İbrahim Başar YARGICI 12/2/2020
Data Structures Midterm

# Question 18:

Package java.math contains a class BigDecimal, used to represent an arbitrary-precision decimal number. Read the documentation for BigDecimal and answer the following questions:

a. Is BigDecimal an immutable class?

b. If bd1.equals(bd2) is true, what is bd1.compareTo(bd2)?

c. If bd1.compareTo(bd2) is 0, when is bd1.equals(bd2) false?

d. If bd1 represents 1.0 and bd2 represents 5.0, by default what is bd1.divide(bd2)?

e. If bd1 represents 1.0 and bd2 represents 3.0, by default what is bd1.divide(bd2)?

f. What is MathContext.DECIMAL128?

g. Modify the BigRational class to store a MathContext that can be initialized from an additional BigRational constructor (or which defaults to MathContext.UNLIMITED). Then add a toBigDecimal method to the BigRational class.

# Answers 18:

a. Yes. BigDecimal class is immutable

b. Info: equals() returns true if both values are exactly same in value and scale. compareTo() only compares their numeric value. We know that compareTo() returns:
-1, 0, or 1 as this BigDecimal is numerically less than, equal to, or greater than val.
So if bd1.equals(bd2) is true, bd1.compareTo(bd2) will return 0.

c. compareTo() will return true if the values are same, but equals() looks the scale too. So for example if bd1 is "2.0" and bd2 is "2.00" compareTo() will return 0 but equals() will return false. Because scales are not same.

d. If bd1 represents 1.0 and bd2 represents 5.0, bd1.divide(bd2) will give 0.2 result because the result terminating in one point.

e. If bd1 represents 1.0 and bd2 represents 3.0, bd1.divide(bd2) will give ArithmeticException. Because the result is 0.3333 and there is no exact representation for decimal result.

f. First Why we are using MathContext:
- precision: the number of digits to be used for an operation; results are rounded to this precision
- roundingMode: a RoundingMode object which specifies the algorithm to be used for rounding.

In document: MathContext object with a precision setting matching the IEEE 754R Decimal128 format, 34 digits, and a rounding mode of HALF_EVEN, the IEEE 754R default.

What does it mean: decimal128 supports exponents between -6143 and +6144; significand has 34 digits (i.e. 0.000000000000000000000000000000000-9.999999999999999999999999999999999).
For example, If bd1 represents 1.0 and bd2 represents 3.0, bd1.divide(bd2, MathContex.DECİMAL128) will give you 0.33333.. there will be 34 units of 3 and this will solve our ArithmeticException problem.

g.

- private MathContext mathContext = MathContext.UNLIMITED; // Default unlimited MathContext
- public BigRational(MathContext mathContext) {

  this.mathContext = mathContext;

  }
- public BigDecimal toBigDecimal(BigRational bigRational) {
  // Taking numerator and denominator which are BigInteger. To use them first we convert them to string.
  String num = bigRational.num.toString();
  String den = bigRational.den.toString();
  // After translating to string, we should divide num to den and it should give us double result.
  double a = Double.*parseDouble*(num) / Double.*parseDouble*(den);
  // Returning new BigDecimal object which val is a and MathContext is from class mathContext variable.
  return new BigDecimal(a, mathContext);
  }