| | [**Overview**](http://docs.google.com/overview-summary.html) | [**Package**](http://docs.google.com/package-summary.html) | **Class** | [**Use**](http://docs.google.com/class-use/DoubleBuffer.html) | [**Tree**](http://docs.google.com/package-tree.html) | [**Deprecated**](http://docs.google.com/deprecated-list.html) | [**Index**](http://docs.google.com/index-files/index-1.html) | [**Help**](http://docs.google.com/help-doc.html) | | --- | --- | --- | --- | --- | --- | --- | --- | | | ***Java™ Platform***  ***Standard Ed. 6*** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| [**PREV CLASS**](http://docs.google.com/java/nio/CharBuffer.html)   [**NEXT CLASS**](http://docs.google.com/java/nio/FloatBuffer.html) | [**FRAMES**](http://docs.google.com/index.html?java/nio/DoubleBuffer.html)    [**NO FRAMES**](http://docs.google.com/DoubleBuffer.html)     [**All Classes**](http://docs.google.com/allclasses-noframe.html) |
| SUMMARY: NESTED | FIELD | CONSTR | [METHOD](#3znysh7) | DETAIL: FIELD | CONSTR | [METHOD](#3dy6vkm) |

## **java.nio**

Class DoubleBuffer

[java.lang.Object](http://docs.google.com/java/lang/Object.html)  
 [java.nio.Buffer](http://docs.google.com/java/nio/Buffer.html)  
 **java.nio.DoubleBuffer**

**All Implemented Interfaces:** [Comparable](http://docs.google.com/java/lang/Comparable.html)<[DoubleBuffer](http://docs.google.com/java/nio/DoubleBuffer.html)>

public abstract class **DoubleBuffer**extends [Buffer](http://docs.google.com/java/nio/Buffer.html)implements [Comparable](http://docs.google.com/java/lang/Comparable.html)<[DoubleBuffer](http://docs.google.com/java/nio/DoubleBuffer.html)>

A double buffer.

This class defines four categories of operations upon double buffers:

* Absolute and relative [*get*](http://docs.google.com/java/nio/DoubleBuffer.html#get()) and [*put*](http://docs.google.com/java/nio/DoubleBuffer.html#put(double)) methods that read and write single doubles;
* Relative [*bulk get*](http://docs.google.com/java/nio/DoubleBuffer.html#get(double%5B%5D)) methods that transfer contiguous sequences of doubles from this buffer into an array; and
* Relative [*bulk put*](http://docs.google.com/java/nio/DoubleBuffer.html#put(double%5B%5D)) methods that transfer contiguous sequences of doubles from a double array or some other double buffer into this buffer; and
* Methods for [compacting](http://docs.google.com/java/nio/DoubleBuffer.html#compact()), [duplicating](http://docs.google.com/java/nio/DoubleBuffer.html#duplicate()), and [slicing](http://docs.google.com/java/nio/DoubleBuffer.html#slice()) a double buffer.

Double buffers can be created either by [*allocation*](http://docs.google.com/java/nio/DoubleBuffer.html#allocate(int)), which allocates space for the buffer's content, by [*wrapping*](http://docs.google.com/java/nio/DoubleBuffer.html#wrap(double%5B%5D)) an existing double array into a buffer, or by creating a [*view*](http://docs.google.com/ByteBuffer.html#views) of an existing byte buffer.

Like a byte buffer, a double buffer is either [*direct* or *non-direct*](http://docs.google.com/ByteBuffer.html#direct). A double buffer created via the wrap methods of this class will be non-direct. A double buffer created as a view of a byte buffer will be direct if, and only if, the byte buffer itself is direct. Whether or not a double buffer is direct may be determined by invoking the [isDirect](http://docs.google.com/java/nio/DoubleBuffer.html#isDirect()) method.

Methods in this class that do not otherwise have a value to return are specified to return the buffer upon which they are invoked. This allows method invocations to be chained.

