Module java.base
Package 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 the scale is zero or positive, the scale is the number of digits to the right of the decimal point. If the scale is 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 × 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 `ArithmeticException` 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 deprecated; 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. 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. 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. The different representations of the same numerical value are called members of the same *cohort*. The natural order of `BigDecimal` considers members of the same cohort to be equal to each other. In contrast, the `equals` method requires both the numerical value and representation to be the same for equality to hold. The results of methods like `scale` and `unscaledValue()` will differ for numerically equal values with different representations.

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 and square root) 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.

For methods and constructors with a `MathContext` parameter, if the result is inexact but the rounding mode is `UNNECESSARY`, an `ArithmeticException` will be thrown.

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())