| | [**Overview**](http://docs.google.com/overview-summary.html) | [**Package**](http://docs.google.com/package-summary.html) | **Class** | [**Use**](http://docs.google.com/class-use/Scanner.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/util/ResourceBundle.Control.html)   [**NEXT CLASS**](http://docs.google.com/java/util/ServiceConfigurationError.html) | [**FRAMES**](http://docs.google.com/index.html?java/util/Scanner.html)    [**NO FRAMES**](http://docs.google.com/Scanner.html)     [**All Classes**](http://docs.google.com/allclasses-noframe.html) |
| SUMMARY: NESTED | FIELD | [CONSTR](#17dp8vu) | [METHOD](#3rdcrjn) | DETAIL: FIELD | [CONSTR](#lnxbz9) | [METHOD](#2xcytpi) |

## **java.util**

Class Scanner

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

**All Implemented Interfaces:** [Iterator](http://docs.google.com/java/util/Iterator.html)<[String](http://docs.google.com/java/lang/String.html)>

public final class **Scanner**extends [Object](http://docs.google.com/java/lang/Object.html)implements [Iterator](http://docs.google.com/java/util/Iterator.html)<[String](http://docs.google.com/java/lang/String.html)>

A simple text scanner which can parse primitive types and strings using regular expressions.

A Scanner breaks its input into tokens using a delimiter pattern, which by default matches whitespace. The resulting tokens may then be converted into values of different types using the various next methods.

For example, this code allows a user to read a number from System.in:

Scanner sc = new Scanner(System.in);  
 int i = sc.nextInt();

As another example, this code allows long types to be assigned from entries in a file myNumbers:

Scanner sc = new Scanner(new File("myNumbers"));  
 while (sc.hasNextLong()) {  
 long aLong = sc.nextLong();  
 }

The scanner can also use delimiters other than whitespace. This example reads several items in from a string:

String input = "1 fish 2 fish red fish blue fish";  
 Scanner s = new Scanner(input).useDelimiter("\\s\*fish\\s\*");  
 System.out.println(s.nextInt());  
 System.out.println(s.nextInt());  
 System.out.println(s.next());  
 System.out.println(s.next());  
 s.close();

prints the following output:

1  
 2  
 red  
 blue

The same output can be generated with this code, which uses a regular expression to parse all four tokens at once:

String input = "1 fish 2 fish red fish blue fish";  
 Scanner s = new Scanner(input);  
 s.findInLine("(\\d+) fish (\\d+) fish (\\w+) fish (\\w+)");  
 MatchResult result = s.match();  
 for (int i=1; i<=result.groupCount(); i++)  
 System.out.println(result.group(i));  
 s.close();

The default whitespace delimiter used by a scanner is as recognized by [Character](http://docs.google.com/java/lang/Character.html).[isWhitespace](http://docs.google.com/java/lang/Character.html#isWhitespace(char)). The [reset()](http://docs.google.com/java/util/Scanner.html#reset()) method will reset the value of the scanner's delimiter to the default whitespace delimiter regardless of whether it was previously changed.

A scanning operation may block waiting for input.

The [next()](http://docs.google.com/java/util/Scanner.html#next()) and [hasNext()](http://docs.google.com/java/util/Scanner.html#hasNext()) methods and their primitive-type companion methods (such as [nextInt()](http://docs.google.com/java/util/Scanner.html#nextInt()) and [hasNextInt()](http://docs.google.com/java/util/Scanner.html#hasNextInt())) first skip any input that matches the delimiter pattern, and then attempt to return the next token. Both hasNext and next methods may block waiting for further input. Whether a hasNext method blocks has no connection to whether or not its associated next method will block.

The [findInLine(java.lang.String)](http://docs.google.com/java/util/Scanner.html#findInLine(java.lang.String)), [findWithinHorizon(java.lang.String, int)](http://docs.google.com/java/util/Scanner.html#findWithinHorizon(java.lang.String,%20int)), and [skip(java.util.regex.Pattern)](http://docs.google.com/java/util/Scanner.html#skip(java.util.regex.Pattern)) methods operate independently of the delimiter pattern. These methods will attempt to match the specified pattern with no regard to delimiters in the input and thus can be used in special circumstances where delimiters are not relevant. These methods may block waiting for more input.

When a scanner throws an [InputMismatchException](http://docs.google.com/java/util/InputMismatchException.html), the scanner will not pass the token that caused the exception, so that it may be retrieved or skipped via some other method.

Depending upon the type of delimiting pattern, empty tokens may be returned. For example, the pattern "\\s+" will return no empty tokens since it matches multiple instances of the delimiter. The delimiting pattern "\\s" could return empty tokens since it only passes one space at a time.

A scanner can read text from any object which implements the [Readable](http://docs.google.com/java/lang/Readable.html) interface. If an invocation of the underlying readable's [Readable.read(java.nio.CharBuffer)](http://docs.google.com/java/lang/Readable.html#read(java.nio.CharBuffer)) method throws an [IOException](http://docs.google.com/java/io/IOException.html) then the scanner assumes that the end of the input has been reached. The most recent IOException thrown by the underlying readable can be retrieved via the [ioException()](http://docs.google.com/java/util/Scanner.html#ioException()) method.

When a Scanner is closed, it will close its input source if the source implements the [Closeable](http://docs.google.com/java/io/Closeable.html) interface.

A Scanner is not safe for multithreaded use without external synchronization.

Unless otherwise mentioned, passing a null parameter into any method of a Scanner will cause a NullPointerException to be thrown.

A scanner will default to interpreting numbers as decimal unless a different radix has been set by using the [useRadix(int)](http://docs.google.com/java/util/Scanner.html#useRadix(int)) method. The [reset()](http://docs.google.com/java/util/Scanner.html#reset()) method will reset the value of the scanner's radix to 10 regardless of whether it was previously changed.

#### Localized numbers

An instance of this class is capable of scanning numbers in the standard formats as well as in the formats of the scanner's locale. A scanner's initial locale is the value returned by the [Locale.getDefault()](http://docs.google.com/java/util/Locale.html#getDefault()) method; it may be changed via the [useLocale(java.util.Locale)](http://docs.google.com/java/util/Scanner.html#useLocale(java.util.Locale)) method. The [reset()](http://docs.google.com/java/util/Scanner.html#reset()) method will reset the value of the scanner's locale to the initial locale regardless of whether it was previously changed.

The localized formats are defined in terms of the following parameters, which for a particular locale are taken from that locale's [DecimalFormat](http://docs.google.com/java/text/DecimalFormat.html) object, df, and its and [DecimalFormatSymbols](http://docs.google.com/java/text/DecimalFormatSymbols.html) object, dfs.

| *LocalGroupSeparator* | The character used to separate thousands groups, *i.e.,* dfs.[getGroupingSeparator()](http://docs.google.com/java/text/DecimalFormatSymbols.html#getGroupingSeparator()) |
| --- | --- |
| *LocalDecimalSeparator* | The character used for the decimal point, *i.e.,* dfs.[getDecimalSeparator()](http://docs.google.com/java/text/DecimalFormatSymbols.html#getDecimalSeparator()) |
| *LocalPositivePrefix* | The string that appears before a positive number (may be empty), *i.e.,* df.[getPositivePrefix()](http://docs.google.com/java/text/DecimalFormat.html#getPositivePrefix()) |
| *LocalPositiveSuffix* | The string that appears after a positive number (may be empty), *i.e.,* df.[getPositiveSuffix()](http://docs.google.com/java/text/DecimalFormat.html#getPositiveSuffix()) |
| *LocalNegativePrefix* | The string that appears before a negative number (may be empty), *i.e.,* df.[getNegativePrefix()](http://docs.google.com/java/text/DecimalFormat.html#getNegativePrefix()) |
| *LocalNegativeSuffix* | The string that appears after a negative number (may be empty), *i.e.,* df.[getNegativeSuffix()](http://docs.google.com/java/text/DecimalFormat.html#getNegativeSuffix()) |
| *LocalNaN* | The string that represents not-a-number for floating-point values, *i.e.,* dfs.[getNaN()](http://docs.google.com/java/text/DecimalFormatSymbols.html#getNaN()) |
| *LocalInfinity* | The string that represents infinity for floating-point values, *i.e.,* dfs.[getInfinity()](http://docs.google.com/java/text/DecimalFormatSymbols.html#getInfinity()) |

