Scanner class in java example pdf

The Java Scanner class is used to read user input. The scanner is built into the java.util package, so external libraries are not needed to use it. While the scanner is the most common class used to generate input, you can also use BufferedReader, InputReader, DataInputStream, and Console classes. Understanding how to get custom input into Java is one of the most important skills. For example, if you're building an application with a form of input, you'll need to process custom input; If you build a to-do list app, you will need to receive and process custom input. In Java, you can use the Scanner class to get custom input that you can process in the program. This tutorial will discuss using a few examples of how to use the Java Scanner class to get custom input. Java Scanner ClassThe Java Scanner class is used to collect user input. The scanner is part of the java.util package, so it can be imported without downloading external libraries. To work with the Scanner class, you must first import it into the code. There are two ways to do this: if you only need to work with the java.util.Scanner class, you can import the Scanner class directly. If you work with other modules in the java.util library, you can import a full library. Below is the code for each of the above methods: import java.util.Scanner; This will allow you to import the Scanner class. This will allow you to import a full java.util library - every class in java.util.It it's worth noting that there are other ways to get custom input into Java. You can use BufferedReader, InputStreamReader, DataInputStream and Console classes. However, the Scanner class is the most popular method of collecting user input in Java, so we'll focus on this class in this article. Now that we know how to import the Scanner class, we can start working with custom input in our code. MORE: How to throw an exception in JavaJava InputPost after importing the Java Scanner class, you can start using it to collect user input. Here's the syntax for the Java Scanner class: Scanner input is a new scanner (System.in); int number - input.nextInt(); In this example, we created a variable called input that collects the following value that the user enters into the console. We then created a variable number that collects the value that the user sends to the console. For example, let's say we build an application for a local computer store that tracks their inventory. The manager asked us to create a simple program that she could use to add items to the inventory list The manager wants to be able to enter two values: the name of the item and its number. So in order to build a program, we have to accept two inputs from the user: the item name and the number of items. The first entry is the name of the item. It will be a string because the element is an element based on text and use different symbols. In the code below, we define this line with code: String product_name. The next input is the number of items. It's going to be a number. In the code below, we define this number with the code: int the number where Int means integer. Here's the code we'd like to use to create this program: import java.util.Scanner; ComputerInventory class - public static emptiness core (String) - Scanner input - new scanner (System.in); System.out.print Line product_name and input.next() System.out.product_name print System.out.print (Number:); int quantity - input.nextInt(); System.out.print When we run our code and insert a few example values, the program returns the following answer: Product Title: 15-inch MacBook Pro 2019Value entered: 15-inch MacBook Pro 2019'u'utity: 7Value entered: 7 As you can see, our program has collected user input. He then returned to the console the value that the user had entered. This allows us to make sure our program works. MORE: Java Compilers: Step by step GuideLet's break down our code step by step. We import the scanner library into our code so we can get custom input. We're announcing a class called ComputerInventory that stores the code for our program. We're initializing the scanner class with an input scanner , a new scanner (System.in); We print the product name on the console and ask the user to submit the product name using input.next(); We will print the product name on the console, which is provided by the user. We will print the number: on the console and encourage the user to submit the amount of product in stock using input.nextInt (); We'll print out the variable number of consoles. Note that we used a different code to collect numbers and strings. When we put together the product name, we used input.next;; and when we collected the amount of product, we used input.nextInt(;; In the next section, we'll discuss why this is the case, and we'll look at the different types of input that the Scanner class accepts. Java Input Types In our example above, we've collected two types of data from the user: the line and the integer. As mentioned, we had to use a different code to collect these types of data. When you're working with the Scanner class, the way you collect user inputs varies depending on the type of data you want to collect. So to assemble the boolean, you'll use a different code than you would assemble the float. Here's a table showing all the methods used to collect user input in Java with the scanner class: If you insert the wrong type of input, your program will raise InputMismatchException. For example, if you try double in the field, which collects booleans, your program will raise the exception. MORE: Java To Initiate Array: Step by Step GuideLet's