compact1, compact2, compact3
java.lang

Class Math

java.lang.Object
    java.lang.Math

```
public final class Math
extends Object
```

The class `Math` contains methods for performing basic numeric operations such as the elementary exponential, logarithm, square root, and trigonometric functions.

Unlike some of the numeric methods of class `StrictMath`, all implementations of the equivalent functions of class `Math` are not defined to return the bit-for-bit same results. This relaxation permits better-performing implementations where strict reproducibility is not required.

By default many of the `Math` methods simply call the equivalent method in `StrictMath` for their implementation. Code generators are encouraged to use platform-specific native libraries or microprocessor instructions, where available, to provide higher-performance implementations of `Math` methods. Such higher-performance implementations still must conform to the specification for `Math`.

The quality of implementation specifications concern two properties, accuracy of the returned result and monotonicity of the method. Accuracy of the floating-point `Math` methods is measured in terms of *ulps*, units in the last place. For a given floating-point format, an ulp of a specific real number value is the distance between the two floating-point values bracketing that numerical value. When discussing the accuracy of a method as a whole rather than at a specific argument, the number of ulps cited is for the worst-case error at any argument. If a method always has an error less than 0.5 ulps, the method always returns the floating-point number nearest the exact result; such a method is *correctly rounded*. A correctly rounded method is generally the best a floating-point approximation can be; however, it is impractical for many floating-point methods to be correctly rounded. Instead, for the `Math` class, a larger error bound of 1 or 2 ulps is allowed for certain methods. Informally, with a 1 ulp error bound, when the exact result is a representable number, the exact result should be returned as the computed result; otherwise, either of the two floating-point values which bracket the exact result may be returned. For exact results large in magnitude, one of the endpoints of the bracket may be infinite. Besides accuracy at individual arguments, maintaining proper relations between the method at different arguments is also important. Therefore, most methods with more than 0.5 ulp errors are required to be *semi-monotonic*: whenever the mathematical function is non-decreasing, so is the floating-point approximation, likewise, whenever the mathematical function is non-increasing, so is the floating-point approximation. Not all approximations that have 1 ulp accuracy will automatically meet the monotonicity requirements.

The platform uses signed two's complement integer arithmetic with int and long primitive types. The developer should choose the primitive type to ensure that arithmetic operations consistently produce correct results, which in some cases means the operations will not overflow the range of values of the computation. The best practice is to choose the primitive type and algorithm to avoid overflow. In cases where the size is `int` or `long` and overflow errors need to be detected, the methods `addExact`, `subtractExact`, `multiplyExact`, and `toIntExact` throw an `ArithmeticException` when the results overflow. For other arithmetic operations such as divide, absolute value, increment, decrement, and negation overflow occurs only with a specific minimum or maximum value and should be checked against the minimum or maximum as appropriate.

Since:
`JDK1.0`

Field Summary

Fields

Modifier and Type	Field and Description
`static double`	**`E`** The `double` value that is closer than any other to *e*, the base of the natural logarithms.
`static double`	**`PI`** The `double` value that is closer than any other to *pi*, the ratio of the circumference of a circle to its diameter.

Method Summary

All Methods | **Static Methods** | **Concrete Methods**

Modifier and Type	Method and Description
`static double`	**`abs`**`(double a)` Returns the absolute value of a `double` value.
`static float`	**`abs`**`(float a)` Returns the absolute value of a `float` value.
`static int`	**`abs`**`(int a)` Returns the absolute value of an `int` value.
`static long`	**`abs`**`(long a)` Returns the absolute value of a `long` value.
`static double`	**`acos`**`(double a)` Returns the arc cosine of a value; the returned angle is in the range 0.0 through *pi*.
`static int`	**`addExact`**`(int x, int y)` Returns the sum of its arguments, throwing an exception if the result overflows an `int`.
`static long`	**`addExact`**`(long x, long y)` Returns the sum of its arguments, throwing an exception if the result overflows a `long`.
`static double`	**`asin`**`(double a)` Returns the arc sine of a value; the returned angle is in the range -*pi*/2 through *pi*/2.
`static double`	**`atan`**`(double a)` Returns the arc tangent of a value; the returned angle is in the range -*pi*/2 through *pi*/2.
`static double`	**`atan2`**`(double y, double x)` Returns the angle *theta* from the conversion of rectangular coordinates (x, y) to polar coordinates (r, *theta*).
`static double`	**`cbrt`**`(double a)`

	Returns the cube root of a `double` value.
`static double`	**`ceil`**`(double a)` Returns the smallest (closest to negative infinity) `double` value that is greater than or equal to the argument and is equal to a mathematical integer.
`static double`	**`copySign`**`(double magnitude, double sign)` Returns the first floating-point argument with the sign of the second floating-point argument.
`static float`	**`copySign`**`(float magnitude, float sign)` Returns the first floating-point argument with the sign of the second floating-point argument.
`static double`	**`cos`**`(double a)` Returns the trigonometric cosine of an angle.
`static double`	**`cosh`**`(double x)` Returns the hyperbolic cosine of a `double` value.
`static int`	**`decrementExact`**`(int a)` Returns the argument decremented by one, throwing an exception if the result overflows an `int`.
`static long`	**`decrementExact`**`(long a)` Returns the argument decremented by one, throwing an exception if the result overflows a `long`.
`static double`	**`exp`**`(double a)` Returns Euler's number e raised to the power of a `double` value.
`static double`	**`expm1`**`(double x)` Returns e^x -1.
`static double`	**`floor`**`(double a)` Returns the largest (closest to positive infinity) `double` value that is less than or equal to the argument and is equal to a mathematical integer.
`static int`	**`floorDiv`**`(int x, int y)` Returns the largest (closest to positive infinity) `int` value that is less than or equal to the algebraic quotient.
`static long`	**`floorDiv`**`(long x, long y)` Returns the largest (closest to positive infinity) `long` value that is less than or equal to the algebraic quotient.
`static int`	**`floorMod`**`(int x, int y)` Returns the floor modulus of the `int` arguments.
`static long`	**`floorMod`**`(long x, long y)` Returns the floor modulus of the `long` arguments.
`static int`	**`getExponent`**`(double d)` Returns the unbiased exponent used in the representation of a `double`.
`static int`	**`getExponent`**`(float f)` Returns the unbiased exponent used in the representation of a `float`.
`static double`	**`hypot`**`(double x, double y)` Returns sqrt(x^2 +y^2) without intermediate overflow or underflow.
`static double`	**`IEEEremainder`**`(double f1, double f2)` Computes the remainder operation on two arguments as prescribed by the IEEE 754 standard.
`static int`	**`incrementExact`**`(int a)` Returns the argument incremented by one, throwing an exception if the result overflows an `int`.
`static long`	**`incrementExact`**`(long a)` Returns the argument incremented by one, throwing an exception if the result overflows a `long`.
`static double`	**`log`**`(double a)` Returns the natural logarithm (base e) of a `double` value.
`static double`	**`log10`**`(double a)` Returns the base 10 logarithm of a `double` value.
`static double`	**`log1p`**`(double x)` Returns the natural logarithm of the sum of the argument and 1.
`static double`	**`max`**`(double a, double b)` Returns the greater of two `double` values.
`static float`	**`max`**`(float a, float b)` Returns the greater of two `float` values.
`static int`	**`max`**`(int a, int b)` Returns the greater of two `int` values.
`static long`	**`max`**`(long a, long b)` Returns the greater of two `long` values.
`static double`	**`min`**`(double a, double b)` Returns the smaller of two `double` values.
`static float`	**`min`**`(float a, float b)` Returns the smaller of two `float` values.
`static int`	**`min`**`(int a, int b)` Returns the smaller of two `int` values.
`static long`	**`min`**`(long a, long b)` Returns the smaller of two `long` values.
`static int`	**`multiplyExact`**`(int x, int y)` Returns the product of the arguments, throwing an exception if the result overflows an `int`.