#### Number syntax

The strings that can be parsed as numbers by an instance of this class are specified in terms of the following regular-expression grammar, where Rmax is the highest digit in the radix being used (for example, Rmax is 9 in base 10).

| *NonASCIIDigit*  :: | = A non-ASCII character c for which [Character.isDigit](http://docs.google.com/java/lang/Character.html#isDigit(char))(c) returns true |
| --- | --- |
|  |
| *Non0Digit*  :: | = [1-*Rmax*] | *NonASCIIDigit* |
|  |
| *Digit*  :: | = [0-*Rmax*] | *NonASCIIDigit* |
|  |
| *GroupedNumeral*  :: | | = ( | *Non0Digit* *Digit*? *Digit*? | | --- | --- | |  | ( *LocalGroupSeparator* *Digit* *Digit* *Digit* )+ ) | |
|  |
| *Numeral*  :: | = ( ( *Digit*+ ) | *GroupedNumeral* ) |
|  |
| *Integer*  :: | = ( [-+]? ( *Numeral* ) ) |
|  | | *LocalPositivePrefix* *Numeral* *LocalPositiveSuffix* |
|  | | *LocalNegativePrefix* *Numeral* *LocalNegativeSuffix* |
|  |
| *DecimalNumeral*  :: | = *Numeral* |
|  | | *Numeral* *LocalDecimalSeparator* *Digit*\* |
|  | | *LocalDecimalSeparator* *Digit*+ |
|  |
| *Exponent*  :: | = ( [eE] [+-]? *Digit*+ ) |
|  |
| *Decimal*  :: | = ( [-+]? *DecimalNumeral* *Exponent*? ) |
|  | | *LocalPositivePrefix* *DecimalNumeral* *LocalPositiveSuffix* *Exponent*? |
|  | | *LocalNegativePrefix* *DecimalNumeral* *LocalNegativeSuffix* *Exponent*? |
|  |
| *HexFloat*  :: | = [-+]? 0[xX][0-9a-fA-F]\*\.[0-9a-fA-F]+ ([pP][-+]?[0-9]+)? |
|  |
| *NonNumber*  :: | = NaN | *LocalNan* | Infinity | *LocalInfinity* |
|  |
| *SignedNonNumber*  :: | = ( [-+]? *NonNumber* ) |
|  | | *LocalPositivePrefix* *NonNumber* *LocalPositiveSuffix* |
|  | | *LocalNegativePrefix* *NonNumber* *LocalNegativeSuffix* |
|  |
| *Float*  :: | = *Decimal* |
|  | | *HexFloat* |
|  | | *SignedNonNumber* |

Whitespace is not significant in the above regular expressions.

**Since:** 1.5

| **Constructor Summary** | |
| --- | --- |
| [**Scanner**](http://docs.google.com/java/util/Scanner.html#Scanner(java.io.File))([File](http://docs.google.com/java/io/File.html) source)            Constructs a new Scanner that produces values scanned from the specified file. |
| [**Scanner**](http://docs.google.com/java/util/Scanner.html#Scanner(java.io.File,%20java.lang.String))([File](http://docs.google.com/java/io/File.html) source, [String](http://docs.google.com/java/lang/String.html) charsetName)            Constructs a new Scanner that produces values scanned from the specified file. |
| [**Scanner**](http://docs.google.com/java/util/Scanner.html#Scanner(java.io.InputStream))([InputStream](http://docs.google.com/java/io/InputStream.html) source)            Constructs a new Scanner that produces values scanned from the specified input stream. |
| [**Scanner**](http://docs.google.com/java/util/Scanner.html#Scanner(java.io.InputStream,%20java.lang.String))([InputStream](http://docs.google.com/java/io/InputStream.html) source, [String](http://docs.google.com/java/lang/String.html) charsetName)            Constructs a new Scanner that produces values scanned from the specified input stream. |
| [**Scanner**](http://docs.google.com/java/util/Scanner.html#Scanner(java.lang.Readable))([Readable](http://docs.google.com/java/lang/Readable.html) source)            Constructs a new Scanner that produces values scanned from the specified source. |
| [**Scanner**](http://docs.google.com/java/util/Scanner.html#Scanner(java.nio.channels.ReadableByteChannel))([ReadableByteChannel](http://docs.google.com/java/nio/channels/ReadableByteChannel.html) source)            Constructs a new Scanner that produces values scanned from the specified channel. |
| [**Scanner**](http://docs.google.com/java/util/Scanner.html#Scanner(java.nio.channels.ReadableByteChannel,%20java.lang.String))([ReadableByteChannel](http://docs.google.com/java/nio/channels/ReadableByteChannel.html) source, [String](http://docs.google.com/java/lang/String.html) charsetName)            Constructs a new Scanner that produces values scanned from the specified channel. |
| [**Scanner**](http://docs.google.com/java/util/Scanner.html#Scanner(java.lang.String))([String](http://docs.google.com/java/lang/String.html) source)            Constructs a new Scanner that produces values scanned from the specified string. |

