java.math

Class BigDecimal

java.lang.Object
  java.lang.Number
    java.math.BigDecimal

All Implemented Interfaces:

Serializable, Comparable<BigDecimal>

```
public class BigDecimal
extends Number
implements Comparable<BigDecimal>
```

Immutable, arbitrary-precision signed decimal numbers. A `BigDecimal` consists of an arbitrary precision integer *unscaled value* and a 32-bit integer *scale*. If zero or positive, the scale is the number of digits to the right of the decimal point. If negative, the unscaled value of the number is multiplied by ten to the power of the negation of the scale. The value of the number represented by the `BigDecimal` is therefore $(unscaledValue \times 10^{-scale})$.

The `BigDecimal` class provides operations for arithmetic, scale manipulation, rounding, comparison, hashing, and format conversion. The `toString()` method provides a canonical representation of a `BigDecimal`.

The `BigDecimal` class gives its user complete control over rounding behavior. If no rounding mode is specified and the exact result cannot be represented, an exception is thrown; otherwise, calculations can be carried out to a chosen precision and rounding mode by supplying an appropriate `MathContext` object to the operation. In either case, eight *rounding modes* are provided for the control of rounding. Using the integer fields in this class (such as `ROUND_HALF_UP`) to represent rounding mode is largely obsolete; the enumeration values of the `RoundingMode enum`, (such as `RoundingMode.HALF_UP`) should be used instead.

When a `MathContext` object is supplied with a precision setting of 0 (for example, `MathContext.UNLIMITED`), arithmetic operations are exact, as are the arithmetic methods which take no `MathContext` object. (This is the only behavior that was supported in releases prior to 5.) As a corollary of computing the exact result, the rounding mode setting of a `MathContext` object with a precision setting of 0 is not used and thus irrelevant. In the case of divide, the exact quotient could have an infinitely long decimal expansion; for example, 1 divided by 3. If the quotient has a nonterminating decimal expansion and the operation is specified to return an exact result, an `ArithmeticException` is thrown. Otherwise, the exact result of the division is returned, as done for other operations.

When the precision setting is not 0, the rules of `BigDecimal` arithmetic are broadly compatible with selected modes of operation of the arithmetic defined in ANSI X3.274-1996 and ANSI X3.274-1996/AM 1-2000 (section 7.4). Unlike those standards, `BigDecimal` includes many rounding modes, which were mandatory for division in `BigDecimal` releases prior to 5. Any conflicts between these ANSI standards and the `BigDecimal` specification are resolved in favor of `BigDecimal`.

Since the same numerical value can have different representations (with different scales), the rules of arithmetic and rounding must specify both the numerical result and the scale used in the result's representation.

In general the rounding modes and precision setting determine how operations return results with a limited number of digits when the exact result has more digits (perhaps infinitely many in the case of division) than the number of digits returned. First, the total number of digits to return is specified by the `MathContext`'s `precision` setting; this determines the result's *precision*. The digit count starts from the leftmost nonzero digit of the exact result. The rounding mode determines how any discarded trailing digits affect the returned result.

For all arithmetic operators , the operation is carried out as though an exact intermediate result were first calculated and then rounded to the number of digits specified by the precision setting (if necessary), using the selected rounding mode. If the exact result is not returned, some digit positions of the exact result are discarded. When rounding increases the magnitude of the returned result, it is possible for a new digit position to be created by a carry propagating to a leading "9" digit. For example, rounding the value 999.9 to three digits rounding up would be numerically equal to one thousand, represented as 100×10^{1}. In such cases, the new "1" is the leading digit position of the returned result.

Besides a logical exact result, each arithmetic operation has a preferred scale for representing a result. The preferred scale for each operation is listed in the table below.

Preferred Scales for Results of Arithmetic Operations

Operation	Preferred Scale of Result
Add	max(addend.scale(), augend.scale())
Subtract	max(minuend.scale(), subtrahend.scale())
Multiply	multiplier.scale() + multiplicand.scale()
Divide	dividend.scale() - divisor.scale()

These scales are the ones used by the methods which return exact arithmetic results; except that an exact divide may have to use a larger scale since the exact result may have more digits. For example, `1/32` is `0.03125`.

Before rounding, the scale of the logical exact intermediate result is the preferred scale for that operation. If the exact numerical result cannot be represented in `precision` digits, rounding selects the set of digits to return and the scale of the result is reduced from the scale of the intermediate result to the least scale which can represent the `precision` digits actually returned. If the exact result can be represented with at most `precision` digits, the representation of the result with the scale closest to the preferred scale is returned. In particular, an exactly representable quotient may be represented in fewer than `precision` digits by removing trailing zeros and decreasing the scale. For example, rounding to three digits using the floor rounding mode,

```
19/100 = 0.19 // integer=19, scale=2
```

but

```
21/110 = 0.190 // integer=190, scale=3
```

Note that for add, subtract, and multiply, the reduction in scale will equal the number of digit positions of the exact result which are discarded. If the rounding causes a carry propagation to create a new high-order digit position, an additional digit of the result is discarded than when no new digit position is created.