Go Go to the computer store. Suppose we wanted to update our first program and allow the manager of our computer store to enter whether the product is on display or held in stock. To do this, we want to collect a new value called on_display which will store input as a boolean, because it can only have two meanings: true or false. Here's a code we could use to collect this data: import java.util.Scanner; ComputerInventory class - public static emptiness core (String) - Scanner input - new scanner (System.in); System.out.print Line product_name and input.next() System.out.product_name print System.out.print (Number:); int quantity - input.nextInt(); System.out.print System.out.print (On display:); boolean on_display and input.nextBoolean System.out.on_display print When we run our code and insert a few example values, the program returns the following answer: Product Title: 15-inch MacBook Pro 2019Value entered: 15-inch MacBook Pro 2019'uourity: 7Value entered: trueValue entered: trueValue entered: trueOur The program works just like our example above. However, this time we collect additional value from the user: whether the product they inserted into the program is on display. We use the following Boolean method to collect this value from the user. We then print this value on the console. ConclusionYou can use the Java scanner class to collect input from the user. The Scanner class is capable of collecting different types of data from users, including short values, strings, booleans, and others. In this tutorial, using a few examples, we learned how to use the Java Scanner class to collect user input. In addition, we discussed the different types of data offered by the Scanner class that we can use to collect user input. Now you have the knowledge you need to collect custom input using the Scanner class like a professional! Want to explore a technology career? java.lang.Object All implemented interfaces: Closeable, AutoCloseable, Iterator'lt;String'gt; Public End Class Scanner Expands Object Implements Iterator, a Closeable Simple Text Scanner that can analyze primitive types and strings using regular expressions. The scanner breaks input into tokens using a delimitator template that coincides with the white space by default. The resulting tokens can be converted into different types of values using different methods. For example, this code allows the user to read a number from the System.in: Scanner sc - a new scanner (System.in); int i s.nextInt (); As another example, this code allows Types to be assigned from myNumbers: Scanner sc and new scanner (new file (myNumbers)) while (sc.hasNextLong()the scanner can also use delimitras other than white space. In this example, you read a few elements from the line: String Writing - 1 fish 2/lt; red fish blue fish; Scanner s - new scanner (entry).useDelimiter (sesfish); 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

Same output can be created using this code, which uses a regular expression to disassemble all four tokens at once: Entry line - 1 fish 2 fish red fish blue fish; Scanner s - new scanner (entry); s.findInLine (d) fish (s.D.) fish (SOUND)); Match resultResult s.match (); For (int i'1; i By default, the whitespace delimitter used by the scanner is recognized as Character.isWhitespace. and then try to return the next marker. Both have Next and the following methods can block the expectation of further input. Whether or not the HasNext block has anything to do with the following method, it will not be blocked. FindInLine (java.lang.String), findWithinHorizon (java.lang.String, int) and skip (java.util.regex.Pattern) methods work independently of the delimiter template. These methods will attempt to comply with this pattern without taking into account the de-liteers at the entrance and thus can be used in special circumstances where the de-izers are not relevant. These methods can block the expectation of more input. When the scanner throws InputMismatchException, the scanner will not transmit the marker that caused the exception, so that it can be retrieved or omitted using any other method. Depending on the type of demarcation pattern, empty tokens can be returned. For example, the S pattern does not return empty tokens because it corresponds to several copies of the delimitator. The delineation pattern can return empty tokens, as it only passes one space at a time. The scanner can read text from any object that implements the readable interface. If IOException challenges Readable's Readable.read (java.nio.CharBuffer) method, the scanner assumes that the end of the input has been reached. The last IOException, abandoned by the underlying reader, can be obtained using the ioException method. When the scanner is closed, it will close its input if the source implements a close interface. The scanner is not safe for multi-read use without external synchronization. If otherwise not mentioned, passing a zero option in any scanner method will result in NullPointerException being thrown. The scanner will be the default numbers as decimal if another radix was installed using the help of use the Radix Method (int). The reset method will drop the scanner's radix value to 10, regardless of whether it has been changed previously. A copy of this class is able to scan numbers in both standard formats and scanner localization formats. The initial location of the scanner is a value returned by Locale.getDefault. it can be modified using useLocale (java.util.Locale). The reset method will reset the location of the scanner to the initial locale, regardless of whether it has been altered