Package edu.princeton.cs.algs4

Class StdIn

Object
    edu.princeton.cs.algs4.StdIn

```
public final class StdIn
extends Object
```

Overview. The `StdIn` class provides static methods for reading strings and numbers from standard input. These functions fall into one of four categories:

- those for reading individual tokens from standard input, one at a time, and converting each to a number, string, or boolean
- those for reading characters from standard input, one at a time
- those for reading lines from standard input, one at a time
- those for reading a sequence of values of the same type from standard input, and returning the values in an array

Generally, it is best not to mix functions from the different categories in the same program.

Getting started. To use this class, you must have `StdIn.class` in your Java classpath. If you used our autoinstaller, you should be all set. Otherwise, either download stdlib.jar and add to your Java classpath or download StdIn.java and put a copy in your working directory.

Reading tokens from standard input and converting to numbers and strings. You can use the following methods to read numbers, strings, and booleans from standard input one at a time:

- `isEmpty()`
- `readInt()`
- `readDouble()`
- `readString()`
- `readShort()`
- `readLong()`
- `readFloat()`
- `readByte()`
- `readBoolean()`

The first method returns true if standard input has no more tokens. Each other method skips over any input that is whitespace. Then, it reads the next token and attempts to convert it into a value of the specified type. If it succeeds, it returns that value; otherwise, it throws an `InputMismatchException`.

Whitespace includes spaces, tabs, and newlines; the full definition is inherited from `Character.isWhitespace(char)`. A *token* is a maximal sequence of non-whitespace characters. The precise rules for describing which tokens can be converted to integers and floating-point numbers are inherited from Scanner, using the locale `Locale.US`; the rules for floating-point numbers are slightly different from those in `Double.valueOf(String)`, but unlikely to be of concern to most programmers.

As an example, the following code fragment reads integers from standard input, one at a time, and prints them one per line.

```
while (!StdIn.isEmpty()) {
    double value = StdIn.readDouble();
    StdOut.println(value);
}
```

Reading characters from standard input. You can use the following two methods to read characters from standard input one at a time:

- `hasNextChar()`
- `readChar()`

The first method returns true if standard input has more input (including whitespace). The second method reads and returns the next character of input on standard input (possibly a whitespace character).

As an example, the following code fragment reads characters from standard input, one character at a time, and prints it to standard output.

```
while (StdIn.hasNextChar()) {
    char c = StdIn.readChar();
    StdOut.print(c);
}
```

Reading lines from standard input. You can use the following two methods to read lines from standard input:

- `hasNextLine()`
- `readLine()`

The first method returns true if standard input has more input (including whitespace). The second method reads and returns the remaining portion of the next line of input on standard input (possibly whitespace), discarding the trailing line separator.

A *line separator* is defined to be one of the following strings: `\n` (Linux), `\r` (old Macintosh), `\r\n` (Windows), `\u2028`, `\u2029`, or `\u0085`.

As an example, the following code fragment reads text from standard input, one line at a time, and prints it to standard output.

```
while (StdIn.hasNextLine()) {
    String line = StdIn.readLine();
    StdOut.println(line);
}
```

Reading a sequence of values of the same type from standard input. You can use the following methods to read a sequence numbers, strings, or booleans (all of the same type) from standard input:

- `readAllDoubles()`
- `readAllInts()`
- `readAllLongs()`
- `readAllStrings()`
- `readAllLines()`
- `readAll()`

The first three methods read of all of remaining token on standard input and converts the tokens to values of the specified type, as in the corresponding `readDouble`, `readInt`, and `readString()` methods. The `readAllLines()` method reads all remaining lines on standard input and returns them as an array of strings. The `readAll()` method reads all remaining input on standard input and returns it as a string.

As an example, the following code fragment reads all of the remaining tokens from standard input and returns them as an array of strings.

```
String[] words = StdIn.readAllStrings();
```

Differences with Scanner. `StdIn` and `Scanner` are both designed to parse tokens and convert them to primitive types and strings. The main differences are summarized below:

- `StdIn` is a set of static methods and reads reads input from only standard input. It is suitable for use before a programmer knows about objects. See `In` for an object-oriented version that handles input from files, URLs, and sockets.
- `StdIn` uses whitespace as the delimiter pattern that separates tokens. `Scanner` supports arbitrary delimiter patterns.
- `StdIn` coerces the character-set encoding to UTF-8, which is the most widely used character encoding for Unicode.

