**Trail:** Learning the Java Language
**Lesson:** Numbers and Strings
**Section:** Strings

*The Java Tutorials have been written for JDK 8. Examples and practices described in this page don't take advantage of improvements introduced in later releases and might use technology no longer available.*
*See Java Language Changes for a summary of updated language features in Java SE 9 and subsequent releases.*
*See JDK Release Notes for information about new features, enhancements, and removed or deprecated options for all JDK releases.*

# Converting Between Numbers and Strings

## Converting Strings to Numbers

Frequently, a program ends up with numeric data in a string object—a value entered by the user, for example.

The `Number` subclasses that wrap primitive numeric types ( `Byte`, `Integer`, `Double`, `Float`, `Long`, and `Short`) each provide a class method named `valueOf` that converts a string to an object of that type. Here is an example, `ValueOfDemo` , that gets two strings from the command line, converts them to numbers, and performs arithmetic operations on the values:

```
public class ValueOfDemo {
    public static void main(String[] args) {

        // this program requires two
        // arguments on the command line
        if (args.length == 2) {
            // convert strings to numbers
            float a = (Float.valueOf(args[0])).floatValue();
            float b = (Float.valueOf(args[1])).floatValue();

            // do some arithmetic
            System.out.println("a + b = " +
                               (a + b));
            System.out.println("a - b = " +
                               (a - b));
            System.out.println("a * b = " +
                               (a * b));
            System.out.println("a / b = " +
                               (a / b));
            System.out.println("a % b = " +
                               (a % b));
        } else {
            System.out.println("This program " +
                "requires two command-line arguments.");
        }
    }
}
```

The following is the output from the program when you use `4.5` and `87.2` for the command-line arguments:

```
a + b = 91.7
a - b = -82.7
a * b = 392.4
a / b = 0.0516055
a % b = 4.5
```

---

**Note:** Each of the `Number` subclasses that wrap primitive numeric types also provides a `parseXXXX()` method (for example, `parseFloat()`) that can be used to convert strings to primitive numbers. Since a primitive type is returned instead of an object, the `parseFloat()` method is more direct than the `valueOf()` method. For example, in the `ValueOfDemo` program, we could use:

```
float a = Float.parseFloat(args[0]);
float b = Float.parseFloat(args[1]);
```

## Converting Numbers to Strings

Sometimes you need to convert a number to a string because you need to operate on the value in its string form. There are several easy ways to convert a number to a string:

```
int i;
// Concatenate "i" with an empty string; conversion is handled for you.
String s1 = "" + i;
```

or

```
// The valueOf class method.
String s2 = String.valueOf(i);
```

Each of the `Number` subclasses includes a class method, `toString()`, that will convert its primitive type to a string. For example:

```
int i;
double d;
String s3 = Integer.toString(i);
String s4 = Double.toString(d);
```

The `ToStringDemo` example uses the `toString` method to convert a number to a string. The program then uses some string methods to compute the number of digits before and after the decimal point:

```
public class ToStringDemo {

    public static void main(String[] args) {
        double d = 858.48;
        String s = Double.toString(d);

        int dot = s.indexOf('.');

        System.out.println(dot + " digits " +
            "before decimal point.");
        System.out.println( (s.length() - dot - 1) +
            " digits after decimal point.");
    }
}
```

The output of this program is:

```
3 digits before decimal point.
2 digits after decimal point.
```



**Previous page:** Strings
**Next page:** Manipulating Characters in a String