Module java.base
Package java.lang

Class StringBuilder

java.lang.Object
    java.lang.StringBuilder

All Implemented Interfaces:

`Serializable`, `Appendable`, `CharSequence`, `Comparable<StringBuilder>`

```
public final class StringBuilder
extends Object
implements Serializable, Comparable<StringBuilder>, CharSequence
```

A mutable sequence of characters. This class provides an API compatible with `StringBuffer`, but with no guarantee of synchronization. This class is designed for use as a drop-in replacement for `StringBuffer` in places where the string buffer was being used by a single thread (as is generally the case). Where possible, it is recommended that this class be used in preference to `StringBuffer` as it will be faster under most implementations.

The principal operations on a `StringBuilder` are the `append` and `insert` methods, which are overloaded so as to accept data of any type. Each effectively converts a given datum to a string and then appends or inserts the characters of that string to the string builder. The `append` method always adds these characters at the end of the builder; the `insert` method adds the characters at a specified point.

For example, if `z` refers to a string builder object whose current contents are "`start`", then the method call `z.append("le")` would cause the string builder to contain "`startle`", whereas `z.insert(4, "le")` would alter the string builder to contain "`starlet`".

In general, if sb refers to an instance of a `StringBuilder`, then `sb.append(x)` has the same effect as `sb.insert(sb.length(), x)`.

Every string builder has a capacity. As long as the length of the character sequence contained in the string builder does not exceed the capacity, it is not necessary to allocate a new internal buffer. If the internal buffer overflows, it is automatically made larger.

Instances of `StringBuilder` are not safe for use by multiple threads. If such synchronization is required then it is recommended that `StringBuffer` be used.

Unless otherwise noted, passing a `null` argument to a constructor or method in this class will cause a `NullPointerException` to be thrown.

API Note:

`StringBuilder` implements `Comparable` but does not override `equals`. Thus, the natural ordering of `StringBuilder` is inconsistent with equals. Care should be exercised if `StringBuilder` objects are used as keys in a `SortedMap` or elements in a `SortedSet`. See `Comparable`, `SortedMap`, or `SortedSet` for more information.

Since:

1.5

See Also:

StringBuffer, String, Serialized Form

Constructor Summary

Constructors

Constructor	Description
`StringBuilder()`	Constructs a string builder with no characters in it and an initial capacity of 16 characters.
`StringBuilder(int capacity)`	Constructs a string builder with no characters in it and an initial capacity specified by the `capacity` argument.
`StringBuilder(CharSequence seq)`	Constructs a string builder that contains the same characters as the specified `CharSequence`.
`StringBuilder(String str)`	Constructs a string builder initialized to the contents of the specified string.

Method Summary

All Methods Instance Methods Concrete Methods

Modifier and Type	Method	Description
`StringBuilder`	`append(boolean b)`	Appends the string representation of the `boolean` argument to the sequence.
`StringBuilder`	`append(char c)`	Appends the string representation of the `char` argument to this sequence.
`StringBuilder`	`append(char[] str)`	Appends the string representation of the `char` array argument to this sequence.
`StringBuilder`	`append(char[] str, int offset, int len)`	Appends the string representation of a subarray of the `char` array argument to this sequence.
`StringBuilder`	`append(double d)`	Appends the string representation of the `double` argument to this sequence.
`StringBuilder`	`append(float f)`	Appends the string representation of the `float` argument to this sequence.

StringBuilder	append(int i)	Appends the string representation of the int argument to this sequence.
StringBuilder	append(long lng)	Appends the string representation of the long argument to this sequence.
StringBuilder	append(CharSequence s)	Appends the specified character sequence to this Appendable.
StringBuilder	append(CharSequence s, int start, int end)	Appends a subsequence of the specified CharSequence to this sequence.
StringBuilder	append(Object obj)	Appends the string representation of the Object argument.
StringBuilder	append(String str)	Appends the specified string to this character sequence.
StringBuilder	append(StringBuffer sb)	Appends the specified StringBuffer to this sequence.
StringBuilder	appendCodePoint(int codePoint)	Appends the string representation of the codePoint argument to this sequence.
int	capacity()	Returns the current capacity.
char	charAt(int index)	Returns the char value in this sequence at the specified index.
IntStream	chars()	Returns a stream of int zero-extending the char values from this sequence.
int	codePointAt(int index)	Returns the character (Unicode code point) at the specified index.
int	codePointBefore(int index)	Returns the character (Unicode code point) before the specified index.
int	codePointCount(int beginIndex, int endIndex)	Returns the number of Unicode code points in the specified text range of this sequence.
IntStream	codePoints()	Returns a stream of code point values from this sequence.
int	compareTo(StringBuilder another)	Compares two StringBuilder instances lexicographically.

