| | [**Overview**](http://docs.google.com/overview-summary.html) | [**Package**](http://docs.google.com/package-summary.html) | **Class** | [**Use**](http://docs.google.com/class-use/Float.html) | [**Tree**](http://docs.google.com/package-tree.html) | [**Deprecated**](http://docs.google.com/deprecated-list.html) | [**Index**](http://docs.google.com/index-files/index-1.html) | [**Help**](http://docs.google.com/help-doc.html) | | --- | --- | --- | --- | --- | --- | --- | --- | | | ***Java™ Platform***  ***Standard Ed. 6*** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| [**PREV CLASS**](http://docs.google.com/java/lang/ExceptionInInitializerError.html)   [**NEXT CLASS**](http://docs.google.com/java/lang/IllegalAccessError.html) | [**FRAMES**](http://docs.google.com/index.html?java/lang/Float.html)    [**NO FRAMES**](http://docs.google.com/Float.html)     [**All Classes**](http://docs.google.com/allclasses-noframe.html) |
| SUMMARY: NESTED | [FIELD](#3znysh7) | [CONSTR](#2et92p0) | [METHOD](#tyjcwt) | DETAIL: [FIELD](#1t3h5sf) | [CONSTR](#z337ya) | [METHOD](#2xcytpi) |

## **java.lang**

Class Float

[java.lang.Object](http://docs.google.com/java/lang/Object.html)  
 [java.lang.Number](http://docs.google.com/java/lang/Number.html)  
 **java.lang.Float**

**All Implemented Interfaces:** [Serializable](http://docs.google.com/java/io/Serializable.html), [Comparable](http://docs.google.com/java/lang/Comparable.html)<[Float](http://docs.google.com/java/lang/Float.html)>

public final class **Float**extends [Number](http://docs.google.com/java/lang/Number.html)implements [Comparable](http://docs.google.com/java/lang/Comparable.html)<[Float](http://docs.google.com/java/lang/Float.html)>

The Float class wraps a value of primitive type float in an object. An object of type Float contains a single field whose type is float.

In addition, this class provides several methods for converting a float to a String and a String to a float, as well as other constants and methods useful when dealing with a float.

**Since:** JDK1.0 **See Also:**[Serialized Form](http://docs.google.com/serialized-form.html#java.lang.Float)

| **Field Summary** | |
| --- | --- |
| static int | [**MAX\_EXPONENT**](http://docs.google.com/java/lang/Float.html#MAX_EXPONENT)            Maximum exponent a finite float variable may have. |
| static float | [**MAX\_VALUE**](http://docs.google.com/java/lang/Float.html#MAX_VALUE)            A constant holding the largest positive finite value of type float, (2-2-23)·2127. |
| static int | [**MIN\_EXPONENT**](http://docs.google.com/java/lang/Float.html#MIN_EXPONENT)            Minimum exponent a normalized float variable may have. |
| static float | [**MIN\_NORMAL**](http://docs.google.com/java/lang/Float.html#MIN_NORMAL)            A constant holding the smallest positive normal value of type float, 2-126. |
| static float | [**MIN\_VALUE**](http://docs.google.com/java/lang/Float.html#MIN_VALUE)            A constant holding the smallest positive nonzero value of type float, 2-149. |
| static float | [**NaN**](http://docs.google.com/java/lang/Float.html#NaN)            A constant holding a Not-a-Number (NaN) value of type float. |
| static float | [**NEGATIVE\_INFINITY**](http://docs.google.com/java/lang/Float.html#NEGATIVE_INFINITY)            A constant holding the negative infinity of type float. |
| static float | [**POSITIVE\_INFINITY**](http://docs.google.com/java/lang/Float.html#POSITIVE_INFINITY)            A constant holding the positive infinity of type float. |
| static int | [**SIZE**](http://docs.google.com/java/lang/Float.html#SIZE)            The number of bits used to represent a float value. |
| static [Class](http://docs.google.com/java/lang/Class.html)<[Float](http://docs.google.com/java/lang/Float.html)> | [**TYPE**](http://docs.google.com/java/lang/Float.html#TYPE)            The Class instance representing the primitive type float. |

| **Constructor Summary** | |
| --- | --- |
| [**Float**](http://docs.google.com/java/lang/Float.html#Float(double))(double value)            Constructs a newly allocated Float object that represents the argument converted to type float. |
| [**Float**](http://docs.google.com/java/lang/Float.html#Float(float))(float value)            Constructs a newly allocated Float object that represents the primitive float argument. |
| [**Float**](http://docs.google.com/java/lang/Float.html#Float(java.lang.String))([String](http://docs.google.com/java/lang/String.html) s)            Constructs a newly allocated Float object that represents the floating-point value of type float represented by the string. |