| **Method Summary** | |
| --- | --- |
| void | [**close**](http://docs.google.com/java/util/Scanner.html#close())()            Closes this scanner. |
| [Pattern](http://docs.google.com/java/util/regex/Pattern.html) | [**delimiter**](http://docs.google.com/java/util/Scanner.html#delimiter())()            Returns the Pattern this Scanner is currently using to match delimiters. |
| [String](http://docs.google.com/java/lang/String.html) | [**findInLine**](http://docs.google.com/java/util/Scanner.html#findInLine(java.util.regex.Pattern))([Pattern](http://docs.google.com/java/util/regex/Pattern.html) pattern)            Attempts to find the next occurrence of the specified pattern ignoring delimiters. |
| [String](http://docs.google.com/java/lang/String.html) | [**findInLine**](http://docs.google.com/java/util/Scanner.html#findInLine(java.lang.String))([String](http://docs.google.com/java/lang/String.html) pattern)            Attempts to find the next occurrence of a pattern constructed from the specified string, ignoring delimiters. |
| [String](http://docs.google.com/java/lang/String.html) | [**findWithinHorizon**](http://docs.google.com/java/util/Scanner.html#findWithinHorizon(java.util.regex.Pattern,%20int))([Pattern](http://docs.google.com/java/util/regex/Pattern.html) pattern, int horizon)            Attempts to find the next occurrence of the specified pattern. |
| [String](http://docs.google.com/java/lang/String.html) | [**findWithinHorizon**](http://docs.google.com/java/util/Scanner.html#findWithinHorizon(java.lang.String,%20int))([String](http://docs.google.com/java/lang/String.html) pattern, int horizon)            Attempts to find the next occurrence of a pattern constructed from the specified string, ignoring delimiters. |
| boolean | [**hasNext**](http://docs.google.com/java/util/Scanner.html#hasNext())()            Returns true if this scanner has another token in its input. |
| boolean | [**hasNext**](http://docs.google.com/java/util/Scanner.html#hasNext(java.util.regex.Pattern))([Pattern](http://docs.google.com/java/util/regex/Pattern.html) pattern)            Returns true if the next complete token matches the specified pattern. |
| boolean | [**hasNext**](http://docs.google.com/java/util/Scanner.html#hasNext(java.lang.String))([String](http://docs.google.com/java/lang/String.html) pattern)            Returns true if the next token matches the pattern constructed from the specified string. |
| boolean | [**hasNextBigDecimal**](http://docs.google.com/java/util/Scanner.html#hasNextBigDecimal())()            Returns true if the next token in this scanner's input can be interpreted as a BigDecimal using the [nextBigDecimal()](http://docs.google.com/java/util/Scanner.html#nextBigDecimal()) method. |
| boolean | [**hasNextBigInteger**](http://docs.google.com/java/util/Scanner.html#hasNextBigInteger())()            Returns true if the next token in this scanner's input can be interpreted as a BigInteger in the default radix using the [nextBigInteger()](http://docs.google.com/java/util/Scanner.html#nextBigInteger()) method. |
| boolean | [**hasNextBigInteger**](http://docs.google.com/java/util/Scanner.html#hasNextBigInteger(int))(int radix)            Returns true if the next token in this scanner's input can be interpreted as a BigInteger in the specified radix using the [nextBigInteger()](http://docs.google.com/java/util/Scanner.html#nextBigInteger()) method. |
| boolean | [**hasNextBoolean**](http://docs.google.com/java/util/Scanner.html#hasNextBoolean())()            Returns true if the next token in this scanner's input can be interpreted as a boolean value using a case insensitive pattern created from the string "true|false". |
| boolean | [**hasNextByte**](http://docs.google.com/java/util/Scanner.html#hasNextByte())()            Returns true if the next token in this scanner's input can be interpreted as a byte value in the default radix using the [nextByte()](http://docs.google.com/java/util/Scanner.html#nextByte()) method. |
| boolean | [**hasNextByte**](http://docs.google.com/java/util/Scanner.html#hasNextByte(int))(int radix)            Returns true if the next token in this scanner's input can be interpreted as a byte value in the specified radix using the [nextByte()](http://docs.google.com/java/util/Scanner.html#nextByte()) method. |
| boolean | [**hasNextDouble**](http://docs.google.com/java/util/Scanner.html#hasNextDouble())()            Returns true if the next token in this scanner's input can be interpreted as a double value using the [nextDouble()](http://docs.google.com/java/util/Scanner.html#nextDouble()) method. |
| boolean | [**hasNextFloat**](http://docs.google.com/java/util/Scanner.html#hasNextFloat())()            Returns true if the next token in this scanner's input can be interpreted as a float value using the [nextFloat()](http://docs.google.com/java/util/Scanner.html#nextFloat()) method. |
| boolean | [**hasNextInt**](http://docs.google.com/java/util/Scanner.html#hasNextInt())()            Returns true if the next token in this scanner's input can be interpreted as an int value in the default radix using the [nextInt()](http://docs.google.com/java/util/Scanner.html#nextInt()) method. |
| boolean | [**hasNextInt**](http://docs.google.com/java/util/Scanner.html#hasNextInt(int))(int radix)            Returns true if the next token in this scanner's input can be interpreted as an int value in the specified radix using the [nextInt()](http://docs.google.com/java/util/Scanner.html#nextInt()) method. |
| boolean | [**hasNextLine**](http://docs.google.com/java/util/Scanner.html#hasNextLine())()            Returns true if there is another line in the input of this scanner. |
| boolean | [**hasNextLong**](http://docs.google.com/java/util/Scanner.html#hasNextLong())()            Returns true if the next token in this scanner's input can be interpreted as a long value in the default radix using the [nextLong()](http://docs.google.com/java/util/Scanner.html#nextLong()) method. |
| boolean | [**hasNextLong**](http://docs.google.com/java/util/Scanner.html#hasNextLong(int))(int radix)            Returns true if the next token in this scanner's input can be interpreted as a long value in the specified radix using the [nextLong()](http://docs.google.com/java/util/Scanner.html#nextLong()) method. |
| boolean | [**hasNextShort**](http://docs.google.com/java/util/Scanner.html#hasNextShort())()            Returns true if the next token in this scanner's input can be interpreted as a short value in the default radix using the [nextShort()](http://docs.google.com/java/util/Scanner.html#nextShort()) method. |
| boolean | [**hasNextShort**](http://docs.google.com/java/util/Scanner.html#hasNextShort(int))(int radix)            Returns true if the next token in this scanner's input can be interpreted as a short value in the specified radix using the [nextShort()](http://docs.google.com/java/util/Scanner.html#nextShort()) method. |
| [IOException](http://docs.google.com/java/io/IOException.html) | [**ioException**](http://docs.google.com/java/util/Scanner.html#ioException())()            Returns the IOException last thrown by this Scanner's underlying Readable. |
| [Locale](http://docs.google.com/java/util/Locale.html) | [**locale**](http://docs.google.com/java/util/Scanner.html#locale())()            Returns this scanner's locale. |
| [MatchResult](http://docs.google.com/java/util/regex/MatchResult.html) | [**match**](http://docs.google.com/java/util/Scanner.html#match())()            Returns the match result of the last scanning operation performed by this scanner. |
| [String](http://docs.google.com/java/lang/String.html) | [**next**](http://docs.google.com/java/util/Scanner.html#next())()            Finds and returns the next complete token from this scanner. |
| [String](http://docs.google.com/java/lang/String.html) | [**next**](http://docs.google.com/java/util/Scanner.html#next(java.util.regex.Pattern))([Pattern](http://docs.google.com/java/util/regex/Pattern.html) pattern)            Returns the next token if it matches the specified pattern. |
| [String](http://docs.google.com/java/lang/String.html) | [**next**](http://docs.google.com/java/util/Scanner.html#next(java.lang.String))([String](http://docs.google.com/java/lang/String.html) pattern)            Returns the next token if it matches the pattern constructed from the specified string. |
| [BigDecimal](http://docs.google.com/java/math/BigDecimal.html) | [**nextBigDecimal**](http://docs.google.com/java/util/Scanner.html#nextBigDecimal())()            Scans the next token of the input as a [BigDecimal](http://docs.google.com/java/math/BigDecimal.html). |
| [BigInteger](http://docs.google.com/java/math/BigInteger.html) | [**nextBigInteger**](http://docs.google.com/java/util/Scanner.html#nextBigInteger())()            Scans the next token of the input as a [BigInteger](http://docs.google.com/java/math/BigInteger.html). |
| [BigInteger](http://docs.google.com/java/math/BigInteger.html) | [**nextBigInteger**](http://docs.google.com/java/util/Scanner.html#nextBigInteger(int))(int radix)            Scans the next token of the input as a [BigInteger](http://docs.google.com/java/math/BigInteger.html). |
| boolean | [**nextBoolean**](http://docs.google.com/java/util/Scanner.html#nextBoolean())()            Scans the next token of the input into a boolean value and returns that value. |
| byte | [**nextByte**](http://docs.google.com/java/util/Scanner.html#nextByte())()            Scans the next token of the input as a byte. |
| byte | [**nextByte**](http://docs.google.com/java/util/Scanner.html#nextByte(int))(int radix)            Scans the next token of the input as a byte. |
| double | [**nextDouble**](http://docs.google.com/java/util/Scanner.html#nextDouble())()            Scans the next token of the input as a double. |
| float | [**nextFloat**](http://docs.google.com/java/util/Scanner.html#nextFloat())()            Scans the next token of the input as a float. |
| int | [**nextInt**](http://docs.google.com/java/util/Scanner.html#nextInt())()            Scans the next token of the input as an int. |
| int | [**nextInt**](http://docs.google.com/java/util/Scanner.html#nextInt(int))(int radix)            Scans the next token of the input as an int. |
| [String](http://docs.google.com/java/lang/String.html) | [**nextLine**](http://docs.google.com/java/util/Scanner.html#nextLine())()            Advances this scanner past the current line and returns the input that was skipped. |
| long | [**nextLong**](http://docs.google.com/java/util/Scanner.html#nextLong())()            Scans the next token of the input as a long. |
| long | [**nextLong**](http://docs.google.com/java/util/Scanner.html#nextLong(int))(int radix)            Scans the next token of the input as a long. |
| short | [**nextShort**](http://docs.google.com/java/util/Scanner.html#nextShort())()            Scans the next token of the input as a short. |
| short | [**nextShort**](http://docs.google.com/java/util/Scanner.html#nextShort(int))(int radix)            Scans the next token of the input as a short. |
| int | [**radix**](http://docs.google.com/java/util/Scanner.html#radix())()            Returns this scanner's default radix. |
| void | [**remove**](http://docs.google.com/java/util/Scanner.html#remove())()            The remove operation is not supported by this implementation of Iterator. |
| [Scanner](http://docs.google.com/java/util/Scanner.html) | [**reset**](http://docs.google.com/java/util/Scanner.html#reset())()            Resets this scanner. |
| [Scanner](http://docs.google.com/java/util/Scanner.html) | [**skip**](http://docs.google.com/java/util/Scanner.html#skip(java.util.regex.Pattern))([Pattern](http://docs.google.com/java/util/regex/Pattern.html) pattern)            Skips input that matches the specified pattern, ignoring delimiters. |
| [Scanner](http://docs.google.com/java/util/Scanner.html) | [**skip**](http://docs.google.com/java/util/Scanner.html#skip(java.lang.String))([String](http://docs.google.com/java/lang/String.html) pattern)            Skips input that matches a pattern constructed from the specified string. |
| [String](http://docs.google.com/java/lang/String.html) | [**toString**](http://docs.google.com/java/util/Scanner.html#toString())()            Returns the string representation of this Scanner. |
| [Scanner](http://docs.google.com/java/util/Scanner.html) | [**useDelimiter**](http://docs.google.com/java/util/Scanner.html#useDelimiter(java.util.regex.Pattern))([Pattern](http://docs.google.com/java/util/regex/Pattern.html) pattern)            Sets this scanner's delimiting pattern to the specified pattern. |
| [Scanner](http://docs.google.com/java/util/Scanner.html) | [**useDelimiter**](http://docs.google.com/java/util/Scanner.html#useDelimiter(java.lang.String))([String](http://docs.google.com/java/lang/String.html) pattern)            Sets this scanner's delimiting pattern to a pattern constructed from the specified String. |
| [Scanner](http://docs.google.com/java/util/Scanner.html) | [**useLocale**](http://docs.google.com/java/util/Scanner.html#useLocale(java.util.Locale))([Locale](http://docs.google.com/java/util/Locale.html) locale)            Sets this scanner's locale to the specified locale. |
| [Scanner](http://docs.google.com/java/util/Scanner.html) | [**useRadix**](http://docs.google.com/java/util/Scanner.html#useRadix(int))(int radix)            Sets this scanner's default radix to the specified radix. |

