“java.math” package

Description for arithmetic of decimals and integers with arbitrary precision. It contains the classes Big Integer –just as the primitive integer -, Overflow if it loses precision, modular arithmetic, GCM (the highest common denominator) generation of prime numbers, manipulation of bits, primality testing (whether the number is a prime number or not).
Bigdecimal: decimal arithmetic with arbitrary precision.
MathContext: it encapsulates preset concepts (precision + rounding off-mode)

Unit tests

Black box

At the beginning, test cases were divided by classes, because the amount of code tested occupied a lot of space. There are even test cases for one method that occupy 5 mb, which is almost impossible to handle.
The testing methodology was: with random code generators, with precision code generators or simply by trying particular cases.
Much of these texts were generated wrongly and have as an exit the obtained result in the expected result, and vice versa, the expected result in the obtained result.
Seven faults have been found in SUN, all of which raise an exception that shouldn’t be there, while they should throw lack of heap, this occurs when a parameter is passed as an integer from [Integer.MaxValue-7 .. Integer.MaxValue].

- testBigIntegerIntRandom007
  (ar.org.fitc.test.math.biginteger.TestBigIntegerConstructors)
  junit.framework.AssertionFailedError: Should not raise an
  Exception...java.lang.NegativeArraySizeException
- testProbablePrime002
  (ar.org.fitc.test.math.biginteger.methods.TestProbablePrime)
  junit.framework.AssertionFailedError: Should not raise an
  Exception...java.lang.NegativeArraySizeException
- testProbablePrime004
  (ar.org.fitc.test.math.biginteger.methods.TestProbablePrime)
  junit.framework.AssertionFailedError: Should not raise an
  Exception...java.lang.NegativeArraySizeException
- testProbablePrime011
  (ar.org.fitc.test.math.biginteger.methods.TestProbablePrime)
  junit.framework.AssertionFailedError: Should not raise an

Exception...java.lang.NegativeArraySizeException

- testProbablePrime013
  (ar.org.fitc.test.math.biginteger.methods.TestProbablePrime)
  junit.framework.AssertionFailedError: Should not raise an
  Exception...java.lang.NegativeArraySizeException
- testProbablePrime015
  (ar.org.fitc.test.math.biginteger.methods.TestProbablePrime)
  junit.framework.AssertionFailedError: Should not raise an
  Exception...java.lang.NegativeArraySizeException
- testProbablePrime016
  (ar.org.fitc.test.math.biginteger.methods.TestProbablePrime)
  junit.framework.AssertionFailedError: Should not raise an
  Exception...java.lang.NegativeArraySizeException

These methods in our package throw Java heap space.
The testing methods divide and divideAndRemainder had to be corrected because they did not execute the tests correctly. This happened because an if sentence was placed incorrectly, which made the tests throw a correct result all the time.
In our implementation the tests that used probably a prime with certainty of over 10000 were reduced to 100, because the amount of memory required made the PC hung up.
New bugs where founded and they are specified in the document Bugs_sun.txt

White Box

The emphasis that Math puts on efficiency and its easy way of corroborating he correction for almost the whole package took the whit box test to a performance analysis. Since we were centered on the code to be presented in the Milestone 2, which did not have any flaw, we only gave our opinions about optimizations.
Among other suggestions: the exchange of FOR sentences for System.arraycopy; the multiplication made *shift* operations, which suggested choosing to apply said function with a number that is multiple of 32. The code of a small passer of MathContext was difficult to read and understand.
Since in Java every object must be coinciding (or make it clear that it is not) this classes achieve this predicate when they are immutable. (When a BigInteger is generated it is never altered). We controlled that this immutability was respected. In particular, there is a method (updFirst) that is executed before returning a BigInteger. After this method is carried out the object is immutable.
We also verified the coverage by adding some tests to improve it. Since the project is not finished yet, the maximum coverage is not required, leaving thus the finalization of the coverage to the third Milestone.

Performance test

As regards performance tests, the profiler of NetBeans was used. At least one test case of each method was executed, and in case the method presented an exception, or if the result was too different, or with the integration test on SUN, the test cases were executed as well. But they were not performed on our package due to the fact that it could not be carried out for the Milestone 2 because not all the classes are implemented.
Clean tables will be found in an excel file on folder “math” and on folder “Windows” or “Linux” according to the operating system wanted.
The file is named “mathperformance” and it disposes of one spreadsheet for the times of the SUN package, another for the times of our package and a third one with the time differences. The negative values mean that the difference is in favor of SUN and the positive values mean that the difference is in favor of our project. Those methods with a yellow background are the ones that will be presented on the Milestone 2.

Integration tests

As regards the integration test, we followed a series of methods initiated for each constructor, for the classes BigDecimal and BigInteger separately (“BigDecimal.txt”, “BigInteger.txt”), and in groups. The results compare with the exit of our implementation.