| **Method Summary** | |
| --- | --- |
| byte | [**byteValue**](http://docs.google.com/java/lang/Float.html#byteValue())()            Returns the value of this Float as a byte (by casting to a byte). |
| static int | [**compare**](http://docs.google.com/java/lang/Float.html#compare(float,%20float))(float f1, float f2)            Compares the two specified float values. |
| int | [**compareTo**](http://docs.google.com/java/lang/Float.html#compareTo(java.lang.Float))([Float](http://docs.google.com/java/lang/Float.html) anotherFloat)            Compares two Float objects numerically. |
| double | [**doubleValue**](http://docs.google.com/java/lang/Float.html#doubleValue())()            Returns the double value of this Float object. |
| boolean | [**equals**](http://docs.google.com/java/lang/Float.html#equals(java.lang.Object))([Object](http://docs.google.com/java/lang/Object.html) obj)            Compares this object against the specified object. |
| static int | [**floatToIntBits**](http://docs.google.com/java/lang/Float.html#floatToIntBits(float))(float value)            Returns a representation of the specified floating-point value according to the IEEE 754 floating-point "single format" bit layout. |
| static int | [**floatToRawIntBits**](http://docs.google.com/java/lang/Float.html#floatToRawIntBits(float))(float value)            Returns a representation of the specified floating-point value according to the IEEE 754 floating-point "single format" bit layout, preserving Not-a-Number (NaN) values. |
| float | [**floatValue**](http://docs.google.com/java/lang/Float.html#floatValue())()            Returns the float value of this Float object. |
| int | [**hashCode**](http://docs.google.com/java/lang/Float.html#hashCode())()            Returns a hash code for this Float object. |
| static float | [**intBitsToFloat**](http://docs.google.com/java/lang/Float.html#intBitsToFloat(int))(int bits)            Returns the float value corresponding to a given bit representation. |
| int | [**intValue**](http://docs.google.com/java/lang/Float.html#intValue())()            Returns the value of this Float as an int (by casting to type int). |
| boolean | [**isInfinite**](http://docs.google.com/java/lang/Float.html#isInfinite())()            Returns true if this Float value is infinitely large in magnitude, false otherwise. |
| static boolean | [**isInfinite**](http://docs.google.com/java/lang/Float.html#isInfinite(float))(float v)            Returns true if the specified number is infinitely large in magnitude, false otherwise. |
| boolean | [**isNaN**](http://docs.google.com/java/lang/Float.html#isNaN())()            Returns true if this Float value is a Not-a-Number (NaN), false otherwise. |
| static boolean | [**isNaN**](http://docs.google.com/java/lang/Float.html#isNaN(float))(float v)            Returns true if the specified number is a Not-a-Number (NaN) value, false otherwise. |
| long | [**longValue**](http://docs.google.com/java/lang/Float.html#longValue())()            Returns value of this Float as a long (by casting to type long). |
| static float | [**parseFloat**](http://docs.google.com/java/lang/Float.html#parseFloat(java.lang.String))([String](http://docs.google.com/java/lang/String.html) s)            Returns a new float initialized to the value represented by the specified String, as performed by the valueOf method of class Float. |
| short | [**shortValue**](http://docs.google.com/java/lang/Float.html#shortValue())()            Returns the value of this Float as a short (by casting to a short). |
| static [String](http://docs.google.com/java/lang/String.html) | [**toHexString**](http://docs.google.com/java/lang/Float.html#toHexString(float))(float f)            Returns a hexadecimal string representation of the float argument. |
| [String](http://docs.google.com/java/lang/String.html) | [**toString**](http://docs.google.com/java/lang/Float.html#toString())()            Returns a string representation of this Float object. |
| static [String](http://docs.google.com/java/lang/String.html) | [**toString**](http://docs.google.com/java/lang/Float.html#toString(float))(float f)            Returns a string representation of the float argument. |
| static [Float](http://docs.google.com/java/lang/Float.html) | [**valueOf**](http://docs.google.com/java/lang/Float.html#valueOf(float))(float f)            Returns a Float instance representing the specified float value. |
| static [Float](http://docs.google.com/java/lang/Float.html) | [**valueOf**](http://docs.google.com/java/lang/Float.html#valueOf(java.lang.String))([String](http://docs.google.com/java/lang/String.html) s)            Returns a Float object holding the float value represented by the argument string s. |