StringBuilder	delete(int start, int end)	Removes the characters in a substring of this sequence.
StringBuilder	deleteCharAt(int index)	Removes the char at the specified position in this sequence.
void	ensureCapacity (int minimumCapacity)	Ensures that the capacity is at least equal to the specified minimum.
void	getChars(int srcBegin, int srcEnd, char[] dst, int dstBegin)	Characters are copied from this sequence into the destination character array dst.
int	indexOf(String str)	Returns the index within this string of the first occurrence of the specified substring.
int	indexOf(String str, int fromIndex)	Returns the index within this string of the first occurrence of the specified substring, starting at the specified index.
StringBuilder	insert(int offset, boolean b)	Inserts the string representation of the boolean argument into this sequence.
StringBuilder	insert(int offset, char c)	Inserts the string representation of the char argument into this sequence.
StringBuilder	insert(int offset, char[] str)	Inserts the string representation of the char array argument into this sequence.
StringBuilder	insert(int index, char[] str, int offset, int len)	Inserts the string representation of a subarray of the str array argument into this sequence.
StringBuilder	insert(int offset, double d)	Inserts the string representation of the double argument into this sequence.
StringBuilder	insert(int offset, float f)	Inserts the string representation of the float argument into this sequence.
StringBuilder	insert(int offset, int i)	Inserts the string representation of the second int argument into this sequence.
StringBuilder	insert(int offset, long l)	Inserts the string representation of the long argument into this sequence.

`StringBuilder`	`insert(int dstOffset, CharSequence s)`	Inserts the specified `CharSequence` into this sequence.
`StringBuilder`	`insert(int dstOffset, CharSequence s, int start, int end)`	Inserts a subsequence of the specified `CharSequence` into this sequence.
`StringBuilder`	`insert(int offset, Object obj)`	Inserts the string representation of the `Object` argument into this character sequence.
`StringBuilder`	`insert(int offset, String str)`	Inserts the string into this character sequence.
`int`	`lastIndexOf(String str)`	Returns the index within this string of the last occurrence of the specified substring.
`int`	`lastIndexOf(String str, int fromIndex)`	Returns the index within this string of the last occurrence of the specified substring, searching backward starting at the specified index.
`int`	`length()`	Returns the length (character count).
`int`	`offsetByCodePoints(int index, int codePointOffset)`	Returns the index within this sequence that is offset from the given `index` by `codePointOffset` code points.
`StringBuilder`	`replace(int start, int end, String str)`	Replaces the characters in a substring of this sequence with characters in the specified `String`.
`StringBuilder`	`reverse()`	Causes this character sequence to be replaced by the reverse of the sequence.
`void`	`setCharAt(int index, char ch)`	The character at the specified index is set to `ch`.
`void`	`setLength(int newLength)`	Sets the length of the character sequence.
`CharSequence`	`subSequence(int start, int end)`	Returns a new character sequence that is a subsequence of this sequence.
`String`	`substring(int start)`	Returns a new `String` that contains a subsequence of

		characters currently contained in this character sequence.
`String`	`substring(int start, int end)`	Returns a new `String` that contains a subsequence of characters currently contained in this sequence.
`String`	`toString()`	Returns a string representing the data in this sequence.
`void`	`trimToSize()`	Attempts to reduce storage used for the character sequence.

Methods declared in class java.lang.Object

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

Methods declared in interface java.lang.CharSequence

`charAt`, `chars`, `codePoints`, `isEmpty`, `length`, `subSequence`

Constructor Details

StringBuilder

`public StringBuilder()`

Constructs a string builder with no characters in it and an initial capacity of 16 characters.

StringBuilder

`public StringBuilder(int capacity)`

Constructs a string builder with no characters in it and an initial capacity specified by the `capacity` argument.

Parameters:

`capacity` - the initial capacity.

Throws:

`NegativeArraySizeException` - if the `capacity` argument is less than `0`.

StringBuilder

`public StringBuilder(String str)`

Constructs a string builder initialized to the contents of the specified string. The initial capacity of the string builder is `16` plus the length of the string argument.

Parameters:

`str` - the initial contents of the buffer.

StringBuilder

```
public StringBuilder(CharSequence seq)
```

Constructs a string builder that contains the same characters as the specified `CharSequence`. The initial capacity of the string builder is `16` plus the length of the `CharSequence` argument.

Parameters:

`seq` - the sequence to copy.

Method Details

compareTo

```
public int compareTo(StringBuilder another)
```

Compares two `StringBuilder` instances lexicographically. This method follows the same rules for lexicographical comparison as defined in the CharSequence.compare(this, another) method.

For finer-grained, locale-sensitive String comparison, refer to `Collator`.

Specified by:

`compareTo` in interface `Comparable`<`StringBuilder`>

Parameters:

`another` - the `StringBuilder` to be compared with

Returns:

the value `0` if this `StringBuilder` contains the same character sequence as that of the argument `StringBuilder`; a negative integer if this `StringBuilder` is lexicographically less than the `StringBuilder` argument; or a positive integer if this `StringBuilder` is lexicographically greater than the `StringBuilder` argument.