Other methods may have slightly different rounding semantics. For example, the result of the `pow` method using the specified algorithm can occasionally differ from the rounded mathematical result by more than one unit in the last place, one *ulp*.

Two types of operations are provided for manipulating the scale of a `BigDecimal`: scaling/rounding operations and decimal point motion operations. Scaling/rounding operations (`setScale` and `round`) return a `BigDecimal` whose value is approximately (or exactly) equal to that of the operand, but whose scale or precision is the specified value; that is, they increase or decrease the precision of the stored number with minimal effect on its value. Decimal point motion operations (`movePointLeft` and `movePointRight`) return a `BigDecimal` created from the operand by moving the decimal point a specified distance in the specified direction.

For the sake of brevity and clarity, pseudo-code is used throughout the descriptions of `BigDecimal` methods. The pseudo-code expression `(i + j)` is shorthand for "a `BigDecimal` whose value is that of the `BigDecimal` `i` added to that of the `BigDecimal` `j`." The pseudo-code expression `(i == j)` is shorthand for "`true` if and only if the `BigDecimal` `i` represents the same value as the `BigDecimal` `j`." Other pseudo-code expressions are interpreted similarly. Square brackets are used to represent the particular `BigInteger` and scale pair defining a `BigDecimal` value; for example [19, 2] is the `BigDecimal` numerically equal to 0.19 having a scale of 2.

Note: care should be exercised if `BigDecimal` objects are used as keys in a `SortedMap` or elements in a `SortedSet` since `BigDecimal`'s *natural ordering* is *inconsistent with equals*. See `Comparable`, `SortedMap` or `SortedSet` for more information.

All methods and constructors for this class throw `NullPointerException` when passed a `null` object reference for any input parameter.

See Also:

`BigInteger`, `MathContext`, `RoundingMode`, `SortedMap`, `SortedSet`, Serialized Form

Field Summary

Fields

Modifier and Type	Field and Description
`static BigDecimal`	**`ONE`** The value 1, with a scale of 0.
`static int`	**`ROUND_CEILING`** Rounding mode to round towards positive infinity.
`static int`	**`ROUND_DOWN`**

	Rounding mode to round towards zero.
static int	**ROUND_FLOOR** Rounding mode to round towards negative infinity.
static int	**ROUND_HALF_DOWN** Rounding mode to round towards "nearest neighbor" unless both neighbors are equidistant, in which case round down.
static int	**ROUND_HALF_EVEN** Rounding mode to round towards the "nearest neighbor" unless both neighbors are equidistant, in which case, round towards the even neighbor.
static int	**ROUND_HALF_UP** Rounding mode to round towards "nearest neighbor" unless both neighbors are equidistant, in which case round up.
static int	**ROUND_UNNECESSARY** Rounding mode to assert that the requested operation has an exact result, hence no rounding is necessary.
static int	**ROUND_UP** Rounding mode to round away from zero.
static BigDecimal	**TEN** The value 10, with a scale of 0.
static BigDecimal	**ZERO** The value 0, with a scale of 0.

Constructor Summary

Constructors

Constructor and Description
BigDecimal(**BigInteger** val) Translates a BigInteger into a BigDecimal.
BigDecimal(**BigInteger** unscaledVal, int scale) Translates a BigInteger unscaled value and an int scale into a BigDecimal.
BigDecimal(**BigInteger** unscaledVal, int scale, **MathContext** mc) Translates a BigInteger unscaled value and an int scale into a BigDecimal, with rounding according to the context settings.
BigDecimal(**BigInteger** val, **MathContext** mc) Translates a BigInteger into a BigDecimal rounding according to the context settings.
BigDecimal(char[] in) Translates a character array representation of a BigDecimal into a BigDecimal, accepting the same sequence of characters as the **BigDecimal(String)** constructor.
BigDecimal(char[] in, int offset, int len) Translates a character array representation of a BigDecimal into a BigDecimal, accepting the same sequence of characters as the **BigDecimal(String)** constructor, while allowing a sub-array to be specified.
BigDecimal(char[] in, int offset, int len, **MathContext** mc) Translates a character array representation of a BigDecimal into a BigDecimal, accepting the same sequence of characters as the **BigDecimal(String)** constructor, while allowing a sub-array to be specified and with rounding according to the context settings.
BigDecimal(char[] in, **MathContext** mc) Translates a character array representation of a BigDecimal into a BigDecimal, accepting the same sequence of characters as the **BigDecimal(String)** constructor and with rounding according to the context settings.
BigDecimal(double val) Translates a double into a BigDecimal which is the exact decimal representation of the double's binary floating-point value.
BigDecimal(double val, **MathContext** mc) Translates a double into a BigDecimal, with rounding according to the context settings.