previously. Localized formats are defined in terms of the following parameters, which for a particular locale are taken from this local object DecimalFormat, df, and his and the DecimalFormatSymbols, dfs. LocalGroupSeparator Character is used for dispersing thousands of groups, i.e. dfs.getGroupingSeparator () LocalDecimalSeparator Character used for decimal point, i.e., dfs.getDeciMalSeparator () LocalPositivePrefix Line, which appears before a positive number (may be empty), i.e. df.getPositivePrefix () LocalPositiveSuffix Line, which appears after a positive number (may be empty), i.e. df.getPositiveSuffix () LocalNegativePrefix Line, which appears before the negative number , df.getNegativePrefix () LocalNegativeSuffix Line, which appears after a negative number (may be empty), i.e. df.getNegativeSuffix () LocalNaN Row that represents a non-number for floating point values, i.e. dfs.getNaN () LocalInfinity Row, which represents infinity for floating point values, i.e. dfs.getInfinity () Lines that can be disassembled as numbers by a copy of this class are indicated in terms of the following regular expression grammar, where Rmax is the highest digit in the radix used (e.g., Rmax 9 in base 10). NonASCIIDigit :: - non-ASCII c symbol, for which Character.isDigit (c) returns true Non0Digit :: NonASCIIDigit Digit :: NonASCIIDigit GroupedNumeral :: No (Non0Dig Digitit? (LocalGroupSeparator Digit Digit Digit) Number :: GroupedManal) Integer :: (Number) LocalPositivePrefix LocalPositiveSuffix (en) LocalNegativePrefix Numeral LocalNegativeSuffix DecumalUmimanal :: Figure LocalDecimalSeparator Digit LocalDecimalSeparator' Figure Exhibitor :: Digit : Decimal :: Decimal?) | LocalPositivePrefix Ten-dimensional LocalPoitiveSuffix Exhibitor? LocalNegativePrefix DecimalNumeral LocalNegativeSuffix Exponent?   HexFloat :: 0'xX-0-9a-fA-Fz. «0-9a-fA-F» («PP»-?» 0-9)?   Nonnumber :: LocalNan (en) Infinity LocalInfinity Signed ByNonNumber :: NonNumber) Local positivePreprifix Nonnnumber | LocalNegativePrefix NonNumber LocalNegativeSuffix Float :: HexFloate SignatureNonNumber Whitespace is not significant in the above regular expressions. Because: 1.5 public scanner (Readable source) creates a new scanner that produces values scanned from the specified source. Options:Source - The source of the symbols that implements the readable interface of the public scanner (source inputStream) creates a new scanner that produces values scanned from the specified input stream. Bytes from the thread are converted into symbols that use the charset of the default base platform. Options: Source - inputStream source (String charsetName) creates a new scanner that produces values scanned from the specified input stream. Bytes from the stream are converted into symbols that use this set of symbols. Options: Source - Input stream for scancharsetName - The type of coding used to convert bytes from stream to symbols to scan Broschi: IllegalArgumentException - if the specified set of characters does not exist public scanner (file source) throws FileNotFoundException creates a new scanner that produces values scanned from the specified file. Bytes from the file are converted into symbols that use the charset of the default base platform. Options: Source - File for Flash Scanning: FileNotFoundException - If the source is not found by the public Scanner (File Source, String charsetName) throws FileNotFoundException creates a new scanner that produces values scanned from the specified file. Bytes from the file are converted into symbols with this set of symbols. Options: Source - Scan filecharsetName - Type of coding used to convert bytes from file to symbols to scan Throws: FileNotFoundException - if the source is not found IllegalArgumentException - if the specified coding is not found by a public scanner (The path source) casts IOException creates a new scanner that produces values scanned from the specified file. Bytes from the file are converted into symbols that use the charset of the default base platform. Options: Source - The Way to File to Scan Throws: IOException - If an I.O.C. error occurs sourceSince: 1.7 public scanner (Route Source, String charsetName) throws IOException creates a new scanner that produces values scanned from the specified file. Bytes from the file are converted into symbols with this set of symbols. Options:source - the way to the file to scancharsetName - The type of coding used to convert bytes from file to symbols to scan Throws: IOException - if the I/O error occurs the discovery of the source of IllegalArgumentException - if specified Not foundC: 1.7 public scanner (Line Source) Creates a new scanner that produces values scanned from out String. Options: Source - Line to scan a public scanner (source ReadableByteChannel) creates a new scanner that produces values scanned from the specified channel. Bytes from the source are converted into symbols that use the charset of the default base platform. Options: Source - ReadableByteChannel source (String charsetName) creates a new scanner that produces values scanned from the specified channel. Bytes from the source are converted into symbols that use this set of symbols. Options:Source - channel for scancharsetName - Type of coding used to convert bytes from channel to symbols to scan Bros: IllegalArgumentException - if the specified set of characters does not exist public void close () closes this scanner. If this scanner has not yet been closed, if its basic readable also implements a close interface, the method of closing the readable will be called. If this scanner is already closed, the call of this method