Since:

11

append

```
public StringBuilder append(Object obj)
```

Appends the string representation of the `Object` argument.

The overall effect is exactly as if the argument were converted to a string by the method `String.valueOf(Object)`, and the characters of that string were then `appended` to this character sequence.

Parameters:

`obj` - an `Object`.

Returns:

a reference to this object.

append

```
public StringBuilder append(String str)
```

Appends the specified string to this character sequence.

The characters of the `String` argument are appended, in order, increasing the length of this sequence by the length of the argument. If `str` is `null`, then the four characters `"null"` are appended.

Let *n* be the length of this character sequence just prior to execution of the `append` method. Then the character at index *k* in the new character sequence is equal to the character at index *k* in the old character sequence, if *k* is less than *n*; otherwise, it is equal to the character at index *k-n* in the argument `str`.

Parameters:

`str` - a string.

Returns:

a reference to this object.

append

```
public StringBuilder append(StringBuffer sb)
```

Appends the specified `StringBuffer` to this sequence.

The characters of the `StringBuffer` argument are appended, in order, to this sequence, increasing the length of this sequence by the length of the argument. If `sb` is `null`, then the four characters `"null"` are appended to this sequence.

Let *n* be the length of this character sequence just prior to execution of the `append` method. Then the character at index *k* in the new character sequence is equal to the character at index *k* in the old character sequence, if *k* is less than *n*; otherwise, it is equal to the character at index *k-n* in the argument `sb`.

Parameters:

`sb` - the `StringBuffer` to append.

Returns:

a reference to this object.

append

```
public StringBuilder append(CharSequence s)
```

Description copied from interface: `Appendable`

Appends the specified character sequence to this `Appendable`.

Depending on which class implements the character sequence `csq`, the entire sequence may not be appended. For instance, if `csq` is a `CharBuffer` then the subsequence to append is defined by the buffer's position and limit.

Specified by:

`append` in interface `Appendable`

Parameters:

`s` - The character sequence to append. If `csq` is `null`, then the four characters `"null"` are appended to this Appendable.

Returns:

A reference to this `Appendable`

append

```
public StringBuilder append(CharSequence s,
                            int start,
                            int end)
```

Appends a subsequence of the specified `CharSequence` to this sequence.

Characters of the argument `s`, starting at index `start`, are appended, in order, to the contents of this sequence up to the (exclusive) index `end`. The length of this sequence is increased by the value of `end - start`.

Let *n* be the length of this character sequence just prior to execution of the `append` method. Then the character at index *k* in this character sequence becomes equal to the character at index *k* in this sequence, if *k* is less than *n*; otherwise, it is equal to the character at index *k+start-n* in the argument `s`.

If `s` is `null`, then this method appends characters as if the s parameter was a sequence containing the four characters `"null"`.

Specified by:

`append` in interface `Appendable`

Parameters:

`s` - the sequence to append.

`start` - the starting index of the subsequence to be appended.

`end` - the end index of the subsequence to be appended.

Returns:

a reference to this object.

Throws:

`IndexOutOfBoundsException` - if `start` is negative, or `start` is greater than `end` or `end` is greater than `s.length()`

append

```
public StringBuilder append(char[] str)
```

Appends the string representation of the `char` array argument to this sequence.

The characters of the array argument are appended, in order, to the contents of this sequence. The length of this sequence increases by the length of the argument.

The overall effect is exactly as if the argument were converted to a string by the method `String.valueOf(char[])`, and the characters of that string were then `appended` to this character sequence.

Parameters:

`str` - the characters to be appended.

Returns:

a reference to this object.

append

```
public StringBuilder append(char[] str,
                            int offset,
                            int len)
```

Appends the string representation of a subarray of the `char` array argument to this sequence.

Characters of the `char` array `str`, starting at index `offset`, are appended, in order, to the contents of this sequence. The length of this sequence increases by the value of `len`.

The overall effect is exactly as if the arguments were converted to a string by the method `String.valueOf(char[],int,int)`, and the characters of that string were then `appended` to this character sequence.

Parameters:

`str` - the characters to be appended.

`offset` - the index of the first `char` to append.

`len` - the number of `char`s to append.

Returns:

a reference to this object.

Throws:

`IndexOutOfBoundsException` - if `offset < 0` or `len < 0` or `offset+len > str.length`

append

```
public StringBuilder append(boolean b)
```

Appends the string representation of the `boolean` argument to the sequence.

The overall effect is exactly as if the argument were converted to a string by the method `String.valueOf(boolean)`, and the characters of that string were then appended to this character sequence.

Parameters:

`b` - a `boolean`.

Returns:

a reference to this object.

append

```
public StringBuilder append(char c)
```

Appends the string representation of the `char` argument to this sequence.

The argument is appended to the contents of this sequence. The length of this sequence increases by `1`.

The overall effect is exactly as if the argument were converted to a string by the method `String.valueOf(char)`, and the character in that string were then appended to this character sequence.