| **Methods inherited from class java.lang.**[**Object**](http://docs.google.com/java/lang/Object.html) |
| --- |
| [clone](http://docs.google.com/java/lang/Object.html#clone()), [finalize](http://docs.google.com/java/lang/Object.html#finalize()), [getClass](http://docs.google.com/java/lang/Object.html#getClass()), [notify](http://docs.google.com/java/lang/Object.html#notify()), [notifyAll](http://docs.google.com/java/lang/Object.html#notifyAll()), [wait](http://docs.google.com/java/lang/Object.html#wait()), [wait](http://docs.google.com/java/lang/Object.html#wait(long)), [wait](http://docs.google.com/java/lang/Object.html#wait(long,%20int)) |

| **Field Detail** |
| --- |

### POSITIVE\_INFINITY

public static final float **POSITIVE\_INFINITY**

A constant holding the positive infinity of type float. It is equal to the value returned by Float.intBitsToFloat(0x7f800000).

**See Also:**[Constant Field Values](http://docs.google.com/constant-values.html#java.lang.Float.POSITIVE_INFINITY)

### NEGATIVE\_INFINITY

public static final float **NEGATIVE\_INFINITY**

A constant holding the negative infinity of type float. It is equal to the value returned by Float.intBitsToFloat(0xff800000).

**See Also:**[Constant Field Values](http://docs.google.com/constant-values.html#java.lang.Float.NEGATIVE_INFINITY)

### NaN

public static final float **NaN**

A constant holding a Not-a-Number (NaN) value of type float. It is equivalent to the value returned by Float.intBitsToFloat(0x7fc00000).

**See Also:**[Constant Field Values](http://docs.google.com/constant-values.html#java.lang.Float.NaN)

### MAX\_VALUE

public static final float **MAX\_VALUE**

A constant holding the largest positive finite value of type float, (2-2-23)·2127. It is equal to the hexadecimal floating-point literal 0x1.fffffeP+127f and also equal to Float.intBitsToFloat(0x7f7fffff).

**See Also:**[Constant Field Values](http://docs.google.com/constant-values.html#java.lang.Float.MAX_VALUE)

### MIN\_NORMAL

public static final float **MIN\_NORMAL**

A constant holding the smallest positive normal value of type float, 2-126. It is equal to the hexadecimal floating-point literal 0x1.0p-126f and also equal to Float.intBitsToFloat(0x00800000).

**Since:** 1.6 **See Also:**[Constant Field Values](http://docs.google.com/constant-values.html#java.lang.Float.MIN_NORMAL)

### MIN\_VALUE

public static final float **MIN\_VALUE**

A constant holding the smallest positive nonzero value of type float, 2-149. It is equal to the hexadecimal floating-point literal 0x0.000002P-126f and also equal to Float.intBitsToFloat(0x1).

**See Also:**[Constant Field Values](http://docs.google.com/constant-values.html#java.lang.Float.MIN_VALUE)

### MAX\_EXPONENT

public static final int **MAX\_EXPONENT**

Maximum exponent a finite float variable may have. It is equal to the value returned by Math.getExponent(Float.MAX\_VALUE).

**Since:** 1.6 **See Also:**[Constant Field Values](http://docs.google.com/constant-values.html#java.lang.Float.MAX_EXPONENT)

### MIN\_EXPONENT

public static final int **MIN\_EXPONENT**

Minimum exponent a normalized float variable may have. It is equal to the value returned by Math.getExponent(Float.MIN\_NORMAL).

**Since:** 1.6 **See Also:**[Constant Field Values](http://docs.google.com/constant-values.html#java.lang.Float.MIN_EXPONENT)

### SIZE

public static final int **SIZE**

The number of bits used to represent a float value.

**Since:** 1.5 **See Also:**[Constant Field Values](http://docs.google.com/constant-values.html#java.lang.Float.SIZE)

### TYPE

public static final [Class](http://docs.google.com/java/lang/Class.html)<[Float](http://docs.google.com/java/lang/Float.html)> **TYPE**

The Class instance representing the primitive type float.

**Since:** JDK1.1

| **Constructor Detail** |
| --- |

### Float

public **Float**(float value)

Constructs a newly allocated Float object that represents the primitive float argument.

**Parameters:**value - the value to be represented by the Float.

### Float

public **Float**(double value)

Constructs a newly allocated Float object that represents the argument converted to type float.