will have no effect. Attempting to perform search operations after the scanner is closed will result in illegal government perception. Indicated: close in the Closeable Specified interface: close in the Interface AutoCloseable Public IOException ioException () Returns IOException the last abandoned by this scanner is the main Readable. This method returns to zero if there is no such exception. Returns: The last exception thrown by the publicly readed delimiter of this scanner's template () returns the pattern that this scanner currently uses to match the deimiters. Returns: The delimitation pattern of this scanner. public use of theDelimiter scanner (pattern pattern) establishes the delimitation pattern of that scanner to the specified pattern. Settings: Pattern - Returns Pattern Delimitation: This useDelimiter (String pattern) is setting the scanner's delimitation pattern on a pattern built from a specified line. Calling this method of useDelimiter (pattern) is just like calling useDelimiter (Pattern compile.). Referring to the reset method, the scanner's demither will be installed by default. Settings: Pattern - a line indicating the delimitation of the Returns:this scanner public locale locale () Returns the locale of this scanner. The scanner's lock affects many elements of its primitive default expressions that correspond to conventional expressions; see the localized figures above. Returns: This scanner in the location of the public scanner useLocale (Locale locale) installs the locale of this scanner in the specified locale. The scanner's lock affects many elements of its primitive default expressions that correspond to conventional expressions; see the localized figures above. Referring to the reset method, the scanner's locale will be installed on the initial locale. - line indicating local returns:this scanner public int radix () Returns the radix of this default scanner. Default. The scanner's radix affects the elements of its default number that correspond to the usual expressions; see the localized figures above. Returns: The default part of this useRadix public scanner (int radix) sets the radix of this scanner by default to the specified radix. The scanner's radiox affects the elements of its default number that correspond to the usual expressions; see the localized figures above. If the radix is smaller, Character.MIN_RADIX or more, Character.MAX_RADIX, then the illegalArgumentExper is abandoned. Referring to the reset method, the scanner will have a radius of up to 10. Options:radix - radix for use when scanning returns numbers: this scanner throws: IllegalArgumentException - if the radix is out of range of the public match MatchResult () Returns the result of the match of the last scan operation performed by this scanner. This method throws Out IllegalStateException if the match has not been executed, or if the last match was not successful. Various nextmethods scanner make match results available if they are completed without any exceptions. For example, after calling the nextInt () method that has returned int, this method returns MatchResult to search for the usual Integer expression, defined above. Similarly findInLine (java.lang.String), findWithinHorizon (java.lang.String, int) and skip (java.util.regex.Pattern) methods will make the match available if they succeed. Returns: Match result for the last match operation Throws: Illegal StateException - If there is no match result available the public toString line () Returns the presentation of the line of this scanner. The view of the scanner line contains information that may be useful for debugging. The exact format is not specified. Redefining: toString in the Object Returns class: The string view of this public boolean hasNext scanner returns correctly if this scanner has another marker in its input. This method can be blocked while waiting for the scan input. The scanner does not advance past any input. Indicated: has Next in the Iterator Returns interface:true if and only if this scanner has another marker Throws: IllegalStateException - if this scanner is closedSee Also:Iterator public String next () Finds and returns the next full marker from this scanner. The full marker is preceded and accompanied by an input that matches the delimiter pattern. This method can be blocked while waiting for the scan input, even if the previous hasNext call came back true. Indicated: next in the Interface Iterator'lt;Returns:Next token throws: NoSuchElementException - if no more tokens are available IllegalStateException - if this scanner is closedSee Also:Iterator public boolean hasNext (String pattern) Returns correctly if the next marker corresponds to the pattern built from the specified line. don't advance past any input. Calling this form method has a string)/string'gt; just like the challenge has Next (Pattern.compile)). Options: Pattern - a line, specifying a template for scanning Returns:true, if and only if the scanner has another marker consistent with the indicated Throws: IllegalStateException pattern - if this scanner is closed by a public string side by side (String pattern) returns the next marker if it corresponds to a pattern built from the specified line. If the match is successful, the scanner moves past inputs that match the pattern. Calling this form method next (pattern) behaves exactly like the next call (Pattern.compile) (pattern).. Options: Pattern - pattern with template for Scanning Returns:next token Throws: NoSuchElementException - if there are no such tokens available IllegalStateException - if this scanner is closed publicly boolean hasNext (pattern