Specified by:

`append` in interface `Appendable`

Parameters:

`c` - a `char`.

Returns:

a reference to this object.

append

```
public StringBuilder append(int i)
```

Appends the string representation of the `int` argument to this sequence.

The overall effect is exactly as if the argument were converted to a string by the method `String.valueOf(int)`, and the characters of that string were then appended to this character sequence.

Parameters:

`i` - an `int`.

Returns:

a reference to this object.

append

```
public StringBuilder append(long lng)
```

Appends the string representation of the `long` argument to this sequence.

The overall effect is exactly as if the argument were converted to a string by the method `String.valueOf(long)`, and the characters of that string were then appended to this character sequence.

Parameters:

`lng` - a `long`.

Returns:

a reference to this object.

append

```
public StringBuilder append(float f)
```

Appends the string representation of the `float` argument to this sequence.

The overall effect is exactly as if the argument were converted to a string by the method `String.valueOf(float)`, and the characters of that string were then appended to this character sequence.

Parameters:

`f` - a `float`.

Returns:

a reference to this object.

append

```
public StringBuilder append(double d)
```

Appends the string representation of the `double` argument to this sequence.

The overall effect is exactly as if the argument were converted to a string by the method `String.valueOf(double)`, and the characters of that string were then appended to this character sequence.

Parameters:

`d` - a `double`.

Returns:

a reference to this object.

appendCodePoint

```
public StringBuilder appendCodePoint(int codePoint)
```

Appends the string representation of the `codePoint` argument to this sequence.

The argument is appended to the contents of this sequence. The length of this sequence increases by `Character.charCount(codePoint)`.

The overall effect is exactly as if the argument were converted to a `char` array by the method `Character.toChars(int)` and the character in that array were then appended to this character sequence.

Parameters:

`codePoint` - a Unicode code point

Returns:

a reference to this object.

Since:

1.5

delete

```
public StringBuilder delete(int start,
                            int end)
```

Removes the characters in a substring of this sequence. The substring begins at the specified `start` and extends to the character at index `end - 1` or to the end of the sequence if no such character exists. If `start` is equal to `end`, no changes are made.

Parameters:

`start` - The beginning index, inclusive.

`end` - The ending index, exclusive.

Returns:

This object.

Throws:

`StringIndexOutOfBoundsException` - if `start` is negative, greater than `length()`, or greater than `end`.

deleteCharAt

```
public StringBuilder deleteCharAt(int index)
```

Removes the `char` at the specified position in this sequence. This sequence is shortened by one `char`.

Note: If the character at the given index is a supplementary character, this method does not remove the entire character. If correct handling of supplementary characters is required, determine the number of `char`s to remove by calling `Character.charCount(thisSequence.codePointAt(index))`, where `thisSequence` is this sequence.

Parameters:

`index` - Index of `char` to remove

Returns:

This object.

Throws:

`StringIndexOutOfBoundsException` - if the `index` is negative or greater than or equal to `length()`.

replace

```
public StringBuilder replace(int start,
                             int end,
                             String str)
```

Replaces the characters in a substring of this sequence with characters in the specified `String`. The substring begins at the specified `start` and extends to the character at index `end - 1` or to the end of the sequence if no such character exists. First the characters in the substring are removed and then the specified `String` is inserted at `start`. (This sequence will be lengthened to accommodate the specified String if necessary.)

Parameters:

`start` - The beginning index, inclusive.

`end` - The ending index, exclusive.

`str` - String that will replace previous contents.

Returns:

This object.

Throws:

`StringIndexOutOfBoundsException` - if `start` is negative, greater than `length()`, or greater than `end`.

insert

```
public StringBuilder insert(int index,
                            char[] str,
                            int offset,
                            int len)
```

Inserts the string representation of a subarray of the `str` array argument into this sequence. The subarray begins at the specified `offset` and extends `len` `char`s. The characters of the subarray are inserted into this sequence at the position indicated by `index`. The length of this sequence increases by `len` `char`s.

Parameters:

`index` - position at which to insert subarray.

`str` - A `char` array.

`offset` - the index of the first `char` in subarray to be inserted.

`len` - the number of `char`s in the subarray to be inserted.

Returns:

This object

Throws:

`StringIndexOutOfBoundsException` - if `index` is negative or greater than `length()`, or `offset` or `len` are negative, or `(offset+len)` is greater than `str.length`.

insert

```
public StringBuilder insert(int offset,
                            Object obj)
```

Inserts the string representation of the `Object` argument into this character sequence.

The overall effect is exactly as if the second argument were converted to a string by the method `String.valueOf(Object)`, and the characters of that string were then `inserted` into this character sequence at the indicated offset.

The `offset` argument must be greater than or equal to `0`, and less than or equal to the length of this sequence.

Parameters:

`offset` - the offset.

`obj` - an `Object`.

Returns:

a reference to this object.

