Math

This package contains a variety of mathematical utilities.

Contents

- Basic standalone math functions are divided into the classes `IntMath`, `LongMath`, `DoubleMath`, and `BigIntegerMath` based on the primary numeric type involved. These classes have parallel structure, but each supports only the relevant subset of functions. Note that similar functions that are less *mathematical* in nature might instead be found in `com.google.common.primitives` classes like `Ints`.
- A variety of statistical calculations (mean, median, etc.) are provided, for both single and paired data sets. Start by reading this overview rather than just browsing Javadoc.
- `LinearTransformation` represents a linear conversion between `double` values of the form `y = mx + b`; for example, a conversion between feet and meters, or between Kelvins and degrees Fahrenheit.

Examples

```
int logFloor = LongMath.log2(n, FLOOR);

int mustNotOverflow = IntMath.checkedMultiply(x, y);

long quotient = LongMath.divide(knownMultipleOfThree, 3, RoundingMode.UNNECESSARY);
// fail fast on non-multiple of 3

BigInteger nearestInteger = DoubleMath.roundToBigInteger(d, RoundingMode.HALF_EVEN);

BigInteger sideLength = BigIntegerMath.sqrt(area, CEILING);
```

Why use these?

- These utilities are already exhaustively tested for unusual overflow conditions. Overflow semantics, if relevant, are clearly specified in the associated documentation. When a precondition fails, it fails fast.
- They have been benchmarked and optimized. While performance inevitably varies depending on particular hardware details, their speed is competitive with -- and in some cases, significantly better than -- analogous functions in Apache Commons `MathUtils`.
- They are designed to encourage readable, correct programming habits. The meaning of `IntMath.log2(x, CEILING)` is unambiguous and obvious even on a casual read-through. The meaning of `32 - Integer.numberOfLeadingZeros(x - 1)` is not.

Note: these utilities are not especially compatible with GWT, nor are they optimized for GWT, due to differing overflow logic.

Math on Integral Types

These utilities deal primarily with three integral types: `int`, `long`, and `BigInteger`. The math utilities on these types are conveniently named `IntMath`, `LongMath`, and `BigIntegerMath`.

Checked Arithmetic

We provide arithmetic methods for `IntMath` and `LongMath` that fail fast on overflow instead of silently ignoring it.

IntMath	**LongMath**
IntMath.checkedAdd	LongMath.checkedAdd
IntMath.checkedSubtract	LongMath.checkedSubtract
IntMath.checkedMultiply	LongMath.checkedMultiply
IntMath.checkedPow	LongMath.checkedPow

```
IntMath.checkedAdd(Integer.MAX_VALUE, Integer.MAX_VALUE); // throws
ArithmeticException
```

Real-valued methods

`IntMath`, `LongMath`, and `BigIntegerMath` have support for a variety of methods with a "precise real value," but that round their result to an integer. These methods accept a `java.math.RoundingMode`. This is the same `RoundingMode` used in the JDK, and is an enum with the following values:

- `DOWN`: round towards 0. (This is the behavior of Java division.)
- `UP`: round away from 0.
- `FLOOR`: round towards negative infinity.
- `CEILING`: round towards positive infinity.
- `UNNECESSARY`: rounding should not be necessary; if it is, fail fast by throwing an `ArithmeticException`.
- `HALF_UP`: round to the nearest half, rounding `x.5` away from 0.
- `HALF_DOWN`: round to the nearest half, rounding `x.5` towards 0.
- `HALF_EVEN`: round to the nearest half, rounding `x.5` to its nearest even neighbor.

These methods are meant to be readable when used: for example, `divide(x, 3, CEILING)` is completely unambiguous even on a casual read-through.

Additionally, each of these functions internally use only integer arithmetic, except in constructing initial approximations for use in `sqrt`.

Operation	**IntMath**	**LongMath**	**BigIntegerMath**
Division	divide(int, int, RoundingMode)	divide(long, long, RoundingMode)	divide(BigInteger, BigInteger, RoundingMode)
Base-2 logarithm	log2(int, RoundingMode)	log2(long, RoundingMode)	log2(BigInteger, RoundingMode)
Base-10 logarithm	log10(int, RoundingMode)	log10(long, RoundingMode)	log10(BigInteger, RoundingMode)
Square root	sqrt(int, RoundingMode)	sqrt(long, RoundingMode)	sqrt(BigInteger, RoundingMode)

```
BigIntegerMath.sqrt(BigInteger.TEN.pow(99), RoundingMode.HALF_EVEN);
   // returns 31622776601683793319988935444327185337195551393252
```

Additional functions

We provide support for a few other mathematical functions we've found useful.

Operation	`IntMath`	`LongMath`	`BigIntegerMath`
Greatest common divisor	`gcd(int, int)`	`gcd(long, long)`	In JDK: `BigInteger.gcd(BigInteger)`
Modulus (always nonnegative, -5 mod 3 is 1)	`mod(int, int)`	`mod(long, long)`	In JDK: `BigInteger.mod(BigInteger)`
Exponentiation (may overflow)	`pow(int, int)`	`pow(long, int)`	In JDK: `BigInteger.pow(int)`
Power-of-two testing	`isPowerOfTwo(int)`	`isPowerOfTwo(long)`	`isPowerOfTwo(BigInteger)`
Factorial (returns `MAX_VALUE` if input too big)	`factorial(int)`	`factorial(int)`	`factorial(int)`
Binomial coefficient (returns `MAX_VALUE` if too big)	`binomial(int, int)`	`binomial(int, int)`	`binomial(int, int)`

Floating-point arithmetic

Floating point arithmetic is pretty thoroughly covered by the JDK, but we added a few useful methods to `DoubleMath`.

Method	Description
`isMathematicalInteger(double)`	Tests if the input is finite and an exact integer.
`roundToInt(double, RoundingMode)`	Rounds the specified number and casts it to an int, if it fits into an int, failing fast otherwise.
`roundToLong(double, RoundingMode)`	Rounds the specified number and casts it to a long, if it fits into a long, failing fast otherwise.
`roundToBigInteger(double, RoundingMode)`	Rounds the specified number to a `BigInteger`, if it is finite, failing fast otherwise.
`log2(double, RoundingMode)`	Takes the base-2 logarithm, and rounds to an `int` using the specified `RoundingMode`. Faster than `Math.log(double)`.