**Parameters:**value - the value to be represented by the Float.

### Float

public **Float**([String](http://docs.google.com/java/lang/String.html) s)  
 throws [NumberFormatException](http://docs.google.com/java/lang/NumberFormatException.html)

Constructs a newly allocated Float object that represents the floating-point value of type float represented by the string. The string is converted to a float value as if by the valueOf method.

**Parameters:**s - a string to be converted to a Float. **Throws:** [NumberFormatException](http://docs.google.com/java/lang/NumberFormatException.html) - if the string does not contain a parsable number.**See Also:**[valueOf(java.lang.String)](http://docs.google.com/java/lang/Float.html#valueOf(java.lang.String))

| **Method Detail** |
| --- |

### toString

public static [String](http://docs.google.com/java/lang/String.html) **toString**(float f)

Returns a string representation of the float argument. All characters mentioned below are ASCII characters.

* If the argument is NaN, the result is the string "NaN".
* Otherwise, the result is a string that represents the sign and magnitude (absolute value) of the argument. If the sign is negative, the first character of the result is '-' ('\u002D'); if the sign is positive, no sign character appears in the result. As for the magnitude *m*:
  + If *m* is infinity, it is represented by the characters "Infinity"; thus, positive infinity produces the result "Infinity" and negative infinity produces the result "-Infinity".
  + If *m* is zero, it is represented by the characters "0.0"; thus, negative zero produces the result "-0.0" and positive zero produces the result "0.0".
  + If *m* is greater than or equal to 10-3 but less than 107, then it is represented as the integer part of *m*, in decimal form with no leading zeroes, followed by '.' ('\u002E'), followed by one or more decimal digits representing the fractional part of *m*.
  + If *m* is less than 10-3 or greater than or equal to 107, then it is represented in so-called "computerized scientific notation." Let *n* be the unique integer such that 10*n*<= *m* < 10*n*+1; then let *a* be the mathematically exact quotient of *m* and 10*n* so that 1 <= *a* < 10. The magnitude is then represented as the integer part of *a*, as a single decimal digit, followed by '.' ('\u002E'), followed by decimal digits representing the fractional part of *a*, followed by the letter 'E' ('\u0045'), followed by a representation of *n* as a decimal integer, as produced by the method [Integer.toString(int)](http://docs.google.com/java/lang/Integer.html#toString(int)).

How many digits must be printed for the fractional part of *m* or *a*? There must be at least one digit to represent the fractional part, and beyond that as many, but only as many, more digits as are needed to uniquely distinguish the argument value from adjacent values of type float. That is, suppose that *x* is the exact mathematical value represented by the decimal representation produced by this method for a finite nonzero argument *f*. Then *f* must be the float value nearest to *x*; or, if two float values are equally close to *x*, then *f* must be one of them and the least significant bit of the significand of *f* must be 0.

To create localized string representations of a floating-point value, use subclasses of [NumberFormat](http://docs.google.com/java/text/NumberFormat.html).

**Parameters:**f - the float to be converted. **Returns:**a string representation of the argument.

### toHexString

public static [String](http://docs.google.com/java/lang/String.html) **toHexString**(float f)

Returns a hexadecimal string representation of the float argument. All characters mentioned below are ASCII characters.

* If the argument is NaN, the result is the string "NaN".
* Otherwise, the result is a string that represents the sign and magnitude (absolute value) of the argument. If the sign is negative, the first character of the result is '-' ('\u002D'); if the sign is positive, no sign character appears in the result. As for the magnitude *m*:
  + If *m* is infinity, it is represented by the string "Infinity"; thus, positive infinity produces the result "Infinity" and negative infinity produces the result "-Infinity".
  + If *m* is zero, it is represented by the string "0x0.0p0"; thus, negative zero produces the result "-0x0.0p0" and positive zero produces the result "0x0.0p0".
  + If *m* is a float value with a normalized representation, substrings are used to represent the significand and exponent fields. The significand is represented by the characters "0x1." followed by a lowercase hexadecimal representation of the rest of the significand as a fraction. Trailing zeros in the hexadecimal representation are removed unless all the digits are zero, in which case a single zero is used. Next, the exponent is represented by "p" followed by a decimal string of the unbiased exponent as if produced by a call to [Integer.toString](http://docs.google.com/java/lang/Integer.html#toString(int)) on the exponent value.
  + If *m* is a float value with a subnormal representation, the significand is represented by the characters "0x0." followed by a hexadecimal representation of the rest of the significand as a fraction. Trailing zeros in the hexadecimal representation are removed. Next, the exponent is represented by "p-126". Note that there must be at least one nonzero digit in a subnormal significand.