Throws:

`StringIndexOutOfBoundsException` - if the offset is invalid.

insert

```
public StringBuilder insert(int offset,
                            String str)
```

Inserts the string into this character sequence.

The characters of the `String` argument are inserted, in order, into this sequence at the indicated offset, moving up any characters originally above that position and increasing the length of this sequence by the length of the argument. If `str` is `null`, then the four characters `"null"` are inserted into this sequence.

The character at index *k* in the new character sequence is equal to:

- the character at index *k* in the old character sequence, if *k* is less than `offset`
- the character at index *k*`-offset` in the argument `str`, if *k* is not less than `offset` but is less than `offset+str.length()`
- the character at index *k*`-str.length()` in the old character sequence, if *k* is not less than `offset+str.length()`

The `offset` argument must be greater than or equal to `0`, and less than or equal to the length of this sequence.

Parameters:

`offset` - the offset.

`str` - a string.

Returns:

a reference to this object.

Throws:

`StringIndexOutOfBoundsException` - if the offset is invalid.

insert

```
public StringBuilder insert(int offset,
                            char[] str)
```

Inserts the string representation of the `char` array argument into this sequence.

The characters of the array argument are inserted into the contents of this sequence at the position indicated by `offset`. The length of this sequence increases by the length of the argument.

The overall effect is exactly as if the second argument were converted to a string by the method `String.valueOf(char[])`, and the characters of that string were then `inserted` into this character sequence at the indicated offset.

The `offset` argument must be greater than or equal to `0`, and less than or equal to the `length` of this sequence.

Parameters:

`offset` - the offset.

`str` - a character array.

Returns:

a reference to this object.

Throws:

`StringIndexOutOfBoundsException` - if the offset is invalid.

insert

```
public StringBuilder insert(int dstOffset,
                            CharSequence s)
```

Inserts the specified `CharSequence` into this sequence.

The characters of the `CharSequence` argument are inserted, in order, into this sequence at the indicated offset, moving up any characters originally above that position and increasing the length of this sequence by the length of the argument s.

The result of this method is exactly the same as if it were an invocation of this object's `insert`(dstOffset, s, 0, s.length()) method.

If `s` is `null`, then the four characters `"null"` are inserted into this sequence.

Parameters:

`dstOffset` - the offset.

`s` - the sequence to be inserted

Returns:

a reference to this object.

Throws:

`IndexOutOfBoundsException` - if the offset is invalid.

insert

```
public StringBuilder insert(int dstOffset,
                            CharSequence s,
                            int start,
                            int end)
```

Inserts a subsequence of the specified `CharSequence` into this sequence.

The subsequence of the argument `s` specified by `start` and `end` are inserted, in order, into this sequence at the specified destination offset, moving up any characters originally above that position. The length of this sequence is increased by `end - start`.

The character at index *k* in this sequence becomes equal to:

- the character at index *k* in this sequence, if *k* is less than `dstOffset`
- the character at index *k*`+start-dstOffset` in the argument `s`, if *k* is greater than or equal to `dstOffset` but is less than `dstOffset+end-start`
- the character at index *k*`-(end-start)` in this sequence, if *k* is greater than or equal to `dstOffset+end-start`

The `dstOffset` argument must be greater than or equal to `0`, and less than or equal to the length of this sequence.

The start argument must be nonnegative, and not greater than `end`.

The end argument must be greater than or equal to `start`, and less than or equal to the length of s.

If `s` is `null`, then this method inserts characters as if the s parameter was a sequence containing the four characters `"null"`.

Parameters:

`dstOffset` - the offset in this sequence.

`s` - the sequence to be inserted.

`start` - the starting index of the subsequence to be inserted.

`end` - the end index of the subsequence to be inserted.

Returns:

a reference to this object.

Throws:

`IndexOutOfBoundsException` - if `dstOffset` is negative or greater than `this.length()`, or `start` or `end` are negative, or `start` is greater than `end` or `end` is greater than `s.length()`

insert

```
public StringBuilder insert(int offset,
                            boolean b)
```

Inserts the string representation of the `boolean` argument into this sequence.

The overall effect is exactly as if the second argument were converted to a string by the method `String.valueOf(boolean)`, and the characters of that string were then `inserted` into this character sequence at the indicated offset.

The `offset` argument must be greater than or equal to `0`, and less than or equal to the length of this sequence.

Parameters:

`offset` - the offset.

`b` - a `boolean`.

Returns:

a reference to this object.

Throws:

`StringIndexOutOfBoundsException` - if the offset is invalid.

insert

```
public StringBuilder insert(int offset,
                            char c)
```

Inserts the string representation of the `char` argument into this sequence.

The overall effect is exactly as if the second argument were converted to a string by the method `String.valueOf(char)`, and the character in that string were then `inserted` into this character sequence at the indicated offset.

The `offset` argument must be greater than or equal to `0`, and less than or equal to the length of this sequence.

Parameters:

`offset` - the offset.

`c` - a `char`.

Returns:

a reference to this object.