| **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()), [equals](http://docs.google.com/java/lang/Object.html#equals(java.lang.Object)), [finalize](http://docs.google.com/java/lang/Object.html#finalize()), [getClass](http://docs.google.com/java/lang/Object.html#getClass()), [hashCode](http://docs.google.com/java/lang/Object.html#hashCode()), [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)) |

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

### Scanner

public **Scanner**([Readable](http://docs.google.com/java/lang/Readable.html) source)

Constructs a new Scanner that produces values scanned from the specified source.

**Parameters:**source - A character source implementing the [Readable](http://docs.google.com/java/lang/Readable.html) interface

### Scanner

public **Scanner**([InputStream](http://docs.google.com/java/io/InputStream.html) source)

Constructs a new Scanner that produces values scanned from the specified input stream. Bytes from the stream are converted into characters using the underlying platform's [default charset](http://docs.google.com/java/nio/charset/Charset.html#defaultCharset()).

**Parameters:**source - An input stream to be scanned

### Scanner

public **Scanner**([InputStream](http://docs.google.com/java/io/InputStream.html) source,  
 [String](http://docs.google.com/java/lang/String.html) charsetName)

Constructs a new Scanner that produces values scanned from the specified input stream. Bytes from the stream are converted into characters using the specified charset.

**Parameters:**source - An input stream to be scannedcharsetName - The encoding type used to convert bytes from the stream into characters to be scanned **Throws:** [IllegalArgumentException](http://docs.google.com/java/lang/IllegalArgumentException.html) - if the specified character set does not exist

### Scanner

public **Scanner**([File](http://docs.google.com/java/io/File.html) source)  
 throws [FileNotFoundException](http://docs.google.com/java/io/FileNotFoundException.html)

Constructs a new Scanner that produces values scanned from the specified file. Bytes from the file are converted into characters using the underlying platform's [default charset](http://docs.google.com/java/nio/charset/Charset.html#defaultCharset()).

**Parameters:**source - A file to be scanned **Throws:** [FileNotFoundException](http://docs.google.com/java/io/FileNotFoundException.html) - if source is not found

### Scanner

public **Scanner**([File](http://docs.google.com/java/io/File.html) source,  
 [String](http://docs.google.com/java/lang/String.html) charsetName)  
 throws [FileNotFoundException](http://docs.google.com/java/io/FileNotFoundException.html)

Constructs a new Scanner that produces values scanned from the specified file. Bytes from the file are converted into characters using the specified charset.

**Parameters:**source - A file to be scannedcharsetName - The encoding type used to convert bytes from the file into characters to be scanned **Throws:** [FileNotFoundException](http://docs.google.com/java/io/FileNotFoundException.html) - if source is not found [IllegalArgumentException](http://docs.google.com/java/lang/IllegalArgumentException.html) - if the specified encoding is not found

### Scanner

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

Constructs a new Scanner that produces values scanned from the specified string.

**Parameters:**source - A string to scan

### Scanner

public **Scanner**([ReadableByteChannel](http://docs.google.com/java/nio/channels/ReadableByteChannel.html) source)

Constructs a new Scanner that produces values scanned from the specified channel. Bytes from the source are converted into characters using the underlying platform's [default charset](http://docs.google.com/java/nio/charset/Charset.html#defaultCharset()).

**Parameters:**source - A channel to scan

### Scanner

public **Scanner**([ReadableByteChannel](http://docs.google.com/java/nio/channels/ReadableByteChannel.html) source,  
 [String](http://docs.google.com/java/lang/String.html) charsetName)

Constructs a new Scanner that produces values scanned from the specified channel. Bytes from the source are converted into characters using the specified charset.

**Parameters:**source - A channel to scancharsetName - The encoding type used to convert bytes from the channel into characters to be scanned **Throws:** [IllegalArgumentException](http://docs.google.com/java/lang/IllegalArgumentException.html) - if the specified character set does not exist

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

### close

public void **close**()

Closes this scanner.

If this scanner has not yet been closed then if its underlying [readable](http://docs.google.com/java/lang/Readable.html) also implements the [Closeable](http://docs.google.com/java/io/Closeable.html) interface then the readable's close method will be invoked. If this scanner is already closed then invoking this method will have no effect.

Attempting to perform search operations after a scanner has been closed will result in an [IllegalStateException](http://docs.google.com/java/lang/IllegalStateException.html).

### ioException

public [IOException](http://docs.google.com/java/io/IOException.html) **ioException**()

Returns the IOException last thrown by this Scanner's underlying Readable. This method returns null if no such exception exists.

**Returns:**the last exception thrown by this scanner's readable

### delimiter

public [Pattern](http://docs.google.com/java/util/regex/Pattern.html) **delimiter**()

Returns the Pattern this Scanner is currently using to match delimiters.

**Returns:**this scanner's delimiting pattern.

### useDelimiter

public [Scanner](http://docs.google.com/java/util/Scanner.html) **useDelimiter**([Pattern](http://docs.google.com/java/util/regex/Pattern.html) pattern)

Sets this scanner's delimiting pattern to the specified pattern.

**Parameters:**pattern - A delimiting pattern **Returns:**this scanner

### useDelimiter

public [Scanner](http://docs.google.com/java/util/Scanner.html) **useDelimiter**([String](http://docs.google.com/java/lang/String.html) pattern)

Sets this scanner's delimiting pattern to a pattern constructed from the specified String.

An invocation of this method of the form useDelimiter(pattern) behaves in exactly the same way as the invocation useDelimiter(Pattern.compile(pattern)).

Invoking the [reset()](http://docs.google.com/java/util/Scanner.html#reset()) method will set the scanner's delimiter to the [default](#3znysh7).

**Parameters:**pattern - A string specifying a delimiting pattern **Returns:**this scanner

### locale

public [Locale](http://docs.google.com/java/util/Locale.html) **locale**()

Returns this scanner's locale.

A scanner's locale affects many elements of its default primitive matching regular expressions; see [localized numbers](#2et92p0) above.

**Returns:**this scanner's locale

### useLocale

public [Scanner](http://docs.google.com/java/util/Scanner.html) **useLocale**([Locale](http://docs.google.com/java/util/Locale.html) locale)

Sets this scanner's locale to the specified locale.

A scanner's locale affects many elements of its default primitive matching regular expressions; see [localized numbers](#2et92p0) above.

Invoking the [reset()](http://docs.google.com/java/util/Scanner.html#reset()) method will set the scanner's locale to the [initial locale](#tyjcwt).

**Parameters:**locale - A string specifying the locale to use **Returns:**this scanner

### radix

public int **radix**()

Returns this scanner's default radix.

A scanner's radix affects elements of its default number matching regular expressions; see [localized numbers](#2et92p0) above.

**Returns:**the default radix of this scanner

### useRadix

public [Scanner](http://docs.google.com/java/util/Scanner.html) **useRadix**(int radix)

Sets this scanner's default radix to the specified radix.

A scanner's radix affects elements of its default number matching regular expressions; see [localized numbers](#2et92p0) above.

If the radix is less than Character.MIN\_RADIX or greater than Character.MAX\_RADIX, then an IllegalArgumentException is thrown.

Invoking the [reset()](http://docs.google.com/java/util/Scanner.html#reset()) method will set the scanner's radix to 10.