static long	**multiplyExact**(long x, long y) Returns the product of the arguments, throwing an exception if the result overflows a long.
static int	**negateExact**(int a) Returns the negation of the argument, throwing an exception if the result overflows an int.
static long	**negateExact**(long a) Returns the negation of the argument, throwing an exception if the result overflows a long.
static double	**nextAfter**(double start, double direction) Returns the floating-point number adjacent to the first argument in the direction of the second argument.
static float	**nextAfter**(float start, double direction) Returns the floating-point number adjacent to the first argument in the direction of the second argument.
static double	**nextDown**(double d) Returns the floating-point value adjacent to d in the direction of negative infinity.
static float	**nextDown**(float f) Returns the floating-point value adjacent to f in the direction of negative infinity.
static double	**nextUp**(double d) Returns the floating-point value adjacent to d in the direction of positive infinity.
static float	**nextUp**(float f) Returns the floating-point value adjacent to f in the direction of positive infinity.
static double	**pow**(double a, double b) Returns the value of the first argument raised to the power of the second argument.
static double	**random**() Returns a double value with a positive sign, greater than or equal to 0.0 and less than 1.0.
static double	**rint**(double a) Returns the double value that is closest in value to the argument and is equal to a mathematical integer.
static long	**round**(double a) Returns the closest long to the argument, with ties rounding to positive infinity.
static int	**round**(float a) Returns the closest int to the argument, with ties rounding to positive infinity.
static double	**scalb**(double d, int scaleFactor) Returns $d \times 2^{scaleFactor}$ rounded as if performed by a single correctly rounded floating-point multiply to a member of the double value set.
static float	**scalb**(float f, int scaleFactor) Returns $f \times 2^{scaleFactor}$ rounded as if performed by a single correctly rounded floating-point multiply to a member of the float value set.
static double	**signum**(double d) Returns the signum function of the argument; zero if the argument is zero, 1.0 if the argument is greater than zero, -1.0 if the argument is less than zero.
static float	**signum**(float f) Returns the signum function of the argument; zero if the argument is zero, 1.0f if the argument is greater than zero, -1.0f if the argument is less than zero.
static double	**sin**(double a) Returns the trigonometric sine of an angle.
static double	**sinh**(double x) Returns the hyperbolic sine of a double value.
static double	**sqrt**(double a) Returns the correctly rounded positive square root of a double value.
static int	**subtractExact**(int x, int y) Returns the difference of the arguments, throwing an exception if the result overflows an int.
static long	**subtractExact**(long x, long y) Returns the difference of the arguments, throwing an exception if the result overflows a long.
static double	**tan**(double a) Returns the trigonometric tangent of an angle.
static double	**tanh**(double x) Returns the hyperbolic tangent of a double value.
static double	**toDegrees**(double angrad) Converts an angle measured in radians to an approximately equivalent angle measured in degrees.
static int	**toIntExact**(long value) Returns the value of the long argument; throwing an exception if the value overflows an int.
static double	**toRadians**(double angdeg) Converts an angle measured in degrees to an approximately equivalent angle measured in radians.
static double	**ulp**(double d) Returns the size of an ulp of the argument.
static float	**ulp**(float f) Returns the size of an ulp of the argument.

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

E

```
public static final double E
```

The double value that is closer than any other to *e*, the base of the natural logarithms.

See Also:
Constant Field Values

PI

```
public static final double PI
```

The double value that is closer than any other to *pi*, the ratio of the circumference of a circle to its diameter.

See Also:
Constant Field Values

Method Detail

sin

```
public static double sin(double a)
```

Returns the trigonometric sine of an angle. Special cases:
- If the argument is NaN or an infinity, then the result is NaN.
- If the argument is zero, then the result is a zero with the same sign as the argument.

The computed result must be within 1 ulp of the exact result. Results must be semi-monotonic.

Parameters:
a - an angle, in radians.

Returns:
the sine of the argument.

cos

```
public static double cos(double a)
```

Returns the trigonometric cosine of an angle. Special cases:
- If the argument is NaN or an infinity, then the result is NaN.

The computed result must be within 1 ulp of the exact result. Results must be semi-monotonic.

Parameters:
a - an angle, in radians.

Returns:
the cosine of the argument.

tan

```
public static double tan(double a)
```

Returns the trigonometric tangent of an angle. Special cases:
- If the argument is NaN or an infinity, then the result is NaN.
- If the argument is zero, then the result is a zero with the same sign as the argument.

The computed result must be within 1 ulp of the exact result. Results must be semi-monotonic.

Parameters:
a - an angle, in radians.

Returns:
the tangent of the argument.

asin

```
public static double asin(double a)
```

Returns the arc sine of a value; the returned angle is in the range -*pi*/2 through *pi*/2. Special cases:
- If the argument is NaN or its absolute value is greater than 1, then the result is NaN.
- If the argument is zero, then the result is a zero with the same sign as the argument.

The computed result must be within 1 ulp of the exact result. Results must be semi-monotonic.

Parameters:
a - the value whose arc sine is to be returned.

Returns:
the arc sine of the argument.

acos

`public static double acos(double a)`

Returns the arc cosine of a value; the returned angle is in the range 0.0 through *pi*. Special case:
- If the argument is NaN or its absolute value is greater than 1, then the result is NaN.

The computed result must be within 1 ulp of the exact result. Results must be semi-monotonic.

Parameters:
`a - the value whose arc cosine is to be returned.`

Returns:
`the arc cosine of the argument.`

atan

`public static double atan(double a)`

Returns the arc tangent of a value; the returned angle is in the range -*pi*/2 through *pi*/2. Special cases:
- If the argument is NaN, then the result is NaN.
- If the argument is zero, then the result is a zero with the same sign as the argument.