Throws:

`IndexOutOfBoundsException` - if the offset is invalid.

insert

```
public StringBuilder insert(int offset,
                            int i)
```

Inserts the string representation of the second `int` argument into this sequence.

The overall effect is exactly as if the second argument were converted to a string by the method `String.valueOf(int)`, and the characters of that string were then `inserted` into this character sequence at the indicated offset.

The `offset` argument must be greater than or equal to `0`, and less than or equal to the length of this sequence.

Parameters:

`offset` - the offset.

`i` - an `int`.

Returns:

a reference to this object.

Throws:

`StringIndexOutOfBoundsException` - if the offset is invalid.

insert

```
public StringBuilder insert(int offset,
                            long l)
```

Inserts the string representation of the `long` argument into this sequence.

The overall effect is exactly as if the second argument were converted to a string by the method `String.valueOf(long)`, and the characters of that string were then `inserted` into this character sequence at the indicated offset.

The `offset` argument must be greater than or equal to `0`, and less than or equal to the length of this sequence.

Parameters:

`offset` - the offset.

`l` - a `long`.

Returns:

a reference to this object.

Throws:

`StringIndexOutOfBoundsException` - if the offset is invalid.

insert

```
public StringBuilder insert(int offset,
                            float f)
```

Inserts the string representation of the `float` argument into this sequence.

The overall effect is exactly as if the second argument were converted to a string by the method `String.valueOf(float)`, and the characters of that string were then `inserted` into this character sequence at the indicated offset.

The `offset` argument must be greater than or equal to `0`, and less than or equal to the length of this sequence.

Parameters:

`offset` - the offset.

`f` - a `float`.

Returns:

a reference to this object.

Throws:

`StringIndexOutOfBoundsException` - if the offset is invalid.

insert

```
public StringBuilder insert(int offset,
                            double d)
```

Inserts the string representation of the `double` argument into this sequence.

The overall effect is exactly as if the second argument were converted to a string by the method `String.valueOf(double)`, and the characters of that string were then `inserted` into this character sequence at the indicated offset.

The `offset` argument must be greater than or equal to `0`, and less than or equal to the length of this sequence.

Parameters:

`offset` - the offset.

`d` - a `double`.

Returns:

a reference to this object.

Throws:

`StringIndexOutOfBoundsException` - if the offset is invalid.

indexOf

```
public int indexOf(String str)
```

Returns the index within this string of the first occurrence of the specified substring.

The returned index is the smallest value `k` for which:

```
 this.toString().startsWith(str, k)
```

If no such value of `k` exists, then `-1` is returned.

Parameters:

`str` - the substring to search for.

Returns:

the index of the first occurrence of the specified substring, or `-1` if there is no such occurrence.

indexOf

```
public int indexOf(String str,
                   int fromIndex)
```

Returns the index within this string of the first occurrence of the specified substring, starting at the specified index.

The returned index is the smallest value `k` for which:

```
     k >= Math.min(fromIndex, this.length()) &&
                   this.toString().startsWith(str, k)
```

If no such value of `k` exists, then `-1` is returned.

Parameters:

`str` - the substring to search for.

`fromIndex` - the index from which to start the search.

Returns:

the index of the first occurrence of the specified substring, starting at the specified index, or `-1` if there is no such occurrence.

lastIndexOf

```
public int lastIndexOf(String str)
```

Returns the index within this string of the last occurrence of the specified substring. The last occurrence of the empty string "" is considered to occur at the index value `this.length()`.

The returned index is the largest value `k` for which:

```
 this.toString().startsWith(str, k)
```

If no such value of `k` exists, then `-1` is returned.

Parameters:

`str` - the substring to search for.

Returns:

the index of the last occurrence of the specified substring, or `-1` if there is no such occurrence.

lastIndexOf

```
public int lastIndexOf(String str,
                       int fromIndex)
```

Returns the index within this string of the last occurrence of the specified substring, searching backward starting at the specified index.

The returned index is the largest value `k` for which:

```
     k <= Math.min(fromIndex, this.length()) &&
                   this.toString().startsWith(str, k)
```

If no such value of `k` exists, then `-1` is returned.

Parameters:

`str` - the substring to search for.

`fromIndex` - the index to start the search from.

Returns:

the index of the last occurrence of the specified substring, searching backward from the specified index, or `-1` if there is no such occurrence.

reverse

```
public StringBuilder reverse()
```

Causes this character sequence to be replaced by the reverse of the sequence. If there are any surrogate pairs included in the sequence, these are treated as single characters for the reverse operation. Thus, the order of the high-low surrogates is never reversed. Let *n* be the character length of this character sequence (not the length in `char` values) just prior to execution of the `reverse` method. Then the character at index *k* in the new character sequence is equal to the character at index *n-k-1* in the old character sequence.

Note that the reverse operation may result in producing surrogate pairs that were unpaired low-surrogates and high-surrogates before the operation. For example, reversing "\uDC00\uD800" produces "\uD800\uDC00" which is a valid surrogate pair.

