class web page. Click on the `java.math` package and then click on the `BigInteger` class. Print the documentation of the BigInteger class. Read it.

   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.

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! Don't forget to add `import java.math.BigInteger` to the beginning of your program.

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 BigIntegers, and then returns the BigInteger as a result. You can leave the parameter alone—it will still be an integer.

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.