The computed result must be within 1 ulp of the exact result. Results must be semi-monotonic.

Parameters:
`a - the value whose arc tangent is to be returned.`

Returns:
`the arc tangent of the argument.`

toRadians

`public static double toRadians(double angdeg)`

Converts an angle measured in degrees to an approximately equivalent angle measured in radians. The conversion from degrees to radians is generally inexact.

Parameters:
`angdeg - an angle, in degrees`

Returns:
`the measurement of the angle angdeg in radians.`

Since:
`1.2`

toDegrees

`public static double toDegrees(double angrad)`

Converts an angle measured in radians to an approximately equivalent angle measured in degrees. The conversion from radians to degrees is generally inexact; users should *not* expect `cos(toRadians(90.0))` to exactly equal `0.0`.

Parameters:
`angrad - an angle, in radians`

Returns:
`the measurement of the angle angrad in degrees.`

Since:
`1.2`

exp

`public static double exp(double a)`

Returns Euler's number *e* raised to the power of a `double` value. Special cases:
- If the argument is NaN, the result is NaN.
- If the argument is positive infinity, then the result is positive infinity.
- If the argument is negative infinity, then the result is positive zero.

The computed result must be within 1 ulp of the exact result. Results must be semi-monotonic.

Parameters:
`a - the exponent to raise e to.`

Returns:
the value e^{a}, where *e* is the base of the natural logarithms.

log

`public static double log(double a)`

Returns the natural logarithm (base *e*) of a `double` value. Special cases:
- If the argument is NaN or less than zero, then the result is NaN.
- If the argument is positive infinity, then the result is positive infinity.
- If the argument is positive zero or negative zero, then the result is negative infinity.

The computed result must be within 1 ulp of the exact result. Results must be semi-monotonic.

Parameters:
`a - a value`

Returns:
`the value ln a, the natural logarithm of a.`

log10

```
public static double log10(double a)
```

Returns the base 10 logarithm of a `double` value. Special cases:
- If the argument is NaN or less than zero, then the result is NaN.
- If the argument is positive infinity, then the result is positive infinity.
- If the argument is positive zero or negative zero, then the result is negative infinity.
- If the argument is equal to 10^n for integer n, then the result is n.

The computed result must be within 1 ulp of the exact result. Results must be semi-monotonic.

Parameters:

`a - a value`

Returns:

`the base 10 logarithm of a.`

Since:

`1.5`

sqrt

```
public static double sqrt(double a)
```

Returns the correctly rounded positive square root of a `double` value. Special cases:
- If the argument is NaN or less than zero, then the result is NaN.
- If the argument is positive infinity, then the result is positive infinity.
- If the argument is positive zero or negative zero, then the result is the same as the argument.

Otherwise, the result is the `double` value closest to the true mathematical square root of the argument value.

Parameters:

`a - a value.`

Returns:

`the positive square root of a. If the argument is NaN or less than zero, the result is NaN.`

cbrt

```
public static double cbrt(double a)
```

Returns the cube root of a `double` value. For positive finite `x`, `cbrt(-x) == -cbrt(x)`; that is, the cube root of a negative value is the negative of the cube root of that value's magnitude. Special cases:
- If the argument is NaN, then the result is NaN.
- If the argument is infinite, then the result is an infinity with the same sign as the argument.
- If the argument is zero, then the result is a zero with the same sign as the argument.

The computed result must be within 1 ulp of the exact result.

Parameters:

`a - a value.`

Returns:

`the cube root of a.`

Since:

`1.5`

IEEEremainder

```
public static double IEEEremainder(double f1,
                                   double f2)
```

Computes the remainder operation on two arguments as prescribed by the IEEE 754 standard. The remainder value is mathematically equal to `f1 - f2` × n, where n is the mathematical integer closest to the exact mathematical value of the quotient `f1/f2`, and if two mathematical integers are equally close to `f1/f2`, then n is the integer that is even. If the remainder is zero, its sign is the same as the sign of the first argument. Special cases:
- If either argument is NaN, or the first argument is infinite, or the second argument is positive zero or negative zero, then the result is NaN.
- If the first argument is finite and the second argument is infinite, then the result is the same as the first argument.

Parameters:

`f1 - the dividend.`

`f2 - the divisor.`

Returns:

`the remainder when f1 is divided by f2.`

ceil

```
public static double ceil(double a)
```

Returns the smallest (closest to negative infinity) `double` value that is greater than or equal to the argument and is equal to a mathematical integer. Special cases:
- If the argument value is already equal to a mathematical integer, then the result is the same as the argument.
- If the argument is NaN or an infinity or positive zero or negative zero, then the result is the same as the argument.
- If the argument value is less than zero but greater than -1.0, then the result is negative zero.

Note that the value of `Math.ceil(x)` is exactly the value of `-Math.floor(-x)`.

Parameters:

`a - a value.`

Returns:

`the smallest (closest to negative infinity) floating-point value that is greater than or equal to the argument and is equal to a mathematical integer.`

floor

```
public static double floor(double a)
```

Returns the largest (closest to positive infinity) `double` value that is less than or equal to the argument and is equal to a mathematical integer. Special cases:

- If the argument value is already equal to a mathematical integer, then the result is the same as the argument.
- If the argument is NaN or an infinity or positive zero or negative zero, then the result is the same as the argument.

Parameters:

`a` - a value.

Returns:

the largest (closest to positive infinity) floating-point value that less than or equal to the argument and is equal to a mathematical integer.

rint

```
public static double rint(double a)
```

Returns the `double` value that is closest in value to the argument and is equal to a mathematical integer. If two `double` values that are mathematical integers are equally close, the result is the integer value that is even. Special cases:

- If the argument value is already equal to a mathematical integer, then the result is the same as the argument.
- If the argument is NaN or an infinity or positive zero or negative zero, then the result is the same as the argument.

Parameters:

`a` - a double value.