pattern) Returns true if the next full marker corresponds to the specified pattern. The full marker is inserted and post-fixed by the input, which corresponds to the delimiter pattern. This method can be blocked while waiting for input. The scanner does not advance past any input. Options: A template - a template for scanning returns:true if and only if this scanner has another marker that matches the indicated Throws: IllegalStateException pattern - if that scanner is closed by a public line side by side (pattern pattern) returns the next marker if it fits the specified pattern. This method can be blocked while waiting for the scan input, even if the previous hasNext call (Pattern) came back true. If the match is successful, the scanner moves past inputs that match the pattern. Options:pattern - pattern for scanning returns: the following token throws: NoSuchElementException - if no more tokens are available IllegalStateException - if this scanner is closed by the public boolean hasNextLine () Returns correctly if there is another line in the input of this scanner. This method can be blocked 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: IllegalStateException - if this scanner is closed by the public line nextLine () Achievements of this scanner past the current line and returns the entrance that was missed. This method returns the rest of the current line, eliminating any line separator at the end. Position is set at the beginning of the next line. As this method continues to search through input looking for a line separator, it can buffer all input searches for the line to skip if no separator lines are present. Returns: Line that was missed Throws: NoSuchElementException - if no line has been found IllegalStateException - if this scanner is closed by public String findInLine (String pattern) Attempts to find the following appearance of a pattern built from the specified line, delimiters. Calling this method the findInLine (pattern) form behaves in the same way as the findInLine (Pattern.compile) call. Settings:pattern - a line that indicates a pattern to search for Returns:text that corresponded to the specified Throws: IllegalStateException pattern - if this scanner is closed by a public String findInLine (pattern) Attempts to find the following appearance of the specified pattern by ignoring the deimiterers. If a pattern is found in front of the next line, the scanner moves past the input that matches and returns a line that fits the pattern. If this pattern is not detected when you enter to the next line separator, the scanner returns to invalid and the scanner position remains the same. This method can block the wait for input that matches the pattern. As this method continues to search through input looking for the specified pattern, it can buffer all input searches of the desired marker if no string of separators are present. Settings: A template for scanning returns: text that corresponded to the specified Throws: IllegalStateException pattern - if this scanner is closed by the public String findWithinHorizon (String pattern, int horizon) Attempts to find the following appearance of a pattern built from the specified line, ignoring the delimiter. Calling this form method findWithinHorizon (pattern) behaves in the same way as the findWithinHorizon (Pattern.compile) call. Settings: Pattern - a line with a template to search for returns:text that corresponded to the specified Throws template: IllegalStateException - if this scanner is closed By IllegalArgumentException - if the horizon is negative public Search For LinesInMorizone (pattern, int horizon) Attempts to find the following appearance of the specified pattern. This method searches through the entrance to the specified search horizon, ignoring the de-ismers. If the pattern is found, the scanner moves past the input that matches, and returns a line that fits the pattern. If this pattern is not detected, the null returns and the position of the scanner remains unchanged. This method can block the wait for input that matches the pattern. The scanner will never look more than a horizon code point beyond its current position. Note that the match can be clipped by the horizon; that is, the result of an arbitrary match could have been different if the horizon had been greater. The scanner views the horizon as transparent, non-anchor connected (see Matcher.useTransparentBounds (boolean) and Matcher.useAnchoringBounds (boolean)). If the horizon is 0, the horizon is ignored, and this method continues to search through the entrance looking for a specified pattern without boundaries. In this case, it can buffer all inputs from the pattern search. If the horizon is negative, Then IllegalArgumentException is thrown. Options:pattern - Returns scanning template: text that fits the specified Throws: IllegalStateException template - if this scanner is closed - if the horizon is a negative public scanner to skip (pattern pattern) Skips an input that corresponds to the specified pattern, ignoring the de-ization. This method skips the input if the anchor match of the specified pattern is successful. If you don't find a match with this pattern in the current position, no NoSuchElementException is not allowed to enter or be thrown away. Because this method tends to fit the pattern, starting with the current position of the scanner, patterns that can match a large number of inputs (. Note that you can skip something without risking NoSuchElementException using a template that can't match anything, such as sc.skip (t). Settings: pattern - line, pointing pattern to skip returns: this