- `StdIn` coerces the locale to `Locale.US`, for consistency with `StdOut`, `Double.parseDouble(String)`, and floating-point literals.
- `StdIn` has convenient methods for reading a single character; reading in sequences of integers, doubles, or strings; and reading in all of the remaining input.

Historical note: `StdIn` preceded `Scanner`; when `Scanner` was introduced, this class was re-implemented to use `Scanner`.

Using standard input. Standard input is a fundamental operating system abstraction on Mac OS X, Windows, and Linux. The methods in `StdIn` are *blocking*, which means that they will wait until you enter input on standard input. If your program has a loop that repeats until standard input is empty, you must signal that the input is finished. To do so, depending on your operating system and IDE, use either `<Ctrl-d>` or `<Ctrl-z>`, on its own line. If you are redirecting standard input from a file, you will not need to do anything to signal that the input is finished.

Known bugs. Java's UTF-8 encoding does not recognize the optional byte-order mask. If the input begins with the optional byte-order mask, `StdIn` will have an extra character `\uFEFF` at the beginning.

Reference. For additional documentation, see Section 1.5 of *Computer Science: An Interdisciplinary Approach* by Robert Sedgewick and Kevin Wayne.

Author:

Robert Sedgewick, Kevin Wayne, David Pritchard

Method Summary

All Methods | Static Methods | Concrete Methods

Modifier and Type	Method	Description
static boolean	**hasNextChar**()	Returns true if standard input has more input (including whitespace).
static boolean	**hasNextLine**()	Returns true if standard input has a next line.
static boolean	**isEmpty**()	Returns true if standard input is empty (except possibly for whitespace).

static void	**main**(**String**[] args)	Interactive test of basic functionality.
static **String**	**readAll**()	Reads and returns the remainder of the input, as a string.
static double[]	**readAllDoubles**()	Reads all remaining tokens from standard input, parses them as doubles, and returns them as an array of doubles.
static int[]	**readAllInts**()	Reads all remaining tokens from standard input, parses them as integers, and returns them as an array of integers.
static **String**[]	**readAllLines**()	Reads all remaining lines from standard input and returns them as an array of strings.
static long[]	**readAllLongs**()	Reads all remaining tokens from standard input, parses them as longs, and returns them as an array of longs.
static **String**[]	**readAllStrings**()	Reads all remaining tokens from standard input and returns them as an array of strings.
static boolean	**readBoolean**()	Reads the next token from standard input, parses it as a boolean, and returns the boolean.
static byte	**readByte**()	Reads the next token from standard input, parses it as a byte, and returns the byte.
static char	**readChar**()	Reads and returns the next character.
static double	**readDouble**()	Reads the next token from standard input, parses it as a double, and returns the double.
static float	**readFloat**()	Reads the next token from standard input, parses it as a float, and returns the float.
static int	**readInt**()	Reads the next token from standard input, parses it as an integer, and returns the integer.
static **String**	**readLine**()	Reads and returns the next line, excluding the line separator if present.
static long	**readLong**()	Reads the next token from standard input, parses it as a long integer, and returns the long integer.

`static short` **`readShort`**`()`	Reads the next token from standard input, parses it as a short integer, and returns the short integer.
`static String` **`readString`**`()`	Reads the next token from standard input and returns it as a `String`.

Methods inherited from class java.lang.Object

`clone`, `equals`, `getClass`, `hashCode`, `notify`, `notifyAll`, `toString`, `wait`, `wait`, `wait`

Method Detail

isEmpty

```
public static boolean isEmpty()
```

Returns true if standard input is empty (except possibly for whitespace). Use this method to know whether the next call to `readString()`, `readDouble()`, etc. will succeed.

Returns:

`true` if standard input is empty (except possibly for whitespace); `false` otherwise

hasNextLine

```
public static boolean hasNextLine()
```

Returns true if standard input has a next line. Use this method to know whether the next call to `readLine()` will succeed. This method is functionally equivalent to `hasNextChar()`.