Returns:

the closest floating-point value to a that is equal to a mathematical integer.

atan2

```
public static double atan2(double y,
                           double x)
```

Returns the angle *theta* from the conversion of rectangular coordinates (x, y) to polar coordinates (r, *theta*). This method computes the phase *theta* by computing an arc tangent of y/x in the range of -*pi* to *pi*. Special cases:

- If either argument is NaN, then the result is NaN.
- If the first argument is positive zero and the second argument is positive, or the first argument is positive and finite and the second argument is positive infinity, then the result is positive zero.
- If the first argument is negative zero and the second argument is positive, or the first argument is negative and finite and the second argument is positive infinity, then the result is negative zero.
- If the first argument is positive zero and the second argument is negative, or the first argument is positive and finite and the second argument is negative infinity, then the result is the `double` value closest to *pi*.
- If the first argument is negative zero and the second argument is negative, or the first argument is negative and finite and the second argument is negative infinity, then the result is the `double` value closest to -*pi*.
- If the first argument is positive and the second argument is positive zero or negative zero, or the first argument is positive infinity and the second argument is finite, then the result is the `double` value closest to *pi*/2.
- If the first argument is negative and the second argument is positive zero or negative zero, or the first argument is negative infinity and the second argument is finite, then the result is the `double` value closest to -*pi*/2.
- If both arguments are positive infinity, then the result is the `double` value closest to *pi*/4.
- If the first argument is positive infinity and the second argument is negative infinity, then the result is the `double` value closest to 3*pi/4.
- If the first argument is negative infinity and the second argument is positive infinity, then the result is the `double` value closest to -*pi*/4.
- If both arguments are negative infinity, then the result is the `double` value closest to -3*pi/4.

The computed result must be within 2 ulps of the exact result. Results must be semi-monotonic.

Parameters:

`y` - the ordinate coordinate

`x` - the abscissa coordinate

Returns:

the *theta* component of the point (*r*, *theta*) in polar coordinates that corresponds to the point (*x*, *y*) in Cartesian coordinates.

pow

```
public static double pow(double a,
                         double b)
```

Returns the value of the first argument raised to the power of the second argument. Special cases:

- If the second argument is positive or negative zero, then the result is 1.0.
- If the second argument is 1.0, then the result is the same as the first argument.
- If the second argument is NaN, then the result is NaN.
- If the first argument is NaN and the second argument is nonzero, then the result is NaN.
- If
  - the absolute value of the first argument is greater than 1 and the second argument is positive infinity, or
  - the absolute value of the first argument is less than 1 and the second argument is negative infinity,

  then the result is positive infinity.
- If
  - the absolute value of the first argument is greater than 1 and the second argument is negative infinity, or
  - the absolute value of the first argument is less than 1 and the second argument is positive infinity,

  then the result is positive zero.
- If the absolute value of the first argument equals 1 and the second argument is infinite, then the result is NaN.
- If
  - the first argument is positive zero and the second argument is greater than zero, or
  - the first argument is positive infinity and the second argument is less than zero,

  then the result is positive zero.
- If
  - the first argument is positive zero and the second argument is less than zero, or
  - the first argument is positive infinity and the second argument is greater than zero,

  then the result is positive infinity.
- If
  - the first argument is negative zero and the second argument is greater than zero but not a finite odd integer, or
  - the first argument is negative infinity and the second argument is less than zero but not a finite odd integer,

- the first argument is negative infinity and the second argument is less than zero but not a finite odd integer,

then the result is positive zero.

- If
  - the first argument is negative zero and the second argument is a positive finite odd integer, or
  - the first argument is negative infinity and the second argument is a negative finite odd integer,

  then the result is negative zero.
- If
  - the first argument is negative zero and the second argument is less than zero but not a finite odd integer, or
  - the first argument is negative infinity and the second argument is greater than zero but not a finite odd integer,

  then the result is positive infinity.
- If
  - the first argument is negative zero and the second argument is a negative finite odd integer, or
  - the first argument is negative infinity and the second argument is a positive finite odd integer,

  then the result is negative infinity.
- If the first argument is finite and less than zero
  - if the second argument is a finite even integer, the result is equal to the result of raising the absolute value of the first argument to the power of the second argument
  - if the second argument is a finite odd integer, the result is equal to the negative of the result of raising the absolute value of the first argument to the power of the second argument
  - if the second argument is finite and not an integer, then the result is NaN.
- If both arguments are integers, then the result is exactly equal to the mathematical result of raising the first argument to the power of the second argument if that result can in fact be represented exactly as a `double` value.

(In the foregoing descriptions, a floating-point value is considered to be an integer if and only if it is finite and a fixed point of the method `ceil` or, equivalently, a fixed point of the method `floor`. A value is a fixed point of a one-argument method if and only if the result of applying the method to the value is equal to the value.)

The computed result must be within 1 ulp of the exact result. Results must be semi-monotonic.

Parameters:

`a - the base.`

`b - the exponent.`

Returns:

`the value` a^b.

round

```
public static int round(float a)
```

Returns the closest `int` to the argument, with ties rounding to positive infinity.

Special cases:

- If the argument is NaN, the result is 0.
- If the argument is negative infinity or any value less than or equal to the value of `Integer.MIN_VALUE`, the result is equal to the value of `Integer.MIN_VALUE`.
- If the argument is positive infinity or any value greater than or equal to the value of `Integer.MAX_VALUE`, the result is equal to the value of `Integer.MAX_VALUE`.

Parameters:

`a - a floating-point value to be rounded to an integer.`

Returns:

`the value of the argument rounded to the nearest int value.`

See Also:

`Integer.MAX_VALUE, Integer.MIN_VALUE`

round

```
public static long round(double a)
```

Returns the closest `long` to the argument, with ties rounding to positive infinity.

Special cases:

- If the argument is NaN, the result is 0.
- If the argument is negative infinity or any value less than or equal to the value of `Long.MIN_VALUE`, the result is equal to the value of `Long.MIN_VALUE`.
- If the argument is positive infinity or any value greater than or equal to the value of `Long.MAX_VALUE`, the result is equal to the value of `Long.MAX_VALUE`.

Parameters:

`a - a floating-point value to be rounded to a long.`

Returns:

`the value of the argument rounded to the nearest long value.`

See Also:

`Long.MAX_VALUE, Long.MIN_VALUE`

random

```
public static double random()
```

Returns a `double` value with a positive sign, greater than or equal to `0.0` and less than `1.0`. Returned values are chosen pseudorandomly with (approximately) uniform distribution from that range.

When this method is first called, it creates a single new pseudorandom-number generator, exactly as if by the expression

```
new java.util.Random()
```

This new pseudorandom-number generator is used thereafter for all calls to this method and is used nowhere else.

This method is properly synchronized to allow correct use by more than one thread. However, if many threads need to generate pseudorandom numbers at a great rate, it may reduce contention for each thread to have its own pseudorandom-number generator.

Returns:

`a pseudorandom double greater than or equal to 0.0 and less than 1.0.`

See Also:

`Random.nextDouble()`

Random.nextDouble()

addExact

```
public static int addExact(int x,
                           int y)
```

Returns the sum of its arguments, throwing an exception if the result overflows an `int`.

Parameters:

`x` - the first value

`y` - the second value

Returns:

the result

Throws:

`ArithmeticException` - if the result overflows an int

Since:

1.8

addExact

```
public static long addExact(long x,
                            long y)
```

Returns the sum of its arguments, throwing an exception if the result overflows a `long`.

