Java.Lang.Float class in Java

Difficulty Level : Basic Last Updated : 09 Aug, 2021

Float class is a wrapper class for the primitive type float which contains several methods to effectively deal with a float value like converting it to a string representation, and vice-versa. An object of Float class can hold a single float value. There are mainly two constructors to initialize a Float object-

- **Float(float b:** Creates a Float object initialized with the value provided.

```
Syntax: public Float(Float d)
Parameters:

d : value with which to initialize
```

- **Float(String s):** Creates a Float object initialized with the parsed float value provided by string representation. Default radix is taken to be 10.

```
Syntax:  public Float(String s) throws NumberFormatException
Parameters:
s : string representation of the byte value
Throws:
NumberFormatException: If the string provided does not represent any
float value.
```

Methods:

1.toString(): Returns the string corresponding to the float value.

```
Syntax : public String toString(float b)
Parameters :
b : float value for which string representation required.
```

2.valueOf() : returns the Float object initialized with the value provided.

```
Syntax : public static Float valueOf(float b)
Parameters :
b : a float value
```

Another overloaded function valueOf(String val) which provides function similar to new Float(Float.parseFloat(val,10))

```
Syntax : public static Float valueOf(String s)
           throws NumberFormatException
Parameters :
s : a String object to be parsed as float
Throws :
NumberFormatException : if String cannot be parsed to a float value.
```

3.parseFloat() : returns float value by parsing the string. Differs from valueOf() as it returns a primitive float value and valueOf() return Float object.

```
Syntax : public static float parseFloat(String val)
             throws NumberFormatException
Parameters :
val : String representation of float
Throws :
NumberFormatException : if String cannot be parsed to a float value
in given radix.
```

4.byteValue() : returns a byte value corresponding to this Float Object.

```
Syntax : public byte byteValue()
```

5.shortValue() : returns a short value corresponding to this Float Object.

```
Syntax : public short shortValue()
```

6.intValue() : returns a int value corresponding to this Float Object.

```
Syntax : public int intValue()
```

7.longValue() : returns a long value corresponding to this Float Object.

```
Syntax : public long longValue()
```

8.doubleValue() : returns a double value corresponding to this Float Object.

```
Syntax : public double doubleValue()
```

9.floatValue() : returns a float value corresponding to this Float Object.

```
Syntax : public float floatValue()
```

10.hashCode() : returns the hashcode corresponding to this Float Object.

```
Syntax : public int hashCode()
```

11.isNaN() : returns true if the float object in consideration is not a number, otherwise false.

```
Syntax : public boolean isNaN()
```

Another static method isNaN(float val) can be used if we don’t need any object of float to be created. It provides similar functionality as the above version.

```
Syntax : public static boolean isNaN(float val)
Parameters :
val : float value to check for
```

12.isInfinite() : returns true if the float object in consideration is very large, otherwise false. Specifically any number beyond 0x7f800000 on positive side and below 0xff800000 on negative side are the infinity values.

```
Syntax : public boolean isInfinite()
```

Another static method isInfinite(float val) can be used if we don't need any object of float to be created. It provides similar functionality as the above version.

```
Syntax : public static boolean isInfinte(float val)
Parameters :
val : float value to check for
```

13.toHexString() : Returns the hexadecimal representation of the argument float value.

```
Syntax : public static String toHexString(float val)
Parameters :
val : float value to be represented as hex string
```

14. floatToIntBits() : returns the IEEE 754 floating-point "single format" bit layout of the given float argument. Detailed summary of IEEE 754 floating-point "single format" can be found here.

```
Syntax : public static int floatToIntBits(float val)
Parameters :
val : float value to convert
```

15.floatToRawIntBits() : returns the IEEE 754 floating-point "single format" bit layout of the given float argument. It differs from previous method as it preserves the Nan values.

```
Syntax : public static int floatToRawIntBits(float val)
Parameters :
val : float value to convert
```

16.IntBitsToFloat() : Returns the float value corresponding to the long bit pattern of the argument. It does reverse work of the previous two methods.

```
Syntax : public static float IntBitsToFloat(long b)
Parameters :
b : long bit pattern
```