**Parameters:**radix - The radix to use when scanning numbers **Returns:**this scanner **Throws:** [IllegalArgumentException](http://docs.google.com/java/lang/IllegalArgumentException.html) - if radix is out of range

### match

public [MatchResult](http://docs.google.com/java/util/regex/MatchResult.html) **match**()

Returns the match result of the last scanning operation performed by this scanner. This method throws IllegalStateException if no match has been performed, or if the last match was not successful.

The various nextmethods of Scanner make a match result available if they complete without throwing an exception. For instance, after an invocation of the [nextInt()](http://docs.google.com/java/util/Scanner.html#nextInt()) method that returned an int, this method returns a MatchResult for the search of the [*Integer*](#1t3h5sf) regular expression defined above. Similarly the [findInLine(java.lang.String)](http://docs.google.com/java/util/Scanner.html#findInLine(java.lang.String)), [findWithinHorizon(java.lang.String, int)](http://docs.google.com/java/util/Scanner.html#findWithinHorizon(java.lang.String,%20int)), and [skip(java.util.regex.Pattern)](http://docs.google.com/java/util/Scanner.html#skip(java.util.regex.Pattern)) methods will make a match available if they succeed.

**Returns:**a match result for the last match operation **Throws:** [IllegalStateException](http://docs.google.com/java/lang/IllegalStateException.html) - If no match result is available

### toString

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

Returns the string representation of this Scanner. The string representation of a Scanner contains information that may be useful for debugging. The exact format is unspecified.

**Overrides:**[toString](http://docs.google.com/java/lang/Object.html#toString()) in class [Object](http://docs.google.com/java/lang/Object.html) **Returns:**The string representation of this scanner

### hasNext

public boolean **hasNext**()

Returns true if this scanner has another token in its input. This method may block while waiting for input to scan. The scanner does not advance past any input.

**Specified by:**[hasNext](http://docs.google.com/java/util/Iterator.html#hasNext()) in interface [Iterator](http://docs.google.com/java/util/Iterator.html)<[String](http://docs.google.com/java/lang/String.html)> **Returns:**true if and only if this scanner has another token **Throws:** [IllegalStateException](http://docs.google.com/java/lang/IllegalStateException.html) - if this scanner is closed**See Also:**[Iterator](http://docs.google.com/java/util/Iterator.html)

### next

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

Finds and returns the next complete token from this scanner. A complete token is preceded and followed by input that matches the delimiter pattern. This method may block while waiting for input to scan, even if a previous invocation of [hasNext()](http://docs.google.com/java/util/Scanner.html#hasNext()) returned true.

**Specified by:**[next](http://docs.google.com/java/util/Iterator.html#next()) in interface [Iterator](http://docs.google.com/java/util/Iterator.html)<[String](http://docs.google.com/java/lang/String.html)> **Returns:**the next token **Throws:** [NoSuchElementException](http://docs.google.com/java/util/NoSuchElementException.html) - if no more tokens are available [IllegalStateException](http://docs.google.com/java/lang/IllegalStateException.html) - if this scanner is closed**See Also:**[Iterator](http://docs.google.com/java/util/Iterator.html)

### remove

public void **remove**()

The remove operation is not supported by this implementation of Iterator.

**Specified by:**[remove](http://docs.google.com/java/util/Iterator.html#remove()) in interface [Iterator](http://docs.google.com/java/util/Iterator.html)<[String](http://docs.google.com/java/lang/String.html)> **Throws:** [UnsupportedOperationException](http://docs.google.com/java/lang/UnsupportedOperationException.html) - if this method is invoked.**See Also:**[Iterator](http://docs.google.com/java/util/Iterator.html)

### hasNext

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

Returns true if the next token matches the pattern constructed from the specified string. The scanner does not advance past any input.

An invocation of this method of the form hasNext(pattern) behaves in exactly the same way as the invocation hasNext(Pattern.compile(pattern)).

**Parameters:**pattern - a string specifying the pattern to scan **Returns:**true if and only if this scanner has another token matching the specified pattern **Throws:** [IllegalStateException](http://docs.google.com/java/lang/IllegalStateException.html) - if this scanner is closed

### next

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

Returns the next token if it matches the pattern constructed from the specified string. If the match is successful, the scanner advances past the input that matched the pattern.

An invocation of this method of the form next(pattern) behaves in exactly the same way as the invocation next(Pattern.compile(pattern)).

**Parameters:**pattern - a string specifying the pattern to scan **Returns:**the next token **Throws:** [NoSuchElementException](http://docs.google.com/java/util/NoSuchElementException.html) - if no such tokens are available [IllegalStateException](http://docs.google.com/java/lang/IllegalStateException.html) - if this scanner is closed

### hasNext

public boolean **hasNext**([Pattern](http://docs.google.com/java/util/regex/Pattern.html) pattern)

Returns true if the next complete token matches the specified pattern. A complete token is prefixed and postfixed by input that matches the delimiter pattern. This method may block while waiting for input. The scanner does not advance past any input.

**Parameters:**pattern - the pattern to scan for **Returns:**true if and only if this scanner has another token matching the specified pattern **Throws:** [IllegalStateException](http://docs.google.com/java/lang/IllegalStateException.html) - if this scanner is closed

### next

public [String](http://docs.google.com/java/lang/String.html) **next**([Pattern](http://docs.google.com/java/util/regex/Pattern.html) pattern)

Returns the next token if it matches the specified pattern. This method may block while waiting for input to scan, even if a previous invocation of [hasNext(Pattern)](http://docs.google.com/java/util/Scanner.html#hasNext(java.util.regex.Pattern)) returned true. If the match is successful, the scanner advances past the input that matched the pattern.

**Parameters:**pattern - the pattern to scan for **Returns:**the next token **Throws:** [NoSuchElementException](http://docs.google.com/java/util/NoSuchElementException.html) - if no more tokens are available [IllegalStateException](http://docs.google.com/java/lang/IllegalStateException.html) - if this scanner is closed

### hasNextLine

public boolean **hasNextLine**()

Returns true if there is another line in the input of this scanner. This method may block while waiting for input. The scanner does not advance past any input.

**Returns:**true if and only if this scanner has another line of input **Throws:** [IllegalStateException](http://docs.google.com/java/lang/IllegalStateException.html) - if this scanner is closed

### nextLine

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

Advances this scanner past the current line and returns the input that was skipped. This method returns the rest of the current line, excluding any line separator at the end. The position is set to the beginning of the next line.

Since this method continues to search through the input looking for a line separator, it may buffer all of the input searching for the line to skip if no line separators are present.

**Returns:**the line that was skipped **Throws:** [NoSuchElementException](http://docs.google.com/java/util/NoSuchElementException.html) - if no line was found [IllegalStateException](http://docs.google.com/java/lang/IllegalStateException.html) - if this scanner is closed

### findInLine

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

Attempts to find the next occurrence of a pattern constructed from the specified string, ignoring delimiters.

An invocation of this method of the form findInLine(pattern) behaves in exactly the same way as the invocation findInLine(Pattern.compile(pattern)).

**Parameters:**pattern - a string specifying the pattern to search for **Returns:**the text that matched the specified pattern **Throws:** [IllegalStateException](http://docs.google.com/java/lang/IllegalStateException.html) - if this scanner is closed

### findInLine

public [String](http://docs.google.com/java/lang/String.html) **findInLine**([Pattern](http://docs.google.com/java/util/regex/Pattern.html) pattern)

Attempts to find the next occurrence of the specified pattern ignoring delimiters. If the pattern is found before the next line separator, the scanner advances past the input that matched and returns the string that matched the pattern. If no such pattern is detected in the input up to the next line separator, then null is returned and the scanner's position is unchanged. This method may block waiting for input that matches the pattern.

Since this method continues to search through the input looking for the specified pattern, it may buffer all of the input searching for the desired token if no line separators are present.

**Parameters:**pattern - the pattern to scan for **Returns:**the text that matched the specified pattern **Throws:** [IllegalStateException](http://docs.google.com/java/lang/IllegalStateException.html) - if this scanner is closed

### findWithinHorizon

public [String](http://docs.google.com/java/lang/String.html) **findWithinHorizon**([String](http://docs.google.com/java/lang/String.html) pattern,  
 int horizon)

Attempts to find the next occurrence of a pattern constructed from the specified string, ignoring delimiters.

An invocation of this method of the form findWithinHorizon(pattern) behaves in exactly the same way as the invocation findWithinHorizon(Pattern.compile(pattern, horizon)).