Parameters:

`x` - the first value

`y` - the second value

Returns:

the result

Throws:

`ArithmeticException` - if the result overflows a long

Since:

1.8

subtractExact

```
public static int subtractExact(int x,
                                int y)
```

Returns the difference of the arguments, throwing an exception if the result overflows an `int`.

Parameters:

`x` - the first value

`y` - the second value to subtract from the first

Returns:

the result

Throws:

`ArithmeticException` - if the result overflows an int

Since:

1.8

subtractExact

```
public static long subtractExact(long x,
                                 long y)
```

Returns the difference of the arguments, throwing an exception if the result overflows a `long`.

Parameters:

`x` - the first value

`y` - the second value to subtract from the first

Returns:

the result

Throws:

`ArithmeticException` - if the result overflows a long

Since:

1.8

multiplyExact

```
public static int multiplyExact(int x,
                                int y)
```

Returns the product of the arguments, throwing an exception if the result overflows an `int`.

Parameters:

`x` - the first value

`y` - the second value

Returns:
the result

Throws:
ArithmeticException - if the result overflows an int

Since:
1.8

multiplyExact

```
public static long multiplyExact(long x,
                                 long y)
```

Returns the product of the arguments, throwing an exception if the result overflows a long.

Parameters:
x - the first value

y - the second value

Returns:
the result

Throws:
ArithmeticException - if the result overflows a long

Since:
1.8

incrementExact

```
public static int incrementExact(int a)
```

Returns the argument incremented by one, throwing an exception if the result overflows an int.

Parameters:
a - the value to increment

Returns:
the result

Throws:
ArithmeticException - if the result overflows an int

Since:
1.8

incrementExact

```
public static long incrementExact(long a)
```

Returns the argument incremented by one, throwing an exception if the result overflows a long.

Parameters:
a - the value to increment

Returns:
the result

Throws:
ArithmeticException - if the result overflows a long

Since:
1.8

decrementExact

```
public static int decrementExact(int a)
```

Returns the argument decremented by one, throwing an exception if the result overflows an int.

Parameters:
a - the value to decrement

Returns:
the result

Throws:
ArithmeticException - if the result overflows an int

Since:
1.8

decrementExact

```
public static long decrementExact(long a)
```

Returns the argument decremented by one, throwing an exception if the result overflows a long.

Parameters:
a - the value to decrement

Returns:
the result

Throws:
ArithmeticException - if the result overflows a long

Since:
1.8

negateExact

```
public static int negateExact(int a)
```

Returns the negation of the argument, throwing an exception if the result overflows an int.

Parameters:
a - the value to negate

Returns:
the result

Throws:
ArithmeticException - if the result overflows an int

Since:
1.8

negateExact

```
public static long negateExact(long a)
```

Returns the negation of the argument, throwing an exception if the result overflows a long.

Parameters:
a - the value to negate

Returns:
the result

Throws:
ArithmeticException - if the result overflows a long

Since:
1.8

toIntExact

```
public static int toIntExact(long value)
```

Returns the value of the long argument; throwing an exception if the value overflows an int.

Parameters:
value - the long value

Returns:
the argument as an int

Throws:
ArithmeticException - if the argument overflows an int

Since:
1.8

floorDiv

```
public static int floorDiv(int x,
                           int y)
```

Returns the largest (closest to positive infinity) int value that is less than or equal to the algebraic quotient. There is one special case, if the dividend is the Integer.MIN_VALUE and the divisor is -1, then integer overflow occurs and the result is equal to the Integer.MIN_VALUE.

Normal integer division operates under the round to zero rounding mode (truncation). This operation instead acts under the round toward negative infinity (floor) rounding mode. The floor rounding mode gives different results than truncation when the exact result is negative.

- If the signs of the arguments are the same, the results of floorDiv and the / operator are the same.
  For example, floorDiv(4, 3) == 1 and (4 / 3) == 1.
- If the signs of the arguments are different, the quotient is negative and floorDiv returns the integer less than or equal to the quotient and the / operator returns the integer closest to zero.
  For example, floorDiv(-4, 3) == -2, whereas (-4 / 3) == -1.

Parameters:
x - the dividend

y - the divisor

Returns:
the largest (closest to positive infinity) int value that is less than or equal to the algebraic quotient.

Throws:
ArithmeticException - if the divisor y is zero

Since:
1.8

See Also:
floorMod(int, int), floor(double)

floorDiv

```
public static long floorDiv(long x,
                            long y)
```

Returns the largest (closest to positive infinity) `long` value that is less than or equal to the algebraic quotient. There is one special case, if the dividend is the Long.MIN_VALUE and the divisor is `-1`, then integer overflow occurs and the result is equal to the `Long.MIN_VALUE`.

Normal integer division operates under the round to zero rounding mode (truncation). This operation instead acts under the round toward negative infinity (floor) rounding mode. The floor rounding mode gives different results than truncation when the exact result is negative.

For examples, see `floorDiv(int, int)`.

Parameters:

`x - the dividend`

`y - the divisor`

Returns:

`the largest (closest to positive infinity) long value that is less than or equal to the algebraic quotient.`

Throws:

`ArithmeticException - if the divisor y is zero`

Since:

1.8

See Also:

`floorMod(long, long), floor(double)`

floorMod

```
public static int floorMod(int x,
                           int y)
```

Returns the floor modulus of the `int` arguments.

The floor modulus is `x - (floorDiv(x, y) * y)`, has the same sign as the divisor y, and is in the range of `-abs(y) < r < +abs(y)`.

The relationship between `floorDiv` and `floorMod` is such that:

- `floorDiv(x, y) * y + floorMod(x, y) == x`

The difference in values between `floorMod` and the `%` operator is due to the difference between `floorDiv` that returns the integer less than or equal to the quotient and the `/` operator that returns the integer closest to zero.

Examples:

- If the signs of the arguments are the same, the results of `floorMod` and the `%` operator are the same.
  - `floorMod(4, 3) == 1; and (4 % 3) == 1`
- If the signs of the arguments are different, the results differ from the `%` operator.
  - `floorMod(+4, -3) == -2; and (+4 % -3) == +1`
  - `floorMod(-4, +3) == +2; and (-4 % +3) == -1`
  - `floorMod(-4, -3) == -1; and (-4 % -3) == -1`

If the signs of arguments are unknown and a positive modulus is needed it can be computed as `(floorMod(x, y) + abs(y)) % abs(y)`.

Parameters:

`x - the dividend`

`y - the divisor`

Returns:

`the floor modulus x - (floorDiv(x, y) * y)`

Throws:

`ArithmeticException - if the divisor y is zero`

Since:

1.8

See Also:

`floorDiv(int, int)`

floorMod

```
public static long floorMod(long x,
                            long y)
```

Returns the floor modulus of the `long` arguments.