Returns:

a reference to this object.

toString

```
public String toString()
```

Returns a string representing the data in this sequence. A new `String` object is allocated and initialized to contain the character sequence currently represented by this object. This `String` is then returned. Subsequent changes to this sequence do not affect the contents of the `String`.

Specified by:

`toString` in interface `CharSequence`

Returns:

a string representation of this sequence of characters.

length

`public int length()`

Returns the length (character count).

Specified by:

`length` in interface `CharSequence`

Returns:

the length of the sequence of characters currently represented by this object

capacity

`public int capacity()`

Returns the current capacity. The capacity is the number of characters that can be stored (including already written characters), beyond which an allocation will occur.

Returns:

the current capacity

ensureCapacity

`public void ensureCapacity(int minimumCapacity)`

Ensures that the capacity is at least equal to the specified minimum. If the current capacity is less than the argument, then a new internal array is allocated with greater capacity. The new capacity is the larger of:

- The `minimumCapacity` argument.
- Twice the old capacity, plus 2.

If the `minimumCapacity` argument is nonpositive, this method takes no action and simply returns. Note that subsequent operations on this object can reduce the actual capacity below that requested here.

Parameters:

`minimumCapacity` - the minimum desired capacity.

trimToSize

`public void trimToSize()`

Attempts to reduce storage used for the character sequence. If the buffer is larger than necessary to hold its current sequence of characters, then it may be resized to become more space efficient. Calling this method may, but is not required to, affect the value returned by a subsequent call to the `capacity()` method.

setLength

`public void setLength(int newLength)`

Sets the length of the character sequence. The sequence is changed to a new character sequence whose length is specified by the argument. For every nonnegative index *k* less than `newLength`, the character at index *k* in the new character sequence is the same as the character at index *k* in the old sequence if *k* is less than the length of the old character sequence; otherwise, it is the null character `'\u0000'`. In other words, if the `newLength` argument is less than the current length, the length is changed to the specified length.

If the `newLength` argument is greater than or equal to the current length, sufficient null characters (`'\u0000'`) are appended so that length becomes the `newLength` argument.

The `newLength` argument must be greater than or equal to `0`.

Parameters:

`newLength` - the new length

Throws:

`IndexOutOfBoundsException` - if the `newLength` argument is negative.

charAt

```
public char charAt(int index)
```

Returns the `char` value in this sequence at the specified index. The first `char` value is at index `0`, the next at index `1`, and so on, as in array indexing.

The index argument must be greater than or equal to `0`, and less than the length of this sequence.

If the `char` value specified by the index is a surrogate, the surrogate value is returned.

Specified by:

`charAt` in interface `CharSequence`

Parameters:

`index` - the index of the desired `char` value.

Returns:

the `char` value at the specified index.

Throws:

`IndexOutOfBoundsException` - if `index` is negative or greater than or equal to `length()`.

codePointAt

```
public int codePointAt(int index)
```

Returns the character (Unicode code point) at the specified index. The index refers to `char` values (Unicode code units) and ranges from `0` to `CharSequence.length() - 1`.

If the `char` value specified at the given index is in the high-surrogate range, the following index is less than the length of this sequence, and the `char` value at the following index is in the low-surrogate range, then the supplementary code point corresponding to this surrogate pair is returned. Otherwise, the `char` value at the given index is returned.

Parameters:

`index` - the index to the `char` values

Returns:

the code point value of the character at the `index`

Throws:

`IndexOutOfBoundsException` - if the `index` argument is negative or not less than the length of this sequence.

codePointBefore

```
public int codePointBefore(int index)
```

Returns the character (Unicode code point) before the specified index. The index refers to `char` values (Unicode code units) and ranges from `1` to `CharSequence.length()`.

If the `char` value at `(index - 1)` is in the low-surrogate range, `(index - 2)` is not negative, and the `char` value at `(index - 2)` is in the high-surrogate range, then the supplementary code point value of the surrogate pair is returned. If the `char` value at `index - 1` is an unpaired low-surrogate or a high-surrogate, the surrogate value is returned.

Parameters:

`index` - the index following the code point that should be returned

Returns:

the Unicode code point value before the given index.

Throws:

`IndexOutOfBoundsException` - if the `index` argument is less than 1 or greater than the length of this sequence.

codePointCount

```
public int codePointCount(int beginIndex,
                          int endIndex)
```

Returns the number of Unicode code points in the specified text range of this sequence. The text range begins at the specified `beginIndex` and extends to the `char` at index `endIndex - 1`. Thus the length (in `char`s) of the text range is `endIndex-beginIndex`. Unpaired surrogates within this sequence count as one code point each.

Parameters:

`beginIndex` - the index to the first `char` of the text range.

`endIndex` - the index after the last `char` of the text range.