Returns:

`true` if standard input has more input (including whitespace); `false` otherwise

hasNextChar

```
public static boolean hasNextChar()
```

Returns true if standard input has more input (including whitespace). Use this method to know whether the next call to `readChar()` will succeed. This method is functionally equivalent to `hasNextLine()`.

Returns:

`true` if standard input has more input (including whitespace); `false` otherwise

readLine

```
public static String readLine()
```

Reads and returns the next line, excluding the line separator if present.

Returns:

the next line, excluding the line separator if present; `null` if no such line

readChar

```
public static char readChar()
```

Reads and returns the next character.

Returns:

the next `char`

Throws:

`NoSuchElementException` - if standard input is empty

readAll

```
public static String readAll()
```

Reads and returns the remainder of the input, as a string.

Returns:

the remainder of the input, as a string

Throws:

`NoSuchElementException` - if standard input is empty

readString

```
public static String readString()
```

Reads the next token from standard input and returns it as a `String`.

Returns:

the next `String`

Throws:

`NoSuchElementException` - if standard input is empty

readInt

```
public static int readInt()
```

Reads the next token from standard input, parses it as an integer, and returns the integer.

Returns:

the next integer on standard input

Throws:

`NoSuchElementException` - if standard input is empty

`InputMismatchException` - if the next token cannot be parsed as an `int`

readDouble

```
public static double readDouble()
```

Reads the next token from standard input, parses it as a double, and returns the double.

Returns:

the next double on standard input

Throws:

`NoSuchElementException` - if standard input is empty

`InputMismatchException` - if the next token cannot be parsed as a `double`

readFloat

```
public static float readFloat()
```

Reads the next token from standard input, parses it as a float, and returns the float.

Returns:

the next float on standard input

Throws:

`NoSuchElementException` - if standard input is empty

`InputMismatchException` - if the next token cannot be parsed as a `float`

readLong

```
public static long readLong()
```

Reads the next token from standard input, parses it as a long integer, and returns the long integer.

Returns:

the next long integer on standard input

Throws:

`NoSuchElementException` - if standard input is empty

`InputMismatchException` - if the next token cannot be parsed as a `long`

readShort

```
public static short readShort()
```

Reads the next token from standard input, parses it as a short integer, and returns the short integer.

Returns:

the next short integer on standard input

Throws:

`NoSuchElementException` - if standard input is empty

`InputMismatchException` - if the next token cannot be parsed as a `short`

readByte

```
public static byte readByte()
```

Reads the next token from standard input, parses it as a byte, and returns the byte.

Returns:

the next byte on standard input

Throws:

`NoSuchElementException` - if standard input is empty

`InputMismatchException` - if the next token cannot be parsed as a `byte`

readBoolean

```
public static boolean readBoolean()
```

Reads the next token from standard input, parses it as a boolean, and returns the boolean.

Returns:

the next boolean on standard input

Throws:

`NoSuchElementException` - if standard input is empty

`InputMismatchException` - if the next token cannot be parsed as a `boolean`: `true` or `1` for true, and `false` or `0` for false, ignoring case

readAllStrings

```
public static String[] readAllStrings()
```

Reads all remaining tokens from standard input and returns them as an array of strings.

Returns:

all remaining tokens on standard input, as an array of strings

readAllLines

```
public static String[] readAllLines()
```

Reads all remaining lines from standard input and returns them as an array of strings.

Returns:

all remaining lines on standard input, as an array of strings

readAllInts

```
public static int[] readAllInts()
```

Reads all remaining tokens from standard input, parses them as integers, and returns them as an array of integers.

Returns:

all remaining integers on standard input, as an array

Throws:

`InputMismatchException` - if any token cannot be parsed as an `int`

readAllLongs

```
public static long[] readAllLongs()
```

Reads all remaining tokens from standard input, parses them as longs, and returns them as an array of longs.

Returns:

all remaining longs on standard input, as an array

Throws:

`InputMismatchException` - if any token cannot be parsed as a `long`

readAllDoubles

```
public static double[] readAllDoubles()
```

Reads all remaining tokens from standard input, parses them as doubles, and returns them as an array of doubles.

Returns:

all remaining doubles on standard input, as an array

Throws:

`InputMismatchException` - if any token cannot be parsed as a `double`

main

```
public static void main(String[] args)
```

Interactive test of basic functionality.

Parameters:

`args` - the command-line arguments