The floor modulus is `x - (floorDiv(x, y) * y)`, has the same sign as the divisor y, and is in the range of `-abs(y) < r < +abs(y)`.

The relationship between `floorDiv` and `floorMod` is such that:

- `floorDiv(x, y) * y + floorMod(x, y) == x`

For examples, see `floorMod(int, int)`.

Parameters:

`x - the dividend`

`y - the divisor`

Returns:

`the floor modulus x - (floorDiv(x, y) * y)`

Throws:

`ArithmeticException - if the divisor y is zero`

Since:

1.8

See Also:

floorDiv(long, long)

abs

```
public static int abs(int a)
```

Returns the absolute value of an `int` value. If the argument is not negative, the argument is returned. If the argument is negative, the negation of the argument is returned.

Note that if the argument is equal to the value of `Integer.MIN_VALUE`, the most negative representable `int` value, the result is that same value, which is negative.

Parameters:

`a` - the argument whose absolute value is to be determined

Returns:

the absolute value of the argument.

abs

```
public static long abs(long a)
```

Returns the absolute value of a `long` value. If the argument is not negative, the argument is returned. If the argument is negative, the negation of the argument is returned.

Note that if the argument is equal to the value of `Long.MIN_VALUE`, the most negative representable `long` value, the result is that same value, which is negative.

Parameters:

`a` - the argument whose absolute value is to be determined

Returns:

the absolute value of the argument.

abs

```
public static float abs(float a)
```

Returns the absolute value of a `float` value. If the argument is not negative, the argument is returned. If the argument is negative, the negation of the argument is returned. Special cases:

- If the argument is positive zero or negative zero, the result is positive zero.
- If the argument is infinite, the result is positive infinity.
- If the argument is NaN, the result is NaN.

In other words, the result is the same as the value of the expression:

```
Float.intBitsToFloat(0x7fffffff & Float.floatToIntBits(a))
```

Parameters:

`a` - the argument whose absolute value is to be determined

Returns:

the absolute value of the argument.

abs

```
public static double abs(double a)
```

Returns the absolute value of a `double` value. If the argument is not negative, the argument is returned. If the argument is negative, the negation of the argument is returned. Special cases:

- If the argument is positive zero or negative zero, the result is positive zero.
- If the argument is infinite, the result is positive infinity.
- If the argument is NaN, the result is NaN.

In other words, the result is the same as the value of the expression:

```
Double.longBitsToDouble((Double.doubleToLongBits(a)<<1)>>>1)
```

Parameters:

`a` - the argument whose absolute value is to be determined

Returns:

the absolute value of the argument.

max

```
public static int max(int a,
                      int b)
```

Returns the greater of two `int` values. That is, the result is the argument closer to the value of `Integer.MAX_VALUE`. If the arguments have the same value, the result is that same value.

Parameters:

`a` - an argument.

`b` - another argument.

Returns:

the larger of a and b.

max

```
public static long max(long a,
                       long b)
```

Returns the greater of two `long` values. That is, the result is the argument closer to the value of `Long.MAX_VALUE`. If the arguments have the same value, the result is that same value.

Parameters:

a - an argument.

b - another argument.

Returns:

the larger of a and b.

max

```
public static float max(float a,
                        float b)
```

Returns the greater of two `float` values. That is, the result is the argument closer to positive infinity. If the arguments have the same value, the result is that same value. If either value is NaN, then the result is NaN. Unlike the numerical comparison operators, this method considers negative zero to be strictly smaller than positive zero. If one argument is positive zero and the other negative zero, the result is positive zero.

Parameters:

a - an argument.

b - another argument.

Returns:

the larger of a and b.

max

```
public static double max(double a,
                         double b)
```

Returns the greater of two `double` values. That is, the result is the argument closer to positive infinity. If the arguments have the same value, the result is that same value. If either value is NaN, then the result is NaN. Unlike the numerical comparison operators, this method considers negative zero to be strictly smaller than positive zero. If one argument is positive zero and the other negative zero, the result is positive zero.

Parameters:

a - an argument.

b - another argument.

Returns:

the larger of a and b.

min

```
public static int min(int a,
                      int b)
```

Returns the smaller of two `int` values. That is, the result the argument closer to the value of `Integer.MIN_VALUE`. If the arguments have the same value, the result is that same value.

Parameters:

a - an argument.

b - another argument.

Returns:

the smaller of a and b.

min

```
public static long min(long a,
                       long b)
```

Returns the smaller of two `long` values. That is, the result is the argument closer to the value of `Long.MIN_VALUE`. If the arguments have the same value, the result is that same value.

Parameters:

a - an argument.

b - another argument.

Returns:

the smaller of a and b.

min

```
public static float min(float a,
                        float b)
```

Returns the smaller of two `float` values. That is, the result is the value closer to negative infinity. If the arguments have the same value, the result is that same value. If either value is NaN, then the result is NaN. Unlike the numerical comparison operators, this method considers negative zero to be strictly smaller than positive zero. If one argument is positive zero and the other is negative zero, the result is negative zero.

Parameters:

a - an argument.

b - another argument.

Returns:

the smaller of a and b.

min

```
public static double min(double a,
```

double b)

Returns the smaller of two double values. That is, the result is the value closer to negative infinity. If the arguments have the same value, the result is that same value. If either value is NaN, then the result is NaN. Unlike the numerical comparison operators, this method considers negative zero to be strictly smaller than positive zero. If one argument is positive zero and the other is negative zero, the result is negative zero.

Parameters:

a - an argument.

b - another argument.

Returns:

the smaller of a and b.

ulp

public static double ulp(double d)

Returns the size of an ulp of the argument. An ulp, unit in the last place, of a double value is the positive distance between this floating-point value and the double value next larger in magnitude. Note that for non-NaN x, ulp(-x) == ulp(x).

Special Cases:

- If the argument is NaN, then the result is NaN.
- If the argument is positive or negative infinity, then the result is positive infinity.
- If the argument is positive or negative zero, then the result is Double.MIN_VALUE.
- If the argument is ±Double.MAX_VALUE, then the result is equal to 2^{971}.

Parameters:

d - the floating-point value whose ulp is to be returned

Returns:

the size of an ulp of the argument

Since:

1.5

ulp

public static float ulp(float f)

Returns the size of an ulp of the argument. An ulp, unit in the last place, of a float value is the positive distance between this floating-point value and the float value next larger in magnitude. Note that for non-NaN x, ulp(-x) == ulp(x).

Special Cases:

- If the argument is NaN, then the result is NaN.
- If the argument is positive or negative infinity, then the result is positive infinity.
- If the argument is positive or negative zero, then the result is Float.MIN_VALUE.
- If the argument is ±Float.MAX_VALUE, then the result is equal to 2^{104}.

