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 difference between the two floating-point values closest to 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; otherwise, either of the two floating-point numbers closest to the exact result may be returned. Besides accuracy at individual arguments, maintaining proper relations between the method at different arguments is also important. Therefore, 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.

Since:

JDK1.0

Field Summary

`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

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 an angle, in the range of 0.0 through *pi*.
static double	**asin**(double a) Returns the arc sine of an angle, in the range of -*pi*/2 through *pi*/2.
static double	**atan**(double a) Returns the arc tangent of an angle, in the range of -*pi*/2 through *pi*/2.
static double	**atan2**(double y, double x) Converts rectangular coordinates (x, y) to polar (r, *theta*).
static double	**ceil**(double a) Returns the smallest (closest to negative infinity) double value that is not less than the argument and is equal to a mathematical integer.
static double	**cos**(double a) Returns the trigonometric cosine of an angle.
static double	**exp**(double a) Returns Euler's number *e* raised to the power of a double value.
static double	**floor**(double a) Returns the largest (closest to positive infinity) double value that is not greater than the argument and is equal to a mathematical integer.
static double	**IEEEremainder**(double f1, double f2) Computes the remainder operation on two arguments as prescribed by the IEEE 754 standard.
static double	**log**(double a) Returns the natural logarithm (base *e*) of a double value.
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 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.
static int	**round**(float a) Returns the closest int to the argument.
static double	**sin**(double a) Returns the trigonometric sine of an angle.
static double	**sqrt**(double a) Returns the correctly rounded positive square root of a double value.
static double	**tan**(double a) Returns the trigonometric tangent of an angle.
static double	**toDegrees**(double angrad) Converts an angle measured in radians to an approximately equivalent angle measured in degrees.
static double	**toRadians**(double angdeg) Converts an angle measured in degrees to an approximately equivalent angle measured in radians.

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.

A result must be within 1 ulp of the correctly rounded 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.

A result must be within 1 ulp of the correctly rounded 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.

A result must be within 1 ulp of the correctly rounded 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 an angle, in the range of -*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.

A result must be within 1 ulp of the correctly rounded 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 an angle, in the range of 0.0 through *pi*. Special case:

- If the argument is NaN or its absolute value is greater than 1, then the result is NaN.

A result must be within 1 ulp of the correctly rounded 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 an angle, in the range of -*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.

A result must be within 1 ulp of the correctly rounded 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.

A result must be within 1 ulp of the correctly rounded 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.

A result must be within 1 ulp of the correctly rounded result. Results must be semi-monotonic.

Parameters:
`a` - a number greater than `0.0`.
Returns:
the value ln `a`, the natural logarithm of `a`.

---

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.

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 not less than 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 not less than 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 not greater than 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 is not greater than 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)
```

Converts rectangular coordinates (`x`, `y`) to polar (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.

A result must be within 2 ulps of the correctly rounded 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,

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.)

A result must be within 1 ulp of the correctly rounded 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. The result is rounded to an integer by adding 1/2, taking the floor of the result, and casting the result to type `int`. In other words, the result is equal to the value of the expression:

```
(int)Math.floor(a + 0.5f)
```

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. The result is rounded to an integer by adding 1/2, taking the floor of the result, and casting the result to type `long`. In other words, the result is equal to the value of the expression:

```
(long)Math.floor(a + 0.5d)
```

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()`

---

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.
See Also:
`Integer.MIN_VALUE`

---

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.
See Also:
`Long.MIN_VALUE`

---

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`.

See Also:

`Long.MAX_VALUE`

---

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`.

See Also:

Long.MAX_VALUE

---

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 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 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`.

See Also:

Long.MIN_VALUE

---

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`.

See Also:

`Long.MIN_VALUE`

---

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 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 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`.

Submit a bug or feature
For further API reference and developer documentation, see Java 2 SDK SE Developer Documentation. That documentation contains more detailed, developer-targeted descriptions, with conceptual overviews, definitions of terms, workarounds, and working code examples.