Name(s)\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**CS100 Lab 05 - Limits and Accuracy of Computation -** One submission (questions 6 through 10) per 2-3 person team. Due midnight, the day before next week’s lab.

In this lab you will experiment with a few integer and floating point types in Java in order to verify their ranges, and to find out what happens when we exceed (or under-run) their capacities. We will be examining the short, int, float, and double. Open DatatypeTests.java in Notepad++ or Eclipse. Compile and run it. Here is a sample run:

Enter an integer to calculate factorials: **5**

(short) factorial of 5 = 120

(int) factorial of 5 = 120

(float) factorial of 5 = 120.0

(float) inverse factorial of 5 = 0.008333334

(double) factorial of 5 = 120.0

(double) inverse factorial of 5 = 0.008333333333333333

For integer types (short and int), the program computes the factorial of the number specified as a command line argument — i.e., for some integer n, the value n×(n-1)×(n-2)×· · ·×2×1 is computed, we write the factorial of n as n!. For floating point types, the same value is computed, along with the inverse factorial, i.e., the value 1/n! Note that the factorial of 0 is defined as 1 (i.e., 0! = 1).

Experiments (COMPLETED AS A CLASS AT THE START OF LAB WITH TA)

1. With our first set of results (for 5!), we can already draw some conclusions. What seem to be the amount of precision (in decimal digits) for the float and double types, respectively?
2. Try playing with numbers in the range 6-10 (we’ll start small — factorials grow fast!) What interesting thing do you notice? Can you make an educated guess as to why this is happening?
3. Keep going up slowly through our integer arguments, say up to around 15. At what point does the int type stop giving us correct answers? Why is it safe at this point to continue using the floating point answers to approximate the “right” answer?
4. At what point does the amount of precision supported by the float type becomes insufficient?
5. At what point do we stop getting a numerical result for the float type? What happens? Can you justify the results?

Questions (COMPLETED IN TEAMS AND SUBMITTED)

1. We were fortunate in our set of experiments that we had multiple results to base our analyses on — this is not always the case. If we were dealing only with integer values, would we be able to discern immediately if the answer made mathematical sense? You should use an example to back up your claim.
2. It is probably evident why a computer engineer would need to understand the limits of accuracy and computation, but why would a software engineer need to know these things?
3. Discuss 2 reasons in detail why computers are not perfect machines for numeric computation.
4. Write java code to sum up (DO NOT MULTIPLY) 0.2 multiple times, maybe up to 100 or so, and print out the running total on each iteration. Explain what you see and why. Would you see the same issue if you summed up 0.25 and why or why not?
5. Here is java code to calculate float machine epsilon; the upper bound on the relative error due to rounding in floating point arithmetic. Anything smaller than that will round to zero.

float machEps = 1.0f;

do {

machEps /= 2.0f;

} while ((float)(1.0 + (machEps/2.0)) != 1.0);

System.out.println("\nCalculated FLOAT machine epsilon = " + machEps );

Add java code to the above to use that calculated float machine epsilon machEps to demonstrate the computer  
 arithmetic is not always commutative when dealing with very large and very small numbers.