NSDecimalNumberBehaviors

Adopted By: NSDecimalNumberHandler

Declared In: Foundation/NSDecimalNumber.h

Protocol Description

The NSDecimalBehaviors protocol declares three methods that control the discretionary aspects of working with NSDecimalNumbers. The **scale** and **roundingMode** methods determine the precision of NSDecimalNumber's return values, and the way in which those values should be rounded to fit that precision. The **exceptionDuringOperation:error:leftOperand:rightOperand:** determines the way in which an NSDecimalNumber should handle different calculation errors.

For an example of a class that adopts the NSDecimalBehaviors protocol, see the specification for NSDecimalNumberHandler.

Method Types

Rounding	– roundingMode – scale
Handling errors	– exceptionDuringOperation:error:leftOperand:rightOperand:

Instance Methods

exceptionDuringOperation:error:leftOperand:rightOperand:

– (NSDecimalNumber *)**exceptionDuringOperation:**(SEL)*method* **error:**(NSCalculationError)*error*
leftOperand:(NSDecimalNumber *)*leftOperand*
rightOperand:(NSDecimalNumber *)*rightOperand*

Specifies what an NSDecimalNumber will do when, in the course of applying *method* to *leftOperand* and *rightOperand*, it encounters *error.*

There are four possible values for *error.* The first three have to do with limits on NSDecimalNumber's ability to represent decimal numbers. An NSDecimalNumber can represent any number that can be expressed as *mantissa* **x** 10^{exponent}, where *mantissa* is a decimal integer up to 38 digits long, and *exponent* is between -256 and 256. If these limits are exceeded, the NSDecimalNumber returns one of the following errors.

- NSCalculationLossOfPrecision. The number can't be represented in 38 significant digits.
- NSCalculationOverflow. The number is too large to represent.
- NSCalculationUnderflow. The number is too small to represent.

The last error is simpler:

- NSCalculationDivideByZero. The caller tried to divide by zero.

In implementing **exceptionDuringOperation:error:leftOperand:rightOperand**, you can handle each of these errors in several ways:

- Raise an exception. For an explantion of exceptions, see the NSException class description in the *Foundation Framework Reference*.
- Return **nil**. The calling *method* will return its value as though no error had occurred. If *error* is NSCalculationLossOfPrecision, *method* will return an imprecise value—that is, one constrained to 38 significant digits. If *error* is NSCalculationUnderflow or NSCalculationOverflow, *method* will return NSDecimalNumber's **notANumber**. You shouldn't return **nil** if *error* is NSDivideByZero.
- Correct the error and return a valid NSDecimalNumber. The calling method will use this as its own return value.

roundingMode

– (NSRoundingMode)**roundingMode**

Returns the way that NSDecimalNumber's **decimalNumberBy...** methods round their return values. There are four possible NSRoundingModes:

- NSRoundDown. The methods round their return values down.
- NSRoundUp. The methods round their return values up.
- NSRoundPlain. The methods round to the closest possible return value. When they are caught halfway between two positive numbers, they round up; when caught between two negative numbers, they round down.
- NSRoundBankers. The methods round to the closest possible return value. When they are caught halfway between two possibilities, they return the possibility whose last digit is even. In practice, this means that, over the long run, numbers will be rounded up as often as they are rounded down; there will be no systematic bias.

The rounding mode only matters if the **scale** method sets a limit on the precision of NSDecimalNumber return values. It has no effect if **scale** returns NSDecimalNoScale.

Assuming that **scale** returns 1, the NSRoundingMode has the following effects on various original values:

Original value	NSRoundDown	NSRoundUp	NSRoundPlain	NSRoundBankers
1.24	1.2	1.3	1.2	1.2
1.26	1.2	1.3	1.3	1.3
1.25	1.2	1.3	1.3	1.2
1.35	1.3	1.4	1.4	1.4
-1.35	-1.4	-1.3	-1.4	-1.4

scale

– (short)**scale**

Limits the precision of the values returned by NSDecimalNumber's **decimalNumberBy...** methods.

Specifically, **scale** returns the number of digits allowed after the decimal separator. If **scale** returns a negative value, it affects the digits *before* the decimal separator as well. If **scale** returns NSDecimalNoScale, the number of digits is unlimited.

Assuming that **roundingMode** returns NSRoundPlain, different values of **scale** have the following effects on the number 123.456:

Scale	Return value
NSDecimalNoScale	123.456
2	123.45
0	123
-2	100