scanner throws: NoSuchElementException - if the specified pattern is not found that corresponds to a pattern built from the specified line. Calling this form skipping method (pattern) behaves just like a skip call (Pattern.compile) (pattern).) Options:pattern - line showing template to skip Returns: This scanner Throws: IllegalStateException - if this scanner is closed to the public boolean hasNextBoolean () Returns correctly if the next marker in this input scanner can be interpreted as a boolean value using a case of insensitive pattern created from the line truefalse. The scanner does not pass by the input that fits. Returns:True if and only if the next marker of this scanner is a valid boolean value Casts: IllegalStateException - if this scanner is closed by the public boolean nextBoolean () scans the next entry marker in the boolean value and returns that value. This method will throw InputMismatchException if the next marker cannot be translated into the actual boolean value. If the match is successful, the scanner moves past the input that matches. Returns:boolean scanned from input: InputMismatchException - if the next marker is not a valid boolean NoSuchElementException - if the input is exhausted ByErstateException - if this scanner is closed public boolean hasNextByte () Returns correctly if the next marker in the input of this scanner can be interpreted as byte value in the default radius by the next scanner. Returns:true if and only if the next token of this scanner is a valid value byte Throws: IllegalStateException - if this scanner is closed by public boolean hasNextByte (int radix) returns if the next token in the input of this scanner can be interpreted as te value in the specified radix with the help of nextByte () method. The scanner does not advance past any input. Options:radix - radix used to interpret the token as byte Returns:true if and only if the next token of this scanner is the valid value of byte Throws: IllegalStateException - if this scanner is closed byte nextByte () Scans the next token input as byte. Calling this nextByte form is just like calling nextByte (radix), where radix is the default radix of this scanner. Returns: Byte scanned from input: InputMismatchException - if the next marker does not correspond to the regular Integer expression, or is outside the NoSuchElementException range - if the input is exhausted - if this scanner is closed in public byte nextByte (int radix) scans the next entry marker as byte. This method will throw InputMismatchException if the next token cannot be translated into the actual te value as described below. If the translation is successful, the scanner moves past the input that matches. If the next token corresponds to the usual Integer expression, defined above, the token is converted into a byse value, as if removing all local specific prefixes, group separators, and localization specific suffixes, and then displaying non-ASCII figures in ASCII figures through Character.digit, pre-location negative mark (-) if local specific negative prefixes and suffixes were present, and transmissions in the result of the line by Options:radix - radix used to interpret the token as te Returns:the byte, Scanned from input: InputMismatchException - if the next token does not correspond to the regular Integer expression, or is out of the NoSuchElementException range - if the input is exhausted By IllegalStateException - if this scanner is closed publicly boolean hasNextShort () Returns correctly if the next marker in the input of this scanner can be interpreted as a short value in the default method. The scanner does not advance past any input. Returns:True, if and only if the next marker of this scanner is a valid short value in the default throws radix: IllegalStateException - if this scanner is closed publicly boolean hasNextShort (int radix) Returns correctly if the next marker in the input of this scanner can be interpreted as a short value in the specified radix using nextShort.) Method. The scanner does not advance past any input. Options:radix - radix used to interpret the token as a short Value Returns:true if and only if the next token of this scanner is a valid short value in the specified radix Throws: IllegalStateException - if this scanner is closed by public short nextShort () Scans the following entering as short. Calling this nextShort form () behaves in the same way as the nextShort (radix) call, where radix radix radius of this scanner. Returns: Short scanned from input: InputMismatchException - if the next marker does not correspond to the regular Integer expression, or is outside the NoSuchElementException range - if the input is exhausted by IllegalStateException - if this scanner is closed by the public short nextShort (int radix) scans the following entry marker as short. This method will throw InputMismatchException if the next token cannot be translated into the actual short value described below. If the translation is successful, the scanner moves past the input that matches. If the next token corresponds to the usual Integer expression, defined above, the token is converted into a short value, as if removing all local specific prefixes, group separators, and localization-specific suffixes, and then displaying non-ASCII figures in ASCII figures through Character.digit, pre-expeding negative mark (-) if local specific negative prefixes and suffixes were present, and transmissions were present. Options:radix - radix used to interpret the token as a short Returns value: short scan from input: InputMismatchException - if the following token does not correspond to