17.equals() : Used to compare the equality of two Float objects. This method returns true if both the objects contain the same float value. Should be used only if checking for equality. In all other cases, compareTo method should be preferred.

```
Syntax : public boolean equals(Object obj)
Parameters :
obj : object to compare with
```

18. compareTo() : Used to compare two Float objects for numerical equality. This should be used when comparing two Float values for numerical equality as it would differentiate between less and greater values. Returns a value less than 0,0,value greater than 0 for less than,equal to and greater than.

```
Syntax : public int compareTo(Float b)
Parameters :
b : Float object to compare with
```

19. compare() : Used to compare two primitive float values for numerical equality. As it is a static method therefore it can be used without creating any object of Float.

```
Syntax : public static int compare(float x,float y)
Parameters :
x : float value
y : another float value
```

Example:

```java
// Java program to illustrate
// various float class methods
// of Java.lang class
public class GfG
{

    public static void main(String[] args)
    {
        float b = 55.05F;
        String bb = "45";

        // Construct two Float objects
        Float x = new Float(b);
        Float y = new Float(bb);

        // toString()
        System.out.println("toString(b) = " + Float.toString(b));

        // valueOf()
        // return Float object
        Float z = Float.valueOf(b);
        System.out.println("valueOf(b) = " + z);
        z = Float.valueOf(bb);
        System.out.println("ValueOf(bb) = " + z);

        // parseFloat()
        // return primitive float value
        float zz = Float.parseFloat(bb);
        System.out.println("parseFloat(bb) = " + zz);

        System.out.println("bytevalue(x) = " + x.byteValue());
        System.out.println("shortvalue(x) = " + x.shortValue());
        System.out.println("intvalue(x) = " + x.intValue());
        System.out.println("longvalue(x) = " + x.longValue());
        System.out.println("doublevalue(x) = " + x.doubleValue());
        System.out.println("floatvalue(x) = " + x.floatValue());

        int hash = x.hashCode();
        System.out.println("hashcode(x) = " + hash);

        boolean eq = x.equals(y);
        System.out.println("x.equals(y) = " + eq);

        int e = Float.compare(x, y);
        System.out.println("compare(x,y) = " + e);

        int f = x.compareTo(y);
        System.out.println("x.compareTo(y) = " + f);

        Float d = Float.valueOf("1010.54789654123654");
        System.out.println("isNaN(d) = " + d.isNaN());

        System.out.println("Float.isNaN(45.12452) = "
                                    + Float.isNaN(45.12452F));

        // Float.POSITIVE_INFINITY stores
        // the positive infinite value
        d = Float.valueOf(Float.POSITIVE_INFINITY + 1);
        System.out.println("Float.isInfinite(d) = "
                            + Float.isInfinite(d.floatValue()));

        float dd = 10245.21452F;
        System.out.println("Float.toString(dd) = "
                                        + Float.toHexString(dd));

        int float_to_int = Float.floatToIntBits(dd);
        System.out.println("Float.floatToLongBits(dd) = "
                                                + float_to_int);

        float int_to_float = Float.intBitsToFloat(float_to_int);
        System.out.println("Float.intBitsToFloat(float_to_long) = "
                                                + int_to_float);
    }

}
```

Output :

```
toString(b) = 55.05
valueOf(b) = 55.05
ValueOf(bb) = 45.0
parseFloat(bb) = 45.0
bytevalue(x) = 55
shortvalue(x) = 55
intvalue(x) = 55
longvalue(x) = 55
doublevalue(x) = 55.04999923706055
floatvalue(x) = 55.05
hashcode(x) = 1113338675
x.equals(y) = false
compare(x,y) = 1
x.compareTo(y) = 1
isNaN(d) = false
Float.isNaN(45.12452) = false
Float.isInfinite(d) = true
Float.toString(dd) = 0x1.4029b8p13
Float.floatToLongBits(dd) = 1176507612
Float.intBitsToFloat(float_to_long) = 10245.215
```

References : Official Java Documentation

This article is contributed by **Rishabh Mahrsee**. If you like GeeksforGeeks and would like to contribute, you can also write an article using write.geeksforgeeks.org or mail your article to review-team@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks.

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.