### Examples

| Floating-point Value | Hexadecimal String |
| --- | --- |
| 1.0 | 0x1.0p0 |
| -1.0 | -0x1.0p0 |
| 2.0 | 0x1.0p1 |
| 3.0 | 0x1.8p1 |
| 0.5 | 0x1.0p-1 |
| 0.25 | 0x1.0p-2 |
| Float.MAX\_VALUE | 0x1.fffffep127 |
| Minimum Normal Value | 0x1.0p-126 |
| Maximum Subnormal Value | 0x0.fffffep-126 |
| Float.MIN\_VALUE | 0x0.000002p-126 |

**Parameters:**f - the float to be converted. **Returns:**a hex string representation of the argument.**Since:** 1.5

### valueOf

public static [Float](http://docs.google.com/java/lang/Float.html) **valueOf**([String](http://docs.google.com/java/lang/String.html) s)  
 throws [NumberFormatException](http://docs.google.com/java/lang/NumberFormatException.html)

Returns a Float object holding the float value represented by the argument string s.

If s is null, then a NullPointerException is thrown.

Leading and trailing whitespace characters in s are ignored. Whitespace is removed as if by the [String.trim()](http://docs.google.com/java/lang/String.html#trim()) method; that is, both ASCII space and control characters are removed. The rest of s should constitute a *FloatValue* as described by the lexical syntax rules:

*FloatValue:* *Signopt* NaN *Signopt* Infinity *Signopt FloatingPointLiteral* *Signopt HexFloatingPointLiteral* *SignedInteger*

*HexFloatingPointLiteral*: *HexSignificand BinaryExponent FloatTypeSuffixopt*

*HexSignificand:* *HexNumeral* *HexNumeral* . 0x *HexDigitsopt* . *HexDigits* 0X *HexDigitsopt* . *HexDigits*

*BinaryExponent:* *BinaryExponentIndicator SignedInteger*

*BinaryExponentIndicator:* p Pwhere *Sign*, *FloatingPointLiteral*, *HexNumeral*, *HexDigits*, *SignedInteger* and *FloatTypeSuffix* are as defined in the lexical structure sections of the of the [Java Language Specification](http://java.sun.com/docs/books/jls/html/). If s does not have the form of a *FloatValue*, then a NumberFormatException is thrown. Otherwise, s is regarded as representing an exact decimal value in the usual "computerized scientific notation" or as an exact hexadecimal value; this exact numerical value is then conceptually converted to an "infinitely precise" binary value that is then rounded to type float by the usual round-to-nearest rule of IEEE 754 floating-point arithmetic, which includes preserving the sign of a zero value. Finally, a Float object representing this float value is returned.

To interpret localized string representations of a floating-point value, use subclasses of [NumberFormat](http://docs.google.com/java/text/NumberFormat.html).

Note that trailing format specifiers, specifiers that determine the type of a floating-point literal (1.0f is a float value; 1.0d is a double value), do *not* influence the results of this method. In other words, the numerical value of the input string is converted directly to the target floating-point type. In general, the two-step sequence of conversions, string to double followed by double to float, is *not* equivalent to converting a string directly to float. For example, if first converted to an intermediate double and then to float, the string

"1.00000017881393421514957253748434595763683319091796875001d"

results in the float value 1.0000002f; if the string is converted directly to float, 1.000000**1**f results.

To avoid calling this method on an invalid string and having a NumberFormatException be thrown, the documentation for [Double.valueOf](http://docs.google.com/java/lang/Double.html#valueOf(java.lang.String)) lists a regular expression which can be used to screen the input.

**Parameters:**s - the string to be parsed. **Returns:**a Float object holding the value represented by the String argument. **Throws:** [NumberFormatException](http://docs.google.com/java/lang/NumberFormatException.html) - if the string does not contain a parsable number.

### valueOf

public static [Float](http://docs.google.com/java/lang/Float.html) **valueOf**(float f)

Returns a Float instance representing the specified float value. If a new Float instance is not required, this method should generally be used in preference to the constructor [Float(float)](http://docs.google.com/java/lang/Float.html#Float(float)), as this method is likely to yield significantly better space and time performance by caching frequently requested values.

**Parameters:**f - a float value. **Returns:**a Float instance representing f.**Since:** 1.5

### parseFloat

public static float **parseFloat**([String](http://docs.google.com/java/lang/String.html) s)  
 throws [NumberFormatException](http://docs.google.com/java/lang/NumberFormatException.html)

Returns a new float initialized to the value represented by the specified String, as performed by the valueOf method of class Float.