the regular Integer expression, or is out of the NoSuchElementException range - if the input is exhausted - if this scanner is closed by the public boolean hasNextInt () Returns correctly, if the next marker in the input of this scanner can be interpreted as an int value in the default radix using the nextInt () method. The scanner does not advance past any input. Returns:true if and only if the next token of this scanner is a valid int value Casts: IllegalStateException - if this scanner is closed to the public boolean hasNextInt (int radix) Returns correctly if the next marker in the input of this scanner can be interpreted as int value in the specified radix using the following () method. The scanner does not advance past any input. Options:radix - radix used to interpret the token as int value Returns:true, if and only if the next token of this scanner is a valid int value Of Throws: IllegalStateException - if this scanner is closed publicly int nextInt () Scans the next entry marker as int. Calling this nextInt form is just like calling nextInt (radix), where radix is the default radix of that scanner. Returns:int scanned from input: InputMismatchException - if the next token does not correspond to the regular Integer expression, or is outside the NoSuchElementException range - if the input is exhausted - if this scanner is closed in public int nextInt (int radix) scans the next entry marker as int. This method will throw if the next token cannot be translated into the actual int value as described below. If the translation is successful, the scanner moves past an entrance that fits. If the next token corresponds to the usual Integer expression, defined above, the token is converted into an int-value, as if removing all local prefixes, groups of separators, and location specific suffixes, and then displaying non-ASCII figures in ASCII figures through Character.digit, pre-expeding negative mark (-) if localized specific negative prefixes and suffixes were present, and transmissions in the result of the integer.par. Options:radix - radix used to interpret the token as int value Returns:the int, Scanned from input: InputMismatchException - if the next token does not correspond to the regular Integer expression, or is out of the NoSuchElementException range - if the input is exhausted By IllegalStateException - if this scanner is closed publicly boolean hasNextLong () Returns correctly if the next marker in the input of this scanner can be interpreted as a long value in the default radx. The scanner does not advance past any input. Returns:true if and only if the next token of this scanner is the actual long value of Throws: IllegalStateException - if this scanner is closed by the public boolean hasNextLong (int radix) returns correctly, if the next marker in the input of this scanner can be interpreted as a long value in the specified radix using the nextLong method. The scanner does not advance past any input. Options:radix is a radix used to interpret the token as a long Returns:true if and only if the next token of this scanner is a valid long value of Throws: IllegalStateException - if this scanner is closed by the public for a long time nextLong () Scans the next entry marker as long. Calling this nextLong form method behaves in the same way as the nextLong (radix) call, where radix is the default radix of this scanner. Returns: Long scanned from input Throws: InputMismatchException - if the next token does not match Integer's regular expression, or is out of range NoSuchElementException - if the input is exhausted IllegalStateException - if this scanner is closed to the public for long nextLong (int radix) scans the next input marker as long. This method will throw InputMismatchException if the next token cannot be translated into the actual long value described below. If the translation is successful, the scanner moves past the input that matches. If the following token corresponds to the usual Integer expression defined above, then the token is converted into a long value, as if removing all local specific prefixes, group separators, and localization of specific negative suffixes, and then displaying non-ASCII numbers in ASCII figures through Character.digit, pre-spending negative mark (-) if localized specific negative and suffixes were present, and transmissions resulted in the long.parseLong line with the specified radix. Options:radix - radix used to interpret the token as Returns value: Long scanned from input: InputMismatchException - if the next marker does not correspond to the regular Integer expression, or is outside the NoSuchElementException range - if the input is exhausted By IllegalStateException - if this scanner is closed by the public boo hasleanNextFloat () returns correctly if the next marker in the input of this scanner can be interpreted as float value with the following. The scanner does not advance past any input. Returns:true if and only if the next token of this scanner is the valid value of the Float Casts: IllegalStateException - if this scanner is closed by a public float nextFloat () scans the next entry marker as a float. This method will throw InputMismatchException if the next marker cannot be translated into the actual float value as described below. If the translation is successful, the scanner moves past the input that matches. If the following token corresponds to the regular Float expression defined above, the token is converted to a floating value, as if removing all local specific prefixes, group separators and local specific suffixes, then displaying non-ASCII figures into ASCII numbers via