**Parameters:**pattern - a string specifying the pattern to search for **Returns:**the text that matched the specified pattern **Throws:** [IllegalStateException](http://docs.google.com/java/lang/IllegalStateException.html) - if this scanner is closed [IllegalArgumentException](http://docs.google.com/java/lang/IllegalArgumentException.html) - if horizon is negative

### findWithinHorizon

public [String](http://docs.google.com/java/lang/String.html) **findWithinHorizon**([Pattern](http://docs.google.com/java/util/regex/Pattern.html) pattern,  
 int horizon)

Attempts to find the next occurrence of the specified pattern.

This method searches through the input up to the specified search horizon, ignoring delimiters. If the pattern is found the scanner advances past the input that matched and returns the string that matched the pattern. If no such pattern is detected then the null is returned and the scanner's position remains unchanged. This method may block waiting for input that matches the pattern.

A scanner will never search more than horizon code points beyond its current position. Note that a match may be clipped by the horizon; that is, an arbitrary match result may have been different if the horizon had been larger. The scanner treats the horizon as a transparent, non-anchoring bound (see [Matcher.useTransparentBounds(boolean)](http://docs.google.com/java/util/regex/Matcher.html#useTransparentBounds(boolean)) and [Matcher.useAnchoringBounds(boolean)](http://docs.google.com/java/util/regex/Matcher.html#useAnchoringBounds(boolean))).

If horizon is 0, then the horizon is ignored and this method continues to search through the input looking for the specified pattern without bound. In this case it may buffer all of the input searching for the pattern.

If horizon is negative, then an IllegalArgumentException is thrown.

**Parameters:**pattern - the pattern to scan for **Returns:**the text that matched the specified pattern **Throws:** [IllegalStateException](http://docs.google.com/java/lang/IllegalStateException.html) - if this scanner is closed [IllegalArgumentException](http://docs.google.com/java/lang/IllegalArgumentException.html) - if horizon is negative

### skip

public [Scanner](http://docs.google.com/java/util/Scanner.html) **skip**([Pattern](http://docs.google.com/java/util/regex/Pattern.html) pattern)

Skips input that matches the specified pattern, ignoring delimiters. This method will skip input if an anchored match of the specified pattern succeeds.

If a match to the specified pattern is not found at the current position, then no input is skipped and a NoSuchElementException is thrown.

Since this method seeks to match the specified pattern starting at the scanner's current position, patterns that can match a lot of input (".\*", for example) may cause the scanner to buffer a large amount of input.

Note that it is possible to skip something without risking a NoSuchElementException by using a pattern that can match nothing, e.g., sc.skip("[ \t]\*").

**Parameters:**pattern - a string specifying the pattern to skip over **Returns:**this scanner **Throws:** [NoSuchElementException](http://docs.google.com/java/util/NoSuchElementException.html) - if the specified pattern is not found [IllegalStateException](http://docs.google.com/java/lang/IllegalStateException.html) - if this scanner is closed

### skip

public [Scanner](http://docs.google.com/java/util/Scanner.html) **skip**([String](http://docs.google.com/java/lang/String.html) pattern)

Skips input that matches a pattern constructed from the specified string.

An invocation of this method of the form skip(pattern) behaves in exactly the same way as the invocation skip(Pattern.compile(pattern)).

**Parameters:**pattern - a string specifying the pattern to skip over **Returns:**this scanner **Throws:** [IllegalStateException](http://docs.google.com/java/lang/IllegalStateException.html) - if this scanner is closed

### hasNextBoolean

public boolean **hasNextBoolean**()

Returns true if the next token in this scanner's input can be interpreted as a boolean value using a case insensitive pattern created from the string "true|false". The scanner does not advance past the input that matched.

**Returns:**true if and only if this scanner's next token is a valid boolean value **Throws:** [IllegalStateException](http://docs.google.com/java/lang/IllegalStateException.html) - if this scanner is closed

### nextBoolean

public boolean **nextBoolean**()

Scans the next token of the input into a boolean value and returns that value. This method will throw InputMismatchException if the next token cannot be translated into a valid boolean value. If the match is successful, the scanner advances past the input that matched.

**Returns:**the boolean scanned from the input **Throws:** [InputMismatchException](http://docs.google.com/java/util/InputMismatchException.html) - if the next token is not a valid boolean [NoSuchElementException](http://docs.google.com/java/util/NoSuchElementException.html) - if input is exhausted [IllegalStateException](http://docs.google.com/java/lang/IllegalStateException.html) - if this scanner is closed

### hasNextByte

public boolean **hasNextByte**()

Returns true if the next token in this scanner's input can be interpreted as a byte value in the default radix using the [nextByte()](http://docs.google.com/java/util/Scanner.html#nextByte()) method. The scanner does not advance past any input.

**Returns:**true if and only if this scanner's next token is a valid byte value **Throws:** [IllegalStateException](http://docs.google.com/java/lang/IllegalStateException.html) - if this scanner is closed

### hasNextByte

public boolean **hasNextByte**(int radix)

Returns true if the next token in this scanner's input can be interpreted as a byte value in the specified radix using the [nextByte()](http://docs.google.com/java/util/Scanner.html#nextByte()) method. The scanner does not advance past any input.

**Parameters:**radix - the radix used to interpret the token as a byte value **Returns:**true if and only if this scanner's next token is a valid byte value **Throws:** [IllegalStateException](http://docs.google.com/java/lang/IllegalStateException.html) - if this scanner is closed

### nextByte

public byte **nextByte**()

Scans the next token of the input as a byte.

An invocation of this method of the form nextByte() behaves in exactly the same way as the invocation nextByte(radix), where radix is the default radix of this scanner.

**Returns:**the byte scanned from the input **Throws:** [InputMismatchException](http://docs.google.com/java/util/InputMismatchException.html) - if the next token does not match the *Integer* regular expression, or is out of range [NoSuchElementException](http://docs.google.com/java/util/NoSuchElementException.html) - if input is exhausted [IllegalStateException](http://docs.google.com/java/lang/IllegalStateException.html) - if this scanner is closed

### nextByte

public byte **nextByte**(int radix)

Scans the next token of the input as a byte. This method will throw InputMismatchException if the next token cannot be translated into a valid byte value as described below. If the translation is successful, the scanner advances past the input that matched.

If the next token matches the [*Integer*](#1t3h5sf) regular expression defined above then the token is converted into a byte value as if by removing all locale specific prefixes, group separators, and locale specific suffixes, then mapping non-ASCII digits into ASCII digits via [Character.digit](http://docs.google.com/java/lang/Character.html#digit(char,%20int)), prepending a negative sign (-) if the locale specific negative prefixes and suffixes were present, and passing the resulting string to [Byte.parseByte](http://docs.google.com/java/lang/Byte.html#parseByte(java.lang.String,%20int)) with the specified radix.

**Parameters:**radix - the radix used to interpret the token as a byte value **Returns:**the byte scanned from the input **Throws:** [InputMismatchException](http://docs.google.com/java/util/InputMismatchException.html) - if the next token does not match the *Integer* regular expression, or is out of range [NoSuchElementException](http://docs.google.com/java/util/NoSuchElementException.html) - if input is exhausted [IllegalStateException](http://docs.google.com/java/lang/IllegalStateException.html) - if this scanner is closed

### hasNextShort

public boolean **hasNextShort**()

Returns true if the next token in this scanner's input can be interpreted as a short value in the default radix using the [nextShort()](http://docs.google.com/java/util/Scanner.html#nextShort()) method. The scanner does not advance past any input.

**Returns:**true if and only if this scanner's next token is a valid short value in the default radix **Throws:** [IllegalStateException](http://docs.google.com/java/lang/IllegalStateException.html) - if this scanner is closed

### hasNextShort

public boolean **hasNextShort**(int radix)

Returns true if the next token in this scanner's input can be interpreted as a short value in the specified radix using the [nextShort()](http://docs.google.com/java/util/Scanner.html#nextShort()) method. The scanner does not advance past any input.

**Parameters:**radix - the radix used to interpret the token as a short value **Returns:**true if and only if this scanner's next token is a valid short value in the specified radix **Throws:** [IllegalStateException](http://docs.google.com/java/lang/IllegalStateException.html) - if this scanner is closed

### nextShort

public short **nextShort**()

Scans the next token of the input as a short.

An invocation of this method of the form nextShort() behaves in exactly the same way as the invocation nextShort(radix), where radix is the default radix of this scanner.

