CSC 204 Lab 5: Numbers, Primitive and Fancy

Chapter 4 of the text covers Java's primitive types. In this lab, you'll see how the difference in the ranges of these types affect a program. You'll have a chance to try out some of the methods in the Math libraries. You'll also see how to use the BigInteger library.

There's some review of command line parameters and exception handling as well. The program uses input from the command line instead of running interactively.

**Goals**
- After doing this lab, you should be able to:
  - Discuss the boundary limits of the types short, int, long, float, and double.
  - Find the limits on the primitive types from the appropriate methods in java.lang.
  - Describe the rounding options of the Math class.
  - Use the basic features of BigInteger.

**Lab Preparation**

Read through this lab. Review the material in Chapter 4.

You may find the on-line documentation for the Math and BigInteger classes useful. Go to java.sun.com and select API specifications in the Reference section on the left column to get to the APIs.

**Materials Needed**

Be sure you have the following on hand during the lab.

- This sheet
- Your textbook.

**Lab Setup**

Make a Lab5 project. Copy the three .java files from our Lab5 folder on Blackhawk into the src folder in the Lab5 project.

**Part 1: Simple factorials**

1. The program factorial.java will compute the factorial of an integer value. Take a look at it. In particular, notice how the initial comments are used to tell you how to call the program.

   *How do you call the program to find 10!?*
2. Factorials grow very quickly. Compute the values below:

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>10!</td>
<td></td>
</tr>
<tr>
<td>11!</td>
<td></td>
</tr>
<tr>
<td>12!</td>
<td></td>
</tr>
<tr>
<td>13!</td>
<td></td>
</tr>
</tbody>
</table>

Something very peculiar has happened.

Describe any observed peculiar behavior?

3. This program can only compute factorials up to 12!. That's not very useful. Let's try changing the type of the result to get larger values.

**Part 2: Bigger factorials**

1. Ah! Maybe changing the result type from `int` to `long` will fix the program. Copy the program `factorial.java` to `lfactorial.java`. Now, go in and change the name to `lfactorial` and everywhere the result is declared from `int` to `long`. (This will be just where `result`, where the factorial method states its return value, and where the factorial method declares its result, `product`.)

2. Compile and run the program. Check the values for 15! and 16!. Keep trying larger values until you see a problem (don't worry, it won't take long!). Give the values and results here:

Has changing the type to `long` helped much?
Part 3: Even bigger factorials

1. So, widening to a long doesn't help much. Maybe changing the result type from long to float will let us compute larger factorials. We'll have to sacrifice some digits of precision, but the extra range may be worth it. Copy the program lfactorial.java to ffactorial.java. Now, go in and change the name to ffactorial and everywhere the result is declared from long to float.

2. Compile and run the program. Compare the values for 15! and 16! with those given by lfactorial. When do you get fewer digits of precision from ffactorial than from lfactorial?

3. Keep trying larger values until you see a problem (don't worry, it won't take long!). Give the values and results here:

Since the floating point numbers have a representation for \( \infty \), we get that instead of overflow values when using floating point numbers.
Part 4: Still bigger factorials

We are rapidly running out of types, but let's try doubles. You got it, copy the program `ffactorial.java` to `dfactorial.java`. Now, go in and change the name to `dfactorial` and everywhere the result is declared from `float` to `double`.

Compile and run the program. **Now how large a value can you compute?**
(You'll want to step by more than 1 in this case. For example, you might want to try 50!, 100!, 150!, etc. to help find the largest value.)

Change the program to work with bytes and shorts. Summarize all of your results in the table below.

<table>
<thead>
<tr>
<th></th>
<th>Largest n</th>
<th>n!</th>
</tr>
</thead>
<tbody>
<tr>
<td>byte</td>
<td></td>
<td></td>
</tr>
<tr>
<td>short</td>
<td></td>
<td></td>
</tr>
<tr>
<td>int</td>
<td></td>
<td></td>
</tr>
<tr>
<td>long</td>
<td></td>
<td></td>
</tr>
<tr>
<td>float</td>
<td></td>
<td></td>
</tr>
<tr>
<td>double</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

While the type `double` is an improvement over the others, there's still a lot of integers we cannot compute the factorial for. We also lose digits of precision when we use floating point types. To improve this, we'll have to move beyond the primitive types. Before doing that, let's look at the other primitive types a bit more closely.
**Part 5: Limits to types**

Since the types in Java have fixed lengths, they have predetermined maximum and minimum values. You can get these from the respective classes, so the largest value of a `byte` is `Byte.MAX_VALUE`.

Edit the program `NumberLimits.java` to print the minimum and maximum values of each of the numeric types. You will be asked to turn a copy of this program in for this lab.

**Part 6: Rounding off with the math library**

There are many different ways to convert floating point numbers to integers. Using the program in `RoundingOff.java`, fill in the table below with the different values that are returned. The methods, `round`, `ceil`, `floor`, `rint` are all defined in the Math library.

<table>
<thead>
<tr>
<th>Expression</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>round (3.4)</code></td>
<td></td>
</tr>
<tr>
<td><code>round (3.5)</code></td>
<td></td>
</tr>
<tr>
<td><code>round (3.6)</code></td>
<td></td>
</tr>
<tr>
<td><code>ceil (3.4)</code></td>
<td></td>
</tr>
<tr>
<td><code>ceil (-3.4)</code></td>
<td></td>
</tr>
<tr>
<td><code>floor (3.6)</code></td>
<td></td>
</tr>
<tr>
<td><code>floor (-3.6)</code></td>
<td></td>
</tr>
<tr>
<td><code>rint (3.5)</code></td>
<td></td>
</tr>
<tr>
<td><code>rint (4.5)</code></td>
<td></td>
</tr>
<tr>
<td><code>rint (3.2)</code></td>
<td></td>
</tr>
<tr>
<td><code>rint (-3.2)</code></td>
<td></td>
</tr>
<tr>
<td><code>rint (-3.8)</code></td>
<td></td>
</tr>
</tbody>
</table>
Part 7: Biggest factorials

1. To go beyond the primitive types, we need to use a new class. Fortunately, the `BigInteger` class is one of the standard Java classes, so to get extremely large integers, we can use this class. Copy the program `factorial.java` to `bifactorial.java`. Now, go in and change the name to `bifactorial` and everywhere the result is declared from `int` to `BigInteger`.

2. Try compiling this program. You should notice a lot of errors. Changing to a non-primitive value takes a bit of doing. First of all, you need to include the library at the top of the program with
   ```java
   import java.math.BigInteger;
   ```

3. Second, there is no direct conversion between integers and BigIntegers, so you'll need to replace
   ```java
   product = 1;
   ```
   with
   ```java
   product = BigInteger.ONE;
   ```
   If you are using an integer value other than 0 or 1, you can create a new BigInteger with its value using `BigInteger.valueOf (the integer)`. So, you could have initialized the product using
   ```java
   product = BigInteger.valueOf(1);
   ```

4. Finally, you can not use the standard mathematical symbols `+`, `-`, `*`, etc on new types. (Some programming languages do allow you to redefine these, but Java is not one of those languages.) Instead of `product *= i;` you'll need
   ```java
   product = product.multiply (BigInteger.valueOf(i));
   ```

5. Get your program to compile and run. Then computes the factorial of a large number (somewhere between 200 and 1000).

   **Deliverables**

   Email the programs from parts 5 and 7 to your instructor and turn in this sheet with the answers filled in for credit on this lab.