Character.digit, anticipating a negative sign (-if local specific negative prefixes and suffixes, and If the token corresponds to localized NaN or infinity lines, either Nan or Infinity are transferred to Float.parseFloat as needed. Returns: Float scanned from input: InputMismatchException - if the next token does not match the regular float expression, or is outside the NoSuchElementException range - if the entry is exhausted - if this scanner is closed by a public boolean hasNextDouble () Returns correctly if the next marker in the input of this scanner can be interpreted as a double value using the nextDouble method. The scanner does not advance past any input. Returns:True if and only if the next token of this scanner is a valid double Throws value: IllegalStateException - if this scanner is closed to the public double nextDouble () scans the next entry marker 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 moves past the input that matches. If the following token corresponds to the regular Float expression defined above, the token is converted into a double value, as if removing all local specific prefixes, group separators and local specific suffixes, then displaying non-ASCII figures into ASCII figures via Character.digit, prior to the negative mark (-) if local specific negative prefixes were present. the token corresponds to localized NaN or infinity lines, either Nan or Infinity are passed to Double.parseDouble as needed. Need. Double scanning from input: InputMismatchException - if the next marker does not correspond to the regular float expression, or is outside the NoSuchElementException range - if the input is exhausted by IllegalStateException - if this scanner is closed by the public boolean hasNextBigInteger () Returns correctly if the next marker in the input of this scanner can be interpreted as a BigInteger in advance. Returns:True, if and only if the next marker of this scanner is valid BigInteger Throws: IllegalStateException - if this scanner is closed to the public boolean hasNextBigInteger (int radix) returns correctly if the next marker in the input of this scanner can be interpreted as BigInteger in the specified radix using the nextBigInteger () method. The scanner does not advance past any input. Options:radix - radix used to interpret the token as Returns integrator:true if and only if the next token of this scanner is valid BigInteger Throws: IllegalStateException - if this scanner is closed public BigInteger nextBigInteger () scans the following token input as BigInteger. Calling this form nextBigInteger () behaves in the same way as calling nextBigInteger (radix), where radix is the default radix of this scanner. Returns:BigInteger is scanned from input: InputMismatchException - if the next marker does not match the regular Integer expression, or is outside the NoSuchElementException range - if the input is exhausted - if this scanner is closed to the public BigInteger NextBigInteger (int radix) scans the next entry token. If the next token corresponds to the usual Integer expression defined above, the token is converted to BigInteger, as if removing all group separators, displaying non-ASCII numbers into ASCII numbers via Character.digit and transferring the resulting line to BigInteger (String, int) with the specified radius. Options:radix - radix used to interpret Returns:the BigInteger token, Scanned from input: InputMismatchException - if the following token does not correspond to the regular Integer expression, or is out of the NoSuchElementException range - if the input is exhausted by IllegalStateException - if this scanner is closed publicly boolean hasNextBigDecimal () returns correctly if the next marker in the input of this scanner can be interpreted as BigDecimal using the following The scanner does not advance past any input. Returns:True, if and only if the next token of this scanner is valid BigDecimal Throws: IllegalStateException - if this scanner is closed publicly BigDecimal nextBigDecimal () scans the next entry marker as BigDecimal. If the next token corresponds to the decimal normal expression, defined above, then is transformed into The BigDecimal value, as by all group separators by displaying non-ASCII numbers into ASCII numbers through Character.digit and passing the resulting line to BigDecimal (String). Returns:BigDecimal is scanned from input: InputMismatchException - if the next marker does not correspond to the decimal regular expression, or is outside the NoSuchElementException range - if the input is exhausted - if this scanner is closed by a public reboot of the scanner () resets this scanner. Resetting the scanner discards all of its clear status information, which may have been altered by calls to useDelimiter (java.util.regex.Pattern), useLocale (java.util.Locale), or useRadix (int). Calling this form is just like a call scanner. Returns: this scanner. 1.6 Send an error or feature For further help with the API and developer documentation, see Java SE Documentation. This documentation contains more detailed developer-centered descriptions, with conceptual reviews, definitions of terms, workflows, and work code examples. The © 1993, 2020, Oracle and/or its affiliates. All rights are reserved. The use depends on the terms of the license. Also see the policy of redistribution of documentation. Policy. scanner class in java example program. scanner class in java example pdf. use of scanner class in java example. scanner class methods in java example. example program using scanner class in java

normal_5f8af059051ec.pdf
normal_5f872344c6eeb.pdf
normal_5f8a21fdaa4f1.pdf