**Since:** 1.4

| **Method Summary** | |
| --- | --- |
| static [DoubleBuffer](http://docs.google.com/java/nio/DoubleBuffer.html) | [**allocate**](http://docs.google.com/java/nio/DoubleBuffer.html#allocate(int))(int capacity)            Allocates a new double buffer. |
| double[] | [**array**](http://docs.google.com/java/nio/DoubleBuffer.html#array())()            Returns the double array that backs this buffer  *(optional operation)*. |
| int | [**arrayOffset**](http://docs.google.com/java/nio/DoubleBuffer.html#arrayOffset())()            Returns the offset within this buffer's backing array of the first element of the buffer  *(optional operation)*. |
| abstract  [DoubleBuffer](http://docs.google.com/java/nio/DoubleBuffer.html) | [**asReadOnlyBuffer**](http://docs.google.com/java/nio/DoubleBuffer.html#asReadOnlyBuffer())()            Creates a new, read-only double buffer that shares this buffer's content. |
| abstract  [DoubleBuffer](http://docs.google.com/java/nio/DoubleBuffer.html) | [**compact**](http://docs.google.com/java/nio/DoubleBuffer.html#compact())()            Compacts this buffer  *(optional operation)*. |
| int | [**compareTo**](http://docs.google.com/java/nio/DoubleBuffer.html#compareTo(java.nio.DoubleBuffer))([DoubleBuffer](http://docs.google.com/java/nio/DoubleBuffer.html) that)            Compares this buffer to another. |
| abstract  [DoubleBuffer](http://docs.google.com/java/nio/DoubleBuffer.html) | [**duplicate**](http://docs.google.com/java/nio/DoubleBuffer.html#duplicate())()            Creates a new double buffer that shares this buffer's content. |
| boolean | [**equals**](http://docs.google.com/java/nio/DoubleBuffer.html#equals(java.lang.Object))([Object](http://docs.google.com/java/lang/Object.html) ob)            Tells whether or not this buffer is equal to another object. |
| abstract  double | [**get**](http://docs.google.com/java/nio/DoubleBuffer.html#get())()            Relative *get* method. |
| [DoubleBuffer](http://docs.google.com/java/nio/DoubleBuffer.html) | [**get**](http://docs.google.com/java/nio/DoubleBuffer.html#get(double%5B%5D))(double[] dst)            Relative bulk *get* method. |
| [DoubleBuffer](http://docs.google.com/java/nio/DoubleBuffer.html) | [**get**](http://docs.google.com/java/nio/DoubleBuffer.html#get(double%5B%5D,%20int,%20int))(double[] dst, int offset, int length)            Relative bulk *get* method. |
| abstract  double | [**get**](http://docs.google.com/java/nio/DoubleBuffer.html#get(int))(int index)            Absolute *get* method. |
| boolean | [**hasArray**](http://docs.google.com/java/nio/DoubleBuffer.html#hasArray())()            Tells whether or not this buffer is backed by an accessible double array. |
| int | [**hashCode**](http://docs.google.com/java/nio/DoubleBuffer.html#hashCode())()            Returns the current hash code of this buffer. |
| abstract  boolean | [**isDirect**](http://docs.google.com/java/nio/DoubleBuffer.html#isDirect())()            Tells whether or not this double buffer is direct. |
| abstract  [ByteOrder](http://docs.google.com/java/nio/ByteOrder.html) | [**order**](http://docs.google.com/java/nio/DoubleBuffer.html#order())()            Retrieves this buffer's byte order. |
| abstract  [DoubleBuffer](http://docs.google.com/java/nio/DoubleBuffer.html) | [**put**](http://docs.google.com/java/nio/DoubleBuffer.html#put(double))(double d)            Relative *put* method  *(optional operation)*. |
| [DoubleBuffer](http://docs.google.com/java/nio/DoubleBuffer.html) | [**put**](http://docs.google.com/java/nio/DoubleBuffer.html#put(double%5B%5D))(double[] src)            Relative bulk *put* method  *(optional operation)*. |
| [DoubleBuffer](http://docs.google.com/java/nio/DoubleBuffer.html) | [**put**](http://docs.google.com/java/nio/DoubleBuffer.html#put(double%5B%5D,%20int,%20int))(double[] src, int offset, int length)            Relative bulk *put* method  *(optional operation)*. |
| [DoubleBuffer](http://docs.google.com/java/nio/DoubleBuffer.html) | [**put**](http://docs.google.com/java/nio/DoubleBuffer.html#put(java.nio.DoubleBuffer))([DoubleBuffer](http://docs.google.com/java/nio/DoubleBuffer.html) src)            Relative bulk *put* method  *(optional operation)*. |
| abstract  [DoubleBuffer](http://docs.google.com/java/nio/DoubleBuffer.html) | [**put**](http://docs.google.com/java/nio/DoubleBuffer.html#put(int,%20double))(int index, double d)            Absolute *put* method  *(optional operation)*. |
| abstract  [DoubleBuffer](http://docs.google.com/java/nio/DoubleBuffer.html) | [**slice**](http://docs.google.com/java/nio/DoubleBuffer.html#slice())()            Creates a new double buffer whose content is a shared subsequence of this buffer's content. |
| [String](http://docs.google.com/java/lang/String.html) | [**toString**](http://docs.google.com/java/nio/DoubleBuffer.html#toString())()            Returns a string summarizing the state of this buffer. |
| static [DoubleBuffer](http://docs.google.com/java/nio/DoubleBuffer.html) | [**wrap**](http://docs.google.com/java/nio/DoubleBuffer.html#wrap(double%5B%5D))(double[] array)            Wraps a double array into a buffer. |
| static [DoubleBuffer](http://docs.google.com/java/nio/DoubleBuffer.html) | [**wrap**](http://docs.google.com/java/nio/DoubleBuffer.html#wrap(double%5B%5D,%20int,%20int))(double[] array, int offset, int length)            Wraps a double array into a buffer. |