BigDecimal(int val)
Translates an int into a BigDecimal.

BigDecimal(int val, **MathContext** mc)
Translates an int into a BigDecimal, with rounding according to the context settings.

BigDecimal(long val)
Translates a long into a BigDecimal.

BigDecimal(long val, **MathContext** mc)
Translates a long into a BigDecimal, with rounding according to the context settings.

BigDecimal(**String** val)
Translates the string representation of a BigDecimal into a BigDecimal.

BigDecimal(**String** val, **MathContext** mc)
Translates the string representation of a BigDecimal into a BigDecimal, accepting the same strings as the **BigDecimal(String)** constructor, with rounding according to the context settings.

Method Summary

Methods

Modifier and Type	Method and Description
BigDecimal	**abs**() Returns a BigDecimal whose value is the absolute value of this BigDecimal, and whose scale is this.scale().
BigDecimal	**abs**(**MathContext** mc) Returns a BigDecimal whose value is the absolute value of this BigDecimal, with rounding according to the context settings.
BigDecimal	**add**(**BigDecimal** augend) Returns a BigDecimal whose value is (this + augend), and whose scale is max(this.scale(), augend.scale()).
BigDecimal	**add**(**BigDecimal** augend, **MathContext** mc) Returns a BigDecimal whose value is (this + augend), with rounding according to the context settings.
byte	**byteValueExact**() Converts this BigDecimal to a byte, checking for lost information.
int	**compareTo**(**BigDecimal** val) Compares this BigDecimal with the specified BigDecimal.
BigDecimal	**divide**(**BigDecimal** divisor) Returns a BigDecimal whose value is (this / divisor), and whose preferred scale is (this.scale() - divisor.scale()); if the exact quotient cannot be represented (because it has a non-terminating decimal expansion) an ArithmeticException is thrown.
BigDecimal	**divide**(**BigDecimal** divisor, int roundingMode) Returns a BigDecimal whose value is (this / divisor), and whose scale is this.scale().
BigDecimal	**divide**(**BigDecimal** divisor, int scale, int roundingMode) Returns a BigDecimal whose value is (this / divisor), and whose scale is as specified.
BigDecimal	**divide**(**BigDecimal** divisor, int scale, **RoundingMode** roundingMode) Returns a BigDecimal whose value is (this / divisor), and whose scale is as specified.
BigDecimal	**divide**(**BigDecimal** divisor, **MathContext** mc) Returns a BigDecimal whose value is (this / divisor), with rounding according to the context settings.
BigDecimal	**divide**(**BigDecimal** divisor, **RoundingMode** roundingMode)

	Returns a BigDecimal whose value is (this / divisor), and whose scale is this.scale().
BigDecimal[]	**divideAndRemainder**(**BigDecimal** divisor) Returns a two-element BigDecimal array containing the result of divideToIntegralValue followed by the result of remainder on the two operands.
BigDecimal[]	**divideAndRemainder**(**BigDecimal** divisor, **MathContext** mc) Returns a two-element BigDecimal array containing the result of divideToIntegralValue followed by the result of remainder on the two operands calculated with rounding according to the context settings.
BigDecimal	**divideToIntegralValue**(**BigDecimal** divisor) Returns a BigDecimal whose value is the integer part of the quotient (this / divisor) rounded down.
BigDecimal	**divideToIntegralValue**(**BigDecimal** divisor, **MathContext** mc) Returns a BigDecimal whose value is the integer part of (this / divisor).
double	**doubleValue**() Converts this BigDecimal to a double.
boolean	**equals**(**Object** x) Compares this BigDecimal with the specified Object for equality.
float	**floatValue**() Converts this BigDecimal to a float.
int	**hashCode**() Returns the hash code for this BigDecimal.
int	**intValue**() Converts this BigDecimal to an int.
int	**intValueExact**() Converts this BigDecimal to an int, checking for lost information.
long	**longValue**() Converts this BigDecimal to a long.
long	**longValueExact**() Converts this BigDecimal to a long, checking for lost information.
BigDecimal	**max**(**BigDecimal** val) Returns the maximum of this BigDecimal and val.
BigDecimal	**min**(**BigDecimal** val) Returns the minimum of this BigDecimal and val.
BigDecimal	**movePointLeft**(int n) Returns a BigDecimal which is equivalent to this one with the decimal point moved n places to the left.
BigDecimal	**movePointRight**(int n) Returns a BigDecimal which is equivalent to this one with the decimal point moved n places to the right.
BigDecimal	**multiply**(**BigDecimal** multiplicand) Returns a BigDecimal whose value is (this × multiplicand), and whose scale is (this.scale() + multiplicand.scale()).
BigDecimal	**multiply**(**BigDecimal** multiplicand, **MathContext** mc) Returns a BigDecimal whose value is (this × multiplicand), with rounding according to the context settings.
BigDecimal	**negate**() Returns a BigDecimal whose value is (-this), and whose scale is this.scale().
BigDecimal	**negate**(**MathContext** mc) Returns a BigDecimal whose value is (-this), with rounding according to the context settings.
BigDecimal	**plus**() Returns a BigDecimal whose value is (+this), and whose scale is this.scale().
BigDecimal	**plus**(**MathContext** mc)