Returns:

the number of Unicode code points in the specified text range

Throws:

`IndexOutOfBoundsException` - if the `beginIndex` is negative, or `endIndex` is larger than the length of this sequence, or `beginIndex` is larger than `endIndex`.

offsetByCodePoints

```
public int offsetByCodePoints(int index,
                              int codePointOffset)
```

Returns the index within this sequence that is offset from the given `index` by `codePointOffset` code points. Unpaired surrogates within the text range given by `index` and `codePointOffset` count as one code point each.

Parameters:

`index` - the index to be offset

`codePointOffset` - the offset in code points

Returns:

the index within this sequence

Throws:

`IndexOutOfBoundsException` - if `index` is negative or larger then the length of this sequence, or if `codePointOffset` is positive and the subsequence starting with `index` has fewer than `codePointOffset` code points, or if `codePointOffset` is negative and the subsequence before `index` has fewer than the absolute value of `codePointOffset` code points.

getChars

```
public void getChars(int srcBegin,
                     int srcEnd,
                     char[] dst,
                     int dstBegin)
```

Characters are copied from this sequence into the destination character array `dst`. The first character to be copied is at index `srcBegin`; the last character to be copied is at index `srcEnd-1`. The total number of characters to be copied is `srcEnd-srcBegin`. The characters are copied into the subarray of `dst` starting at index `dstBegin` and ending at index:

```
 dstbegin + (srcEnd-srcBegin) - 1
```

Parameters:

`srcBegin` - start copying at this offset.

`srcEnd` - stop copying at this offset.

`dst` - the array to copy the data into.

`dstBegin` - offset into `dst`.

Throws:

`IndexOutOfBoundsException` - if any of the following is true:

- `srcBegin` is negative
- `dstBegin` is negative
- the `srcBegin` argument is greater than the `srcEnd` argument.
- `srcEnd` is greater than `this.length()`.

- `dstBegin+srcEnd-srcBegin` is greater than `dst.length`

setCharAt

```
public void setCharAt(int index,
                      char ch)
```

The character at the specified index is set to `ch`. This sequence is altered to represent a new character sequence that is identical to the old character sequence, except that it contains the character `ch` at position `index`.

The index argument must be greater than or equal to `0`, and less than the length of this sequence.

Parameters:

`index` - the index of the character to modify.

`ch` - the new character.

Throws:

`IndexOutOfBoundsException` - if `index` is negative or greater than or equal to `length()`.

substring

```
public String substring(int start)
```

Returns a new `String` that contains a subsequence of characters currently contained in this character sequence. The substring begins at the specified index and extends to the end of this sequence.

Parameters:

`start` - The beginning index, inclusive.

Returns:

The new string.

Throws:

`StringIndexOutOfBoundsException` - if `start` is less than zero, or greater than the length of this object.

subSequence

```
public CharSequence subSequence(int start,
                                int end)
```

Returns a new character sequence that is a subsequence of this sequence.

An invocation of this method of the form

```
 sb.subSequence(begin, end)
```

behaves in exactly the same way as the invocation

```
 sb.substring(begin, end)
```

This method is provided so that this class can implement the `CharSequence` interface.

Specified by:

`subSequence` in interface `CharSequence`

Parameters:

`start` - the start index, inclusive.

`end` - the end index, exclusive.

Returns:

the specified subsequence.

Throws:

`IndexOutOfBoundsException` - if `start` or `end` are negative, if `end` is greater than `length()`, or if `start` is greater than `end`

substring

```
public String substring(int start,
                        int end)
```

Returns a new `String` that contains a subsequence of characters currently contained in this sequence. The substring begins at the specified `start` and extends to the character at index `end - 1`.

Parameters:

`start` - The beginning index, inclusive.

`end` - The ending index, exclusive.

Returns:

The new string.

Throws:

`StringIndexOutOfBoundsException` - if `start` or `end` are negative or greater than `length()`, or `start` is greater than `end`.

chars

```
public IntStream chars()
```

Returns a stream of `int` zero-extending the `char` values from this sequence. Any char which maps to a surrogate code point is passed through uninterpreted.

The stream binds to this sequence when the terminal stream operation commences (specifically, for mutable sequences the spliterator for the stream is *late-binding*). If the sequence is modified during that operation then the result is undefined.

Specified by:

`chars` in interface `CharSequence`

Returns:

an IntStream of char values from this sequence

Since:

9

codePoints

```
public IntStream codePoints()
```

Returns a stream of code point values from this sequence. Any surrogate pairs encountered in the sequence are combined as if by Character.toCodePoint and the result is passed to the stream. Any other code units, including ordinary BMP characters, unpaired surrogates, and undefined code units, are zero-extended to `int` values which are then passed to the stream.

The stream binds to this sequence when the terminal stream operation commences (specifically, for mutable sequences the spliterator for the stream is *late-binding*). If the sequence is modified during that operation then the result is undefined.

Specified by:

`codePoints` in interface `CharSequence`

Returns:

an IntStream of Unicode code points from this sequence

Since:

9