**Parameters:**s - the string to be parsed. **Returns:**the float value represented by the string argument. **Throws:** [NumberFormatException](http://docs.google.com/java/lang/NumberFormatException.html) - if the string does not contain a parsable float.**Since:** 1.2 **See Also:**[valueOf(String)](http://docs.google.com/java/lang/Float.html#valueOf(java.lang.String))

### isNaN

public static boolean **isNaN**(float v)

Returns true if the specified number is a Not-a-Number (NaN) value, false otherwise.

**Parameters:**v - the value to be tested. **Returns:**true if the argument is NaN; false otherwise.

### isInfinite

public static boolean **isInfinite**(float v)

Returns true if the specified number is infinitely large in magnitude, false otherwise.

**Parameters:**v - the value to be tested. **Returns:**true if the argument is positive infinity or negative infinity; false otherwise.

### isNaN

public boolean **isNaN**()

Returns true if this Float value is a Not-a-Number (NaN), false otherwise.

**Returns:**true if the value represented by this object is NaN; false otherwise.

### isInfinite

public boolean **isInfinite**()

Returns true if this Float value is infinitely large in magnitude, false otherwise.

**Returns:**true if the value represented by this object is positive infinity or negative infinity; false otherwise.

### toString

public [String](http://docs.google.com/java/lang/String.html) **toString**()

Returns a string representation of this Float object. The primitive float value represented by this object is converted to a String exactly as if by the method toString of one argument.

**Overrides:**[toString](http://docs.google.com/java/lang/Object.html#toString()) in class [Object](http://docs.google.com/java/lang/Object.html) **Returns:**a String representation of this object.**See Also:**[toString(float)](http://docs.google.com/java/lang/Float.html#toString(float))

### byteValue

public byte **byteValue**()

Returns the value of this Float as a byte (by casting to a byte).

**Overrides:**[byteValue](http://docs.google.com/java/lang/Number.html#byteValue()) in class [Number](http://docs.google.com/java/lang/Number.html) **Returns:**the float value represented by this object converted to type byte

### shortValue

public short **shortValue**()

Returns the value of this Float as a short (by casting to a short).

**Overrides:**[shortValue](http://docs.google.com/java/lang/Number.html#shortValue()) in class [Number](http://docs.google.com/java/lang/Number.html) **Returns:**the float value represented by this object converted to type short**Since:** JDK1.1

### intValue

public int **intValue**()

Returns the value of this Float as an int (by casting to type int).

**Specified by:**[intValue](http://docs.google.com/java/lang/Number.html#intValue()) in class [Number](http://docs.google.com/java/lang/Number.html) **Returns:**the float value represented by this object converted to type int

### longValue

public long **longValue**()

Returns value of this Float as a long (by casting to type long).

**Specified by:**[longValue](http://docs.google.com/java/lang/Number.html#longValue()) in class [Number](http://docs.google.com/java/lang/Number.html) **Returns:**the float value represented by this object converted to type long

### floatValue

public float **floatValue**()

Returns the float value of this Float object.

**Specified by:**[floatValue](http://docs.google.com/java/lang/Number.html#floatValue()) in class [Number](http://docs.google.com/java/lang/Number.html) **Returns:**the float value represented by this object

### doubleValue

public double **doubleValue**()

Returns the double value of this Float object.

**Specified by:**[doubleValue](http://docs.google.com/java/lang/Number.html#doubleValue()) in class [Number](http://docs.google.com/java/lang/Number.html) **Returns:**the float value represented by this object is converted to type double and the result of the conversion is returned.

### hashCode

public int **hashCode**()

Returns a hash code for this Float object. The result is the integer bit representation, exactly as produced by the method [floatToIntBits(float)](http://docs.google.com/java/lang/Float.html#floatToIntBits(float)), of the primitive float value represented by this Float object.

**Overrides:**[hashCode](http://docs.google.com/java/lang/Object.html#hashCode()) in class [Object](http://docs.google.com/java/lang/Object.html) **Returns:**a hash code value for this object.**See Also:**[Object.equals(java.lang.Object)](http://docs.google.com/java/lang/Object.html#equals(java.lang.Object)), [Hashtable](http://docs.google.com/java/util/Hashtable.html)

### equals

public boolean **equals**([Object](http://docs.google.com/java/lang/Object.html) obj)

Compares this object against the specified object. The result is true if and only if the argument is not null and is a Float object that represents a float with the same value as the float represented by this object. For this purpose, two float values are considered to be the same if and only if the method [floatToIntBits(float)](http://docs.google.com/java/lang/Float.html#floatToIntBits(float)) returns the identical int value when applied to each.

