**Java Rounding Method Comparison**

Looking for a good rounding method for a project, I stumbled upon what appears to be a great debate about proper rounding technique. Wanting to form my own conclusions, I ran a test to compare the performance of the following five methods of rounding a number to n decimal places:

* y = Math.*round*(x \* {10n}) / {10n};
* y = Math.*floor*(x \* {10n} + 0.5) / {10n};
* bd = **new** BigDecimal(x).setScale({n}, RoundingMode.*HALF\_UP*);
* bd = BigDecimal.*valueOf*(x).setScale({n}, RoundingMode.*HALF\_UP*);
* df = **new** DecimalFormat("#.{#n}");  
  z = df.format(x);

I first ran a speed test on each method with a variety of variables. I ran 1 million repetitions in each trial, ran a handful of trials for each test, and estimated the average time.

I generated the input using the following methods:

* x = i / 1000000.0;
* x = Math.random();

I stored the output in the following formats:

* Native output type
* double
* String
* Both double and String

I explored rounding to different numbers of decimal places, but found no significant time difference in any of the methods.

I found that round and floor were generally the fastest, and DecimalFormat was always the slowest, but there were some interesting anomalies between tests. floor may be slightly faster than round, but the difference wasn’t big enough to be significant. The new BigDecimal method was actually faster than round and floor for iterative input and String output. When outputting doubles, round and floor took the same amount of time for both types of input, accounting for the time difference in generating the input. But when outputting Strings, the random input took significantly longer than the iterative input. In general, random input was much slower than iterative input. BigDecimal.valueOf was faster for random input, but new BigDecimal was faster for iterative input.

I then attempted to test the accuracy of each method. I struggled with an effective way to do this. I first compared the String output of each method with the expected output and counted the number of errors, for n = 1, 2, and 3, using iterative input with one more decimal place than the output. A quick test showed that the output always erred on the same values no matter the number of decimal places in the input. Both DecimalFormat and BigDecimal.valueOf were 100% accurate. round and floor performed identically, and were over 99% accurate in each test. new BigDecimal was the worst, with slightly better than 95% accuracy. In all cases, the errors occurred at corner conditions. During this test, I discovered that BigDecimals do not automatically strip trailing zeroes.

I then attempted to measure the imprecision of the double output of each method, with the same conditions as the previous test. I found that the imprecision of the double output depended only on the number of decimal places and not the generation method. The imprecision of the double type is about 17 orders of magnitude smaller than the value of the double.

My test seems to show that the best rounding method depends on the context of use. round and floor are essentially identical in performance. Between the two, I would use round just for slightly cleaner code. round and floor are good for situations where speed is crucial and precision is less important. Many would argue that the slight chance for a wrong result is never acceptable, but I think it depends on the situation. round and floor are not very useful for large values, since they are limited by the range of their long return type. doubles are also less precise for large values, since the double type can accurately represent about 16 digits.

BigDecimals are best for high precision uses, since they can represent values with arbitrary precision. Based on my test, it’s hard to say how much slower BigDecimal is than round and floor. BigDecimals can be a little difficult to display how you want them, since stripTrailingZeros does not seem to work for values of 0, and the toString method has some slightly bizarre formatting rules. I would say, when in doubt, use BigDecimal. The only caveat, never use new BigDecimal(double val), since it tries to represent val exactly, capturing the imprecision of doubles that you’re trying to avoid in the first place.

The purpose of DecimalFormat is formatting for output. Since the return type is String, it doesn’t make sense to use DecimalFormat as part of a calculation. If speed is not a concern, DecimalFormat is the best option for output formatting, since it is the most versatile in terms of formatting options and flexibility.