**Returns:**the short scanned from the input **Throws:** [InputMismatchException](http://docs.google.com/java/util/InputMismatchException.html) - if the next token does not match the *Integer* regular expression, or is out of range [NoSuchElementException](http://docs.google.com/java/util/NoSuchElementException.html) - if input is exhausted [IllegalStateException](http://docs.google.com/java/lang/IllegalStateException.html) - if this scanner is closed

### nextShort

public short **nextShort**(int radix)

Scans the next token of the input as a short. This method will throw InputMismatchException if the next token cannot be translated into a valid short value as described below. If the translation is successful, the scanner advances past the input that matched.

If the next token matches the [*Integer*](#1t3h5sf) regular expression defined above then the token is converted into a short value as if by removing all locale specific prefixes, group separators, and locale specific suffixes, then mapping non-ASCII digits into ASCII digits via [Character.digit](http://docs.google.com/java/lang/Character.html#digit(char,%20int)), prepending a negative sign (-) if the locale specific negative prefixes and suffixes were present, and passing the resulting string to [Short.parseShort](http://docs.google.com/java/lang/Short.html#parseShort(java.lang.String,%20int)) with the specified radix.

**Parameters:**radix - the radix used to interpret the token as a short value **Returns:**the short scanned from the input **Throws:** [InputMismatchException](http://docs.google.com/java/util/InputMismatchException.html) - if the next token does not match the *Integer* regular expression, or is out of range [NoSuchElementException](http://docs.google.com/java/util/NoSuchElementException.html) - if input is exhausted [IllegalStateException](http://docs.google.com/java/lang/IllegalStateException.html) - if this scanner is closed

### hasNextInt

public boolean **hasNextInt**()

Returns true if the next token in this scanner's input can be interpreted as an int value in the default radix using the [nextInt()](http://docs.google.com/java/util/Scanner.html#nextInt()) method. The scanner does not advance past any input.

**Returns:**true if and only if this scanner's next token is a valid int value **Throws:** [IllegalStateException](http://docs.google.com/java/lang/IllegalStateException.html) - if this scanner is closed

### hasNextInt

public boolean **hasNextInt**(int radix)

Returns true if the next token in this scanner's input can be interpreted as an int value in the specified radix using the [nextInt()](http://docs.google.com/java/util/Scanner.html#nextInt()) method. The scanner does not advance past any input.

**Parameters:**radix - the radix used to interpret the token as an int value **Returns:**true if and only if this scanner's next token is a valid int value **Throws:** [IllegalStateException](http://docs.google.com/java/lang/IllegalStateException.html) - if this scanner is closed

### nextInt

public int **nextInt**()

Scans the next token of the input as an int.

An invocation of this method of the form nextInt() behaves in exactly the same way as the invocation nextInt(radix), where radix is the default radix of this scanner.

**Returns:**the int scanned from the input **Throws:** [InputMismatchException](http://docs.google.com/java/util/InputMismatchException.html) - if the next token does not match the *Integer* regular expression, or is out of range [NoSuchElementException](http://docs.google.com/java/util/NoSuchElementException.html) - if input is exhausted [IllegalStateException](http://docs.google.com/java/lang/IllegalStateException.html) - if this scanner is closed

### nextInt

public int **nextInt**(int radix)

Scans the next token of the input as an int. This method will throw InputMismatchException if the next token cannot be translated into a valid int value as described below. If the translation is successful, the scanner advances past the input that matched.

If the next token matches the [*Integer*](#1t3h5sf) regular expression defined above then the token is converted into an int value as if by removing all locale specific prefixes, group separators, and locale specific suffixes, then mapping non-ASCII digits into ASCII digits via [Character.digit](http://docs.google.com/java/lang/Character.html#digit(char,%20int)), prepending a negative sign (-) if the locale specific negative prefixes and suffixes were present, and passing the resulting string to [Integer.parseInt](http://docs.google.com/java/lang/Integer.html#parseInt(java.lang.String,%20int)) with the specified radix.

**Parameters:**radix - the radix used to interpret the token as an int value **Returns:**the int scanned from the input **Throws:** [InputMismatchException](http://docs.google.com/java/util/InputMismatchException.html) - if the next token does not match the *Integer* regular expression, or is out of range [NoSuchElementException](http://docs.google.com/java/util/NoSuchElementException.html) - if input is exhausted [IllegalStateException](http://docs.google.com/java/lang/IllegalStateException.html) - if this scanner is closed

### hasNextLong

public boolean **hasNextLong**()

Returns true if the next token in this scanner's input can be interpreted as a long value in the default radix using the [nextLong()](http://docs.google.com/java/util/Scanner.html#nextLong()) method. The scanner does not advance past any input.

**Returns:**true if and only if this scanner's next token is a valid long value **Throws:** [IllegalStateException](http://docs.google.com/java/lang/IllegalStateException.html) - if this scanner is closed

### hasNextLong

public boolean **hasNextLong**(int radix)

Returns true if the next token in this scanner's input can be interpreted as a long value in the specified radix using the [nextLong()](http://docs.google.com/java/util/Scanner.html#nextLong()) method. The scanner does not advance past any input.

**Parameters:**radix - the radix used to interpret the token as a long value **Returns:**true if and only if this scanner's next token is a valid long value **Throws:** [IllegalStateException](http://docs.google.com/java/lang/IllegalStateException.html) - if this scanner is closed

### nextLong

public long **nextLong**()

Scans the next token of the input as a long.

An invocation of this method of the form nextLong() behaves in exactly the same way as the invocation nextLong(radix), where radix is the default radix of this scanner.

**Returns:**the long scanned from the input **Throws:** [InputMismatchException](http://docs.google.com/java/util/InputMismatchException.html) - if the next token does not match the *Integer* regular expression, or is out of range [NoSuchElementException](http://docs.google.com/java/util/NoSuchElementException.html) - if input is exhausted [IllegalStateException](http://docs.google.com/java/lang/IllegalStateException.html) - if this scanner is closed

### nextLong

public long **nextLong**(int radix)

Scans the next token of the input as a long. This method will throw InputMismatchException if the next token cannot be translated into a valid long value as described below. If the translation is successful, the scanner advances past the input that matched.

If the next token matches the [*Integer*](#1t3h5sf) regular expression defined above then the token is converted into a long value as if by removing all locale specific prefixes, group separators, and locale specific suffixes, then mapping non-ASCII digits into ASCII digits via [Character.digit](http://docs.google.com/java/lang/Character.html#digit(char,%20int)), prepending a negative sign (-) if the locale specific negative prefixes and suffixes were present, and passing the resulting string to [Long.parseLong](http://docs.google.com/java/lang/Long.html#parseLong(java.lang.String,%20int)) with the specified radix.

**Parameters:**radix - the radix used to interpret the token as an int value **Returns:**the long scanned from the input **Throws:** [InputMismatchException](http://docs.google.com/java/util/InputMismatchException.html) - if the next token does not match the *Integer* regular expression, or is out of range [NoSuchElementException](http://docs.google.com/java/util/NoSuchElementException.html) - if input is exhausted [IllegalStateException](http://docs.google.com/java/lang/IllegalStateException.html) - if this scanner is closed

### hasNextFloat

public boolean **hasNextFloat**()

Returns true if the next token in this scanner's input can be interpreted as a float value using the [nextFloat()](http://docs.google.com/java/util/Scanner.html#nextFloat()) method. The scanner does not advance past any input.

**Returns:**true if and only if this scanner's next token is a valid float value **Throws:** [IllegalStateException](http://docs.google.com/java/lang/IllegalStateException.html) - if this scanner is closed

### nextFloat

public float **nextFloat**()

Scans the next token of the input as a float. This method will throw InputMismatchException if the next token cannot be translated into a valid float value as described below. If the translation is successful, the scanner advances past the input that matched.

If the next token matches the [*Float*](#2s8eyo1) regular expression defined above then the token is converted into a float value as if by removing all locale specific prefixes, group separators, and locale specific suffixes, then mapping non-ASCII digits into ASCII digits via [Character.digit](http://docs.google.com/java/lang/Character.html#digit(char,%20int)), prepending a negative sign (-) if the locale specific negative prefixes and suffixes were present, and passing the resulting string to [Float.parseFloat](http://docs.google.com/java/lang/Float.html#parseFloat(java.lang.String)). If the token matches the localized NaN or infinity strings, then either "Nan" or "Infinity" is passed to [Float.parseFloat](http://docs.google.com/java/lang/Float.html#parseFloat(java.lang.String)) as appropriate.