Note that in most cases, for two instances of class Float, f1 and f2, the value of f1.equals(f2) is true if and only if

f1.floatValue() == f2.floatValue()

also has the value true. However, there are two exceptions:

* If f1 and f2 both represent Float.NaN, then the equals method returns true, even though Float.NaN==Float.NaN has the value false.
* If f1 represents +0.0f while f2 represents -0.0f, or vice versa, the equal test has the value false, even though 0.0f==-0.0f has the value true.

This definition allows hash tables to operate properly.

**Overrides:**[equals](http://docs.google.com/java/lang/Object.html#equals(java.lang.Object)) in class [Object](http://docs.google.com/java/lang/Object.html) **Parameters:**obj - the object to be compared **Returns:**true if the objects are the same; false otherwise.**See Also:**[floatToIntBits(float)](http://docs.google.com/java/lang/Float.html#floatToIntBits(float))

### floatToIntBits

public static int **floatToIntBits**(float value)

Returns a representation of the specified floating-point value according to the IEEE 754 floating-point "single format" bit layout.

Bit 31 (the bit that is selected by the mask 0x80000000) represents the sign of the floating-point number. Bits 30-23 (the bits that are selected by the mask 0x7f800000) represent the exponent. Bits 22-0 (the bits that are selected by the mask 0x007fffff) represent the significand (sometimes called the mantissa) of the floating-point number.

If the argument is positive infinity, the result is 0x7f800000.

If the argument is negative infinity, the result is 0xff800000.

If the argument is NaN, the result is 0x7fc00000.

In all cases, the result is an integer that, when given to the [intBitsToFloat(int)](http://docs.google.com/java/lang/Float.html#intBitsToFloat(int)) method, will produce a floating-point value the same as the argument to floatToIntBits (except all NaN values are collapsed to a single "canonical" NaN value).

**Parameters:**value - a floating-point number. **Returns:**the bits that represent the floating-point number.

### floatToRawIntBits

public static int **floatToRawIntBits**(float value)

Returns a representation of the specified floating-point value according to the IEEE 754 floating-point "single format" bit layout, preserving Not-a-Number (NaN) values.

Bit 31 (the bit that is selected by the mask 0x80000000) represents the sign of the floating-point number. Bits 30-23 (the bits that are selected by the mask 0x7f800000) represent the exponent. Bits 22-0 (the bits that are selected by the mask 0x007fffff) represent the significand (sometimes called the mantissa) of the floating-point number.

If the argument is positive infinity, the result is 0x7f800000.

If the argument is negative infinity, the result is 0xff800000.

If the argument is NaN, the result is the integer representing the actual NaN value. Unlike the floatToIntBits method, floatToRawIntBits does not collapse all the bit patterns encoding a NaN to a single "canonical" NaN value.

In all cases, the result is an integer that, when given to the [intBitsToFloat(int)](http://docs.google.com/java/lang/Float.html#intBitsToFloat(int)) method, will produce a floating-point value the same as the argument to floatToRawIntBits.

**Parameters:**value - a floating-point number. **Returns:**the bits that represent the floating-point number.**Since:** 1.3

### intBitsToFloat

public static float **intBitsToFloat**(int bits)

Returns the float value corresponding to a given bit representation. The argument is considered to be a representation of a floating-point value according to the IEEE 754 floating-point "single format" bit layout.

If the argument is 0x7f800000, the result is positive infinity.

If the argument is 0xff800000, the result is negative infinity.

If the argument is any value in the range 0x7f800001 through 0x7fffffff or in the range 0xff800001 through 0xffffffff, the result is a NaN. No IEEE 754 floating-point operation provided by Java can distinguish between two NaN values of the same type with different bit patterns. Distinct values of NaN are only distinguishable by use of the Float.floatToRawIntBits method.

In all other cases, let *s*, *e*, and *m* be three values that can be computed from the argument:

int s = ((bits >> 31) == 0) ? 1 : -1;  
 int e = ((bits >> 23) & 0xff);  
 int m = (e == 0) ?  
 (bits & 0x7fffff) << 1 :  
 (bits & 0x7fffff) | 0x800000;

Then the floating-point result equals the value of the mathematical expression *s*·*m*·2*e*-150.