	Returns a BigDecimal whose value is (+this), with rounding according to the context settings.
BigDecimal	**pow**(int n) Returns a BigDecimal whose value is ($this^n$), The power is computed exactly, to unlimited precision.
BigDecimal	**pow**(int n, **MathContext** mc) Returns a BigDecimal whose value is ($this^n$).
int	**precision**() Returns the *precision* of this BigDecimal.
BigDecimal	**remainder**(**BigDecimal** divisor) Returns a BigDecimal whose value is (this % divisor).
BigDecimal	**remainder**(**BigDecimal** divisor, **MathContext** mc) Returns a BigDecimal whose value is (this % divisor), with rounding according to the context settings.
BigDecimal	**round**(**MathContext** mc) Returns a BigDecimal rounded according to the MathContext settings.
int	**scale**() Returns the *scale* of this BigDecimal.
BigDecimal	**scaleByPowerOfTen**(int n) Returns a BigDecimal whose numerical value is equal to (this * 10^n).
BigDecimal	**setScale**(int newScale) Returns a BigDecimal whose scale is the specified value, and whose value is numerically equal to this BigDecimal's.
BigDecimal	**setScale**(int newScale, int roundingMode) Returns a BigDecimal whose scale is the specified value, and whose unscaled value is determined by multiplying or dividing this BigDecimal's unscaled value by the appropriate power of ten to maintain its overall value.
BigDecimal	**setScale**(int newScale, **RoundingMode** roundingMode) Returns a BigDecimal whose scale is the specified value, and whose unscaled value is determined by multiplying or dividing this BigDecimal's unscaled value by the appropriate power of ten to maintain its overall value.
short	**shortValueExact**() Converts this BigDecimal to a short, checking for lost information.
int	**signum**() Returns the signum function of this BigDecimal.
BigDecimal	**stripTrailingZeros**() Returns a BigDecimal which is numerically equal to this one but with any trailing zeros removed from the representation.
BigDecimal	**subtract**(**BigDecimal** subtrahend) Returns a BigDecimal whose value is (this - subtrahend), and whose scale is max(this.scale(), subtrahend.scale()).
BigDecimal	**subtract**(**BigDecimal** subtrahend, **MathContext** mc) Returns a BigDecimal whose value is (this - subtrahend), with rounding according to the context settings.
BigInteger	**toBigInteger**() Converts this BigDecimal to a BigInteger.
BigInteger	**toBigIntegerExact**() Converts this BigDecimal to a BigInteger, checking for lost information.
String	**toEngineeringString**() Returns a string representation of this BigDecimal, using engineering notation if an exponent is needed.
String	**toPlainString**() Returns a string representation of this BigDecimal without an exponent field.
String	**toString**()

	Returns the string representation of this `BigDecimal`, using scientific notation if an exponent is needed.
BigDecimal	**ulp**() Returns the size of an ulp, a unit in the last place, of this `BigDecimal`.
BigInteger	**unscaledValue**() Returns a `BigInteger` whose value is the *unscaled value* of this `BigDecimal`.
static **BigDecimal**	**valueOf**(double val) Translates a `double` into a `BigDecimal`, using the `double`'s canonical string representation provided by the **Double.toString(double)** method.
static **BigDecimal**	**valueOf**(long val) Translates a `long` value into a `BigDecimal` with a scale of zero.
static **BigDecimal**	**valueOf**(long unscaledVal, int scale) Translates a `long` unscaled value and an `int` scale into a `BigDecimal`.

Methods inherited from class java.lang.Number

byteValue, shortValue

Methods inherited from class java.lang.Object

clone, finalize, getClass, notify, notifyAll, wait, wait, wait

Field Detail

ZERO

```
public static final BigDecimal ZERO
```

The value 0, with a scale of 0.

Since:

1.5

ONE

```
public static final BigDecimal ONE
```

The value 1, with a scale of 0.

Since:

1.5

TEN

```
public static final BigDecimal TEN
```

The value 10, with a scale of 0.

Since:

1.5