Parameters:

f - the floating-point value whose ulp is to be returned

Returns:

the size of an ulp of the argument

Since:

1.5

signum

public static double signum(double d)

Returns the signum function of the argument; zero if the argument is zero, 1.0 if the argument is greater than zero, -1.0 if the argument is less than zero.

Special Cases:

- If the argument is NaN, then the result is NaN.
- If the argument is positive zero or negative zero, then the result is the same as the argument.

Parameters:

d - the floating-point value whose signum is to be returned

Returns:

the signum function of the argument

Since:

1.5

signum

public static float signum(float f)

Returns the signum function of the argument; zero if the argument is zero, 1.0f if the argument is greater than zero, -1.0f if the argument is less than zero.

Special Cases:

- If the argument is NaN, then the result is NaN.
- If the argument is positive zero or negative zero, then the result is the same as the argument.

Parameters:

f - the floating-point value whose signum is to be returned

Returns:

the signum function of the argument

Since:

1.5

sinh

```
public static double sinh(double x)
```

Returns the hyperbolic sine of a `double` value. The hyperbolic sine of x is defined to be $(e^x - e^{-x})/2$ where e is Euler's number.

Special cases:

- If the argument is NaN, then the result is NaN.
- If the argument is infinite, then the result is an infinity with the same sign as the argument.
- If the argument is zero, then the result is a zero with the same sign as the argument.

The computed result must be within 2.5 ulps of the exact result.

Parameters:

`x - The number whose hyperbolic sine is to be returned.`

Returns:

`The hyperbolic sine of x.`

Since:

1.5

cosh

```
public static double cosh(double x)
```

Returns the hyperbolic cosine of a `double` value. The hyperbolic cosine of x is defined to be $(e^x + e^{-x})/2$ where e is Euler's number.

Special cases:

- If the argument is NaN, then the result is NaN.
- If the argument is infinite, then the result is positive infinity.
- If the argument is zero, then the result is `1.0`.

The computed result must be within 2.5 ulps of the exact result.

Parameters:

`x - The number whose hyperbolic cosine is to be returned.`

Returns:

`The hyperbolic cosine of x.`

Since:

1.5

tanh

```
public static double tanh(double x)
```

Returns the hyperbolic tangent of a `double` value. The hyperbolic tangent of x is defined to be $(e^x - e^{-x})/(e^x + e^{-x})$, in other words, $\sinh(x)/\cosh(x)$. Note that the absolute value of the exact tanh is always less than 1.

Special cases:

- If the argument is NaN, then the result is NaN.
- If the argument is zero, then the result is a zero with the same sign as the argument.
- If the argument is positive infinity, then the result is `+1.0`.
- If the argument is negative infinity, then the result is `-1.0`.

The computed result must be within 2.5 ulps of the exact result. The result of `tanh` for any finite input must have an absolute value less than or equal to 1. Note that once the exact result of tanh is within 1/2 of an ulp of the limit value of ±1, correctly signed `±1.0` should be returned.

Parameters:

`x - The number whose hyperbolic tangent is to be returned.`

Returns:

`The hyperbolic tangent of x.`

Since:

1.5

hypot

```
public static double hypot(double x,
                           double y)
```

Returns sqrt(x^2 +y^2) without intermediate overflow or underflow.

Special cases:

- If either argument is infinite, then the result is positive infinity.
- If either argument is NaN and neither argument is infinite, then the result is NaN.

The computed result must be within 1 ulp of the exact result. If one parameter is held constant, the results must be semi-monotonic in the other parameter.

Parameters:

`x - a value`

`y - a value`

Returns:

`sqrt(`x^2 +y^2`) without intermediate overflow or underflow`

Since:

1.5

expm1

```
public static double expm1(double x)
```

Returns e^x -1. Note that for values of x near 0, the exact sum of `expm1(x) + 1` is much closer to the true result of e^x than `exp(x)`.

Special cases:

- If the argument is NaN, the result is NaN.
- If the argument is positive infinity, then the result is positive infinity.
- If the argument is negative infinity, then the result is -1.0.
- If the argument is zero, then the result is a zero with the same sign as the argument.

The computed result must be within 1 ulp of the exact result. Results must be semi-monotonic. The result of `expm1` for any finite input must be greater than or equal to `-1.0`. Note that once the exact result of e^X - 1 is within 1/2 ulp of the limit value -1, `-1.0` should be returned.

Parameters:

`x - the exponent to raise e to in the computation of` e^X `-1.`

Returns:

`the value` e^X `- 1.`

Since:

1.5

log1p

```
public static double log1p(double x)
```

Returns the natural logarithm of the sum of the argument and 1. Note that for small values `x`, the result of `log1p(x)` is much closer to the true result of ln(1 + x) than the floating-point evaluation of `log(1.0+x)`.

Special cases:

- If the argument is NaN or less than -1, then the result is NaN.
- If the argument is positive infinity, then the result is positive infinity.
- If the argument is negative one, then the result is negative infinity.
- If the argument is zero, then the result is a zero with the same sign as the argument.

The computed result must be within 1 ulp of the exact result. Results must be semi-monotonic.

Parameters:

`x - a value`

Returns:

`the value ln(x + 1), the natural log of x + 1`

Since:

1.5

copySign

```
public static double copySign(double magnitude,
                              double sign)
```

Returns the first floating-point argument with the sign of the second floating-point argument. Note that unlike the `StrictMath.copySign` method, this method does not require NaN `sign` arguments to be treated as positive values; implementations are permitted to treat some NaN arguments as positive and other NaN arguments as negative to allow greater performance.

Parameters:

`magnitude - the parameter providing the magnitude of the result`

`sign - the parameter providing the sign of the result`

Returns:

`a value with the magnitude of magnitude and the sign of sign.`

Since:

1.6

copySign

```
public static float copySign(float magnitude,
                             float sign)
```

Returns the first floating-point argument with the sign of the second floating-point argument. Note that unlike the `StrictMath.copySign` method, this method does not require NaN `sign` arguments to be treated as positive values; implementations are permitted to treat some NaN arguments as positive and other NaN arguments as negative to allow greater performance.

Parameters:

`magnitude - the parameter providing the magnitude of the result`

`sign - the parameter providing the sign of the result`

Returns:

`a value with the magnitude of magnitude and the sign of sign.`

Since:

1.6

getExponent

- If the argument is zero or subnormal, then the result is Float.MIN_EXPONENT -1.