Note that this method may not be able to return a float NaN with exactly same bit pattern as the int argument. IEEE 754 distinguishes between two kinds of NaNs, quiet NaNs and *signaling NaNs*. The differences between the two kinds of NaN are generally not visible in Java. Arithmetic operations on signaling NaNs turn them into quiet NaNs with a different, but often similar, bit pattern. However, on some processors merely copying a signaling NaN also performs that conversion. In particular, copying a signaling NaN to return it to the calling method may perform this conversion. So intBitsToFloat may not be able to return a float with a signaling NaN bit pattern. Consequently, for some int values, floatToRawIntBits(intBitsToFloat(start)) may *not* equal start. Moreover, which particular bit patterns represent signaling NaNs is platform dependent; although all NaN bit patterns, quiet or signaling, must be in the NaN range identified above.

**Parameters:**bits - an integer. **Returns:**the float floating-point value with the same bit pattern.

### compareTo

public int **compareTo**([Float](http://docs.google.com/java/lang/Float.html) anotherFloat)

Compares two Float objects numerically. There are two ways in which comparisons performed by this method differ from those performed by the Java language numerical comparison operators (<, <=, ==, >= >) when applied to primitive float values:

* Float.NaN is considered by this method to be equal to itself and greater than all other float values (including Float.POSITIVE\_INFINITY).
* 0.0f is considered by this method to be greater than -0.0f.

This ensures that the *natural ordering* of Float objects imposed by this method is *consistent with equals*.

**Specified by:**[compareTo](http://docs.google.com/java/lang/Comparable.html#compareTo(T)) in interface [Comparable](http://docs.google.com/java/lang/Comparable.html)<[Float](http://docs.google.com/java/lang/Float.html)> **Parameters:**anotherFloat - the Float to be compared. **Returns:**the value 0 if anotherFloat is numerically equal to this Float; a value less than 0 if this Float is numerically less than anotherFloat; and a value greater than 0 if this Float is numerically greater than anotherFloat.**Since:** 1.2 **See Also:**[Comparable.compareTo(Object)](http://docs.google.com/java/lang/Comparable.html#compareTo(T))

### compare

public static int **compare**(float f1,  
 float f2)

Compares the two specified float values. The sign of the integer value returned is the same as that of the integer that would be returned by the call:

new Float(f1).compareTo(new Float(f2))

**Parameters:**f1 - the first float to compare.f2 - the second float to compare. **Returns:**the value 0 if f1 is numerically equal to f2; a value less than 0 if f1 is numerically less than f2; and a value greater than 0 if f1 is numerically greater than f2.**Since:** 1.4

| | [**Overview**](http://docs.google.com/overview-summary.html) | [**Package**](http://docs.google.com/package-summary.html) | **Class** | [**Use**](http://docs.google.com/class-use/Float.html) | [**Tree**](http://docs.google.com/package-tree.html) | [**Deprecated**](http://docs.google.com/deprecated-list.html) | [**Index**](http://docs.google.com/index-files/index-1.html) | [**Help**](http://docs.google.com/help-doc.html) | | --- | --- | --- | --- | --- | --- | --- | --- | | | ***Java™ Platform***  ***Standard Ed. 6*** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| [**PREV CLASS**](http://docs.google.com/java/lang/ExceptionInInitializerError.html)   [**NEXT CLASS**](http://docs.google.com/java/lang/IllegalAccessError.html) | [**FRAMES**](http://docs.google.com/index.html?java/lang/Float.html)    [**NO FRAMES**](http://docs.google.com/Float.html)     [**All Classes**](http://docs.google.com/allclasses-noframe.html) |
| SUMMARY: NESTED | [FIELD](#3znysh7) | [CONSTR](#2et92p0) | [METHOD](#tyjcwt) | DETAIL: [FIELD](#1t3h5sf) | [CONSTR](#z337ya) | [METHOD](#2xcytpi) |

[Submit a bug or feature](http://bugs.sun.com/services/bugreport/index.jsp)

For further API reference and developer documentation, see [Java SE Developer Documentation](http://docs.google.com/webnotes/devdocs-vs-specs.html). That documentation contains more detailed, developer-targeted descriptions, with conceptual overviews, definitions of terms, workarounds, and working code examples.

Copyright 2006 Sun Microsystems, Inc. All rights reserved. Use is subject to [license terms](http://docs.google.com/legal/license.html). Also see the [documentation redistribution policy](http://java.sun.com/docs/redist.html).