**Returns:**the float scanned from the input **Throws:** [InputMismatchException](http://docs.google.com/java/util/InputMismatchException.html) - if the next token does not match the *Float* regular expression, or is out of range [NoSuchElementException](http://docs.google.com/java/util/NoSuchElementException.html) - if input is exhausted [IllegalStateException](http://docs.google.com/java/lang/IllegalStateException.html) - if this scanner is closed

### hasNextDouble

public boolean **hasNextDouble**()

Returns true if the next token in this scanner's input can be interpreted as a double value using the [nextDouble()](http://docs.google.com/java/util/Scanner.html#nextDouble()) method. The scanner does not advance past any input.

**Returns:**true if and only if this scanner's next token is a valid double value **Throws:** [IllegalStateException](http://docs.google.com/java/lang/IllegalStateException.html) - if this scanner is closed

### nextDouble

public double **nextDouble**()

Scans the next token of the input as a double. This method will throw InputMismatchException if the next token cannot be translated into a valid double value. If the translation is successful, the scanner advances past the input that matched.

If the next token matches the [*Float*](#2s8eyo1) regular expression defined above then the token is converted into a double value as if by removing all locale specific prefixes, group separators, and locale specific suffixes, then mapping non-ASCII digits into ASCII digits via [Character.digit](http://docs.google.com/java/lang/Character.html#digit(char,%20int)), prepending a negative sign (-) if the locale specific negative prefixes and suffixes were present, and passing the resulting string to [Double.parseDouble](http://docs.google.com/java/lang/Double.html#parseDouble(java.lang.String)). If the token matches the localized NaN or infinity strings, then either "Nan" or "Infinity" is passed to [Double.parseDouble](http://docs.google.com/java/lang/Double.html#parseDouble(java.lang.String)) as appropriate.

**Returns:**the double scanned from the input **Throws:** [InputMismatchException](http://docs.google.com/java/util/InputMismatchException.html) - if the next token does not match the *Float* regular expression, or is out of range [NoSuchElementException](http://docs.google.com/java/util/NoSuchElementException.html) - if the input is exhausted [IllegalStateException](http://docs.google.com/java/lang/IllegalStateException.html) - if this scanner is closed

### hasNextBigInteger

public boolean **hasNextBigInteger**()

Returns true if the next token in this scanner's input can be interpreted as a BigInteger in the default radix using the [nextBigInteger()](http://docs.google.com/java/util/Scanner.html#nextBigInteger()) method. The scanner does not advance past any input.

**Returns:**true if and only if this scanner's next token is a valid BigInteger **Throws:** [IllegalStateException](http://docs.google.com/java/lang/IllegalStateException.html) - if this scanner is closed

### hasNextBigInteger

public boolean **hasNextBigInteger**(int radix)

Returns true if the next token in this scanner's input can be interpreted as a BigInteger in the specified radix using the [nextBigInteger()](http://docs.google.com/java/util/Scanner.html#nextBigInteger()) method. The scanner does not advance past any input.

**Parameters:**radix - the radix used to interpret the token as an integer **Returns:**true if and only if this scanner's next token is a valid BigInteger **Throws:** [IllegalStateException](http://docs.google.com/java/lang/IllegalStateException.html) - if this scanner is closed

### nextBigInteger

public [BigInteger](http://docs.google.com/java/math/BigInteger.html) **nextBigInteger**()

Scans the next token of the input as a [BigInteger](http://docs.google.com/java/math/BigInteger.html).

An invocation of this method of the form nextBigInteger() behaves in exactly the same way as the invocation nextBigInteger(radix), where radix is the default radix of this scanner.

**Returns:**the BigInteger scanned from the input **Throws:** [InputMismatchException](http://docs.google.com/java/util/InputMismatchException.html) - if the next token does not match the *Integer* regular expression, or is out of range [NoSuchElementException](http://docs.google.com/java/util/NoSuchElementException.html) - if the input is exhausted [IllegalStateException](http://docs.google.com/java/lang/IllegalStateException.html) - if this scanner is closed

### nextBigInteger

public [BigInteger](http://docs.google.com/java/math/BigInteger.html) **nextBigInteger**(int radix)

Scans the next token of the input as a [BigInteger](http://docs.google.com/java/math/BigInteger.html).

If the next token matches the [*Integer*](#1t3h5sf) regular expression defined above then the token is converted into a BigInteger value as if by removing all group separators, mapping non-ASCII digits into ASCII digits via the [Character.digit](http://docs.google.com/java/lang/Character.html#digit(char,%20int)), and passing the resulting string to the [BigInteger(String, int)](http://docs.google.com/java/math/BigInteger.html#BigInteger(java.lang.String)) constructor with the specified radix.

**Parameters:**radix - the radix used to interpret the token **Returns:**the BigInteger scanned from the input **Throws:** [InputMismatchException](http://docs.google.com/java/util/InputMismatchException.html) - if the next token does not match the *Integer* regular expression, or is out of range [NoSuchElementException](http://docs.google.com/java/util/NoSuchElementException.html) - if the input is exhausted [IllegalStateException](http://docs.google.com/java/lang/IllegalStateException.html) - if this scanner is closed

### hasNextBigDecimal

public boolean **hasNextBigDecimal**()

Returns true if the next token in this scanner's input can be interpreted as a BigDecimal using the [nextBigDecimal()](http://docs.google.com/java/util/Scanner.html#nextBigDecimal()) method. The scanner does not advance past any input.

**Returns:**true if and only if this scanner's next token is a valid BigDecimal **Throws:** [IllegalStateException](http://docs.google.com/java/lang/IllegalStateException.html) - if this scanner is closed

### nextBigDecimal

public [BigDecimal](http://docs.google.com/java/math/BigDecimal.html) **nextBigDecimal**()

Scans the next token of the input as a [BigDecimal](http://docs.google.com/java/math/BigDecimal.html).

If the next token matches the [*Decimal*](#4d34og8) regular expression defined above then the token is converted into a BigDecimal value as if by removing all group separators, mapping non-ASCII digits into ASCII digits via the [Character.digit](http://docs.google.com/java/lang/Character.html#digit(char,%20int)), and passing the resulting string to the [BigDecimal(String)](http://docs.google.com/java/math/BigDecimal.html#BigDecimal(java.lang.String)) constructor.

**Returns:**the BigDecimal scanned from the input **Throws:** [InputMismatchException](http://docs.google.com/java/util/InputMismatchException.html) - if the next token does not match the *Decimal* regular expression, or is out of range [NoSuchElementException](http://docs.google.com/java/util/NoSuchElementException.html) - if the input is exhausted [IllegalStateException](http://docs.google.com/java/lang/IllegalStateException.html) - if this scanner is closed

### reset

public [Scanner](http://docs.google.com/java/util/Scanner.html) **reset**()

Resets this scanner.

Resetting a scanner discards all of its explicit state information which may have been changed by invocations of [useDelimiter(java.util.regex.Pattern)](http://docs.google.com/java/util/Scanner.html#useDelimiter(java.util.regex.Pattern)), [useLocale(java.util.Locale)](http://docs.google.com/java/util/Scanner.html#useLocale(java.util.Locale)), or [useRadix(int)](http://docs.google.com/java/util/Scanner.html#useRadix(int)).

An invocation of this method of the form scanner.reset() behaves in exactly the same way as the invocation

scanner.useDelimiter("\\p{javaWhitespace}+")  
 .useLocale(Locale.getDefault())  
 .useRadix(10);

**Returns:**this scanner**Since:** 1.6

| | [**Overview**](http://docs.google.com/overview-summary.html) | [**Package**](http://docs.google.com/package-summary.html) | **Class** | [**Use**](http://docs.google.com/class-use/Scanner.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/util/ResourceBundle.Control.html)   [**NEXT CLASS**](http://docs.google.com/java/util/ServiceConfigurationError.html) | [**FRAMES**](http://docs.google.com/index.html?java/util/Scanner.html)    [**NO FRAMES**](http://docs.google.com/Scanner.html)     [**All Classes**](http://docs.google.com/allclasses-noframe.html) |
| SUMMARY: NESTED | FIELD | [CONSTR](#17dp8vu) | [METHOD](#3rdcrjn) | DETAIL: FIELD | [CONSTR](#lnxbz9) | [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).