Parameters:

f - a float value

Returns:

the unbiased exponent of the argument

Since:

1.6

getExponent

```
public static int getExponent(double d)
```

Returns the unbiased exponent used in the representation of a double. Special cases:

- If the argument is NaN or infinite, then the result is Double.MAX_EXPONENT + 1.
- If the argument is zero or subnormal, then the result is Double.MIN_EXPONENT -1.

Parameters:

d - a double value

Returns:

the unbiased exponent of the argument

Since:

1.6

nextAfter

```
public static double nextAfter(double start,
                               double direction)
```

Returns the floating-point number adjacent to the first argument in the direction of the second argument. If both arguments compare as equal the second argument is returned.

Special cases:

- If either argument is a NaN, then NaN is returned.
- If both arguments are signed zeros, direction is returned unchanged (as implied by the requirement of returning the second argument if the arguments compare as equal).
- If start is ±Double.MIN_VALUE and direction has a value such that the result should have a smaller magnitude, then a zero with the same sign as start is returned.
- If start is infinite and direction has a value such that the result should have a smaller magnitude, Double.MAX_VALUE with the same sign as start is returned.
- If start is equal to ± Double.MAX_VALUE and direction has a value such that the result should have a larger magnitude, an infinity with same sign as start is returned.

Parameters:

start - starting floating-point value

direction - value indicating which of start's neighbors or start should be returned

Returns:

The floating-point number adjacent to start in the direction of direction.

Since:

1.6

nextAfter

```
public static float nextAfter(float start,
                              double direction)
```

Returns the floating-point number adjacent to the first argument in the direction of the second argument. If both arguments compare as equal a value equivalent to the second argument is returned.

Special cases:

- If either argument is a NaN, then NaN is returned.
- If both arguments are signed zeros, a value equivalent to direction is returned.
- If start is ±Float.MIN_VALUE and direction has a value such that the result should have a smaller magnitude, then a zero with the same sign as start is returned.
- If start is infinite and direction has a value such that the result should have a smaller magnitude, Float.MAX_VALUE with the same sign as start is returned.
- If start is equal to ± Float.MAX_VALUE and direction has a value such that the result should have a larger magnitude, an infinity with same sign as start is returned.

Parameters:

start - starting floating-point value

direction - value indicating which of start's neighbors or start should be returned

Returns:

The floating-point number adjacent to start in the direction of direction.

Since:

1.6

nextUp

```
public static double nextUp(double d)
```

Returns the floating-point value adjacent to d in the direction of positive infinity. This method is semantically equivalent to nextAfter(d, Double.POSITIVE_INFINITY); however, a nextUp implementation may run faster than its equivalent nextAfter call.

Special Cases:

- If the argument is NaN, the result is NaN.
- If the argument is positive infinity, the result is positive infinity.
- If the argument is zero, the result is Double.MIN_VALUE

Parameters:

d - starting floating-point value

Returns:

The adjacent floating-point value closer to positive infinity.

Since:

1.6

nextUp

```
public static float nextUp(float f)
```

Returns the floating-point value adjacent to `f` in the direction of positive infinity. This method is semantically equivalent to `nextAfter(f, Float.POSITIVE_INFINITY)`; however, a `nextUp` implementation may run faster than its equivalent `nextAfter` call.

Special Cases:

- If the argument is NaN, the result is NaN.
- If the argument is positive infinity, the result is positive infinity.
- If the argument is zero, the result is `Float.MIN_VALUE`

Parameters:

`f` - starting floating-point value

Returns:

The adjacent floating-point value closer to positive infinity.

Since:

1.6

nextDown

```
public static double nextDown(double d)
```

Returns the floating-point value adjacent to `d` in the direction of negative infinity. This method is semantically equivalent to `nextAfter(d, Double.NEGATIVE_INFINITY)`; however, a `nextDown` implementation may run faster than its equivalent `nextAfter` call.

Special Cases:

- If the argument is NaN, the result is NaN.
- If the argument is negative infinity, the result is negative infinity.
- If the argument is zero, the result is `-Double.MIN_VALUE`

Parameters:

`d` - starting floating-point value

Returns:

The adjacent floating-point value closer to negative infinity.

Since:

1.8

nextDown

```
public static float nextDown(float f)
```

Returns the floating-point value adjacent to `f` in the direction of negative infinity. This method is semantically equivalent to `nextAfter(f, Float.NEGATIVE_INFINITY)`; however, a `nextDown` implementation may run faster than its equivalent `nextAfter` call.

Special Cases:

- If the argument is NaN, the result is NaN.
- If the argument is negative infinity, the result is negative infinity.
- If the argument is zero, the result is `-Float.MIN_VALUE`

Parameters:

`f` - starting floating-point value

Returns:

The adjacent floating-point value closer to negative infinity.

Since:

1.8

scalb

```
public static double scalb(double d,
                           int scaleFactor)
```

Returns $d \times 2^{scaleFactor}$ rounded as if performed by a single correctly rounded floating-point multiply to a member of the double value set. See the Java Language Specification for a discussion of floating-point value sets. If the exponent of the result is between `Double.MIN_EXPONENT` and `Double.MAX_EXPONENT`, the answer is calculated exactly. If the exponent of the result would be larger than `Double.MAX_EXPONENT`, an infinity is returned. Note that if the result is subnormal, precision may be lost; that is, when `scalb(x, n)` is subnormal, `scalb(scalb(x, n), -n)` may not equal `x`. When the result is non-NaN, the result has the same sign as `d`.

Special cases:

- If the first argument is NaN, NaN is returned.
- If the first argument is infinite, then an infinity of the same sign is returned.
- If the first argument is zero, then a zero of the same sign is returned.

Parameters:

`d` - number to be scaled by a power of two.

`scaleFactor` - power of 2 used to scale `d`

Returns:

$d \times 2^{scaleFactor}$

Since:

1.6

scalb

```
public static float scalb(float f,
                          int scaleFactor)
```

Returns $f \times 2^{scaleFactor}$ rounded as if performed by a single correctly rounded floating-point multiply to a member of the float value set. See the Java Language Specification for a discussion of floating-point value sets. If the exponent of the result is between `Float.MIN_EXPONENT` and `Float.MAX_EXPONENT`, the answer is calculated exactly. If the exponent of the result would be larger than `Float.MAX_EXPONENT`, an infinity is returned. Note that if the result is subnormal, precision may be lost; that is, when `scalb(x, n)` is subnormal, `scalb(scalb(x, n), -n)` may not equal `x`. When the result is non-NaN, the result has the same sign as `f`.

Special cases:

- If the first argument is NaN, NaN is returned.
- If the first argument is infinite, then an infinity of the same sign is returned.
- If the first argument is zero, then a zero of the same sign is returned.

Parameters:

`f - number to be scaled by a power of two.`

`scaleFactor - power of 2 used to scale f`

Returns:

$f \times 2^{scaleFactor}$

Since:

1.6