| **Methods inherited from class java.nio.**[**Buffer**](http://docs.google.com/java/nio/Buffer.html) |
| --- |
| [capacity](http://docs.google.com/java/nio/Buffer.html#capacity()), [clear](http://docs.google.com/java/nio/Buffer.html#clear()), [flip](http://docs.google.com/java/nio/Buffer.html#flip()), [hasRemaining](http://docs.google.com/java/nio/Buffer.html#hasRemaining()), [isReadOnly](http://docs.google.com/java/nio/Buffer.html#isReadOnly()), [limit](http://docs.google.com/java/nio/Buffer.html#limit()), [limit](http://docs.google.com/java/nio/Buffer.html#limit(int)), [mark](http://docs.google.com/java/nio/Buffer.html#mark()), [position](http://docs.google.com/java/nio/Buffer.html#position()), [position](http://docs.google.com/java/nio/Buffer.html#position(int)), [remaining](http://docs.google.com/java/nio/Buffer.html#remaining()), [reset](http://docs.google.com/java/nio/Buffer.html#reset()), [rewind](http://docs.google.com/java/nio/Buffer.html#rewind()) |

| **Methods inherited from class java.lang.**[**Object**](http://docs.google.com/java/lang/Object.html) |
| --- |
| [clone](http://docs.google.com/java/lang/Object.html#clone()), [finalize](http://docs.google.com/java/lang/Object.html#finalize()), [getClass](http://docs.google.com/java/lang/Object.html#getClass()), [notify](http://docs.google.com/java/lang/Object.html#notify()), [notifyAll](http://docs.google.com/java/lang/Object.html#notifyAll()), [wait](http://docs.google.com/java/lang/Object.html#wait()), [wait](http://docs.google.com/java/lang/Object.html#wait(long)), [wait](http://docs.google.com/java/lang/Object.html#wait(long,%20int)) |

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

### allocate

public static [DoubleBuffer](http://docs.google.com/java/nio/DoubleBuffer.html) **allocate**(int capacity)

Allocates a new double buffer.

The new buffer's position will be zero, its limit will be its capacity, and its mark will be undefined. It will have a [backing array](http://docs.google.com/java/nio/DoubleBuffer.html#array()), and its [array offset](http://docs.google.com/java/nio/DoubleBuffer.html#arrayOffset()) will be zero.

**Parameters:**capacity - The new buffer's capacity, in doubles **Returns:**The new double buffer **Throws:** [IllegalArgumentException](http://docs.google.com/java/lang/IllegalArgumentException.html) - If the capacity is a negative integer

### wrap

public static [DoubleBuffer](http://docs.google.com/java/nio/DoubleBuffer.html) **wrap**(double[] array,  
 int offset,  
 int length)

Wraps a double array into a buffer.

The new buffer will be backed by the given double array; that is, modifications to the buffer will cause the array to be modified and vice versa. The new buffer's capacity will be array.length, its position will be offset, its limit will be offset + length, and its mark will be undefined. Its [backing array](http://docs.google.com/java/nio/DoubleBuffer.html#array()) will be the given array, and its [array offset](http://docs.google.com/java/nio/DoubleBuffer.html#arrayOffset()) will be zero.

**Parameters:**array - The array that will back the new bufferoffset - The offset of the subarray to be used; must be non-negative and no larger than array.length. The new buffer's position will be set to this value.length - The length of the subarray to be used; must be non-negative and no larger than array.length - offset. The new buffer's limit will be set to offset + length. **Returns:**The new double buffer **Throws:** [IndexOutOfBoundsException](http://docs.google.com/java/lang/IndexOutOfBoundsException.html) - If the preconditions on the offset and length parameters do not hold

### wrap

public static [DoubleBuffer](http://docs.google.com/java/nio/DoubleBuffer.html) **wrap**(double[] array)

Wraps a double array into a buffer.

The new buffer will be backed by the given double array; that is, modifications to the buffer will cause the array to be modified and vice versa. The new buffer's capacity and limit will be array.length, its position will be zero, and its mark will be undefined. Its [backing array](http://docs.google.com/java/nio/DoubleBuffer.html#array()) will be the given array, and its [array offset](http://docs.google.com/java/nio/DoubleBuffer.html#arrayOffset()) will be zero.

**Parameters:**array - The array that will back this buffer **Returns:**The new double buffer

### slice

public abstract [DoubleBuffer](http://docs.google.com/java/nio/DoubleBuffer.html) **slice**()

Creates a new double buffer whose content is a shared subsequence of this buffer's content.

The content of the new buffer will start at this buffer's current position. Changes to this buffer's content will be visible in the new buffer, and vice versa; the two buffers' position, limit, and mark values will be independent.

The new buffer's position will be zero, its capacity and its limit will be the number of doubles remaining in this buffer, and its mark will be undefined. The new buffer will be direct if, and only if, this buffer is direct, and it will be read-only if, and only if, this buffer is read-only.

**Returns:**The new double buffer

### duplicate

public abstract [DoubleBuffer](http://docs.google.com/java/nio/DoubleBuffer.html) **duplicate**()

Creates a new double buffer that shares this buffer's content.

The content of the new buffer will be that of this buffer. Changes to this buffer's content will be visible in the new buffer, and vice versa; the two buffers' position, limit, and mark values will be independent.

The new buffer's capacity, limit, position, and mark values will be identical to those of this buffer. The new buffer will be direct if, and only if, this buffer is direct, and it will be read-only if, and only if, this buffer is read-only.

**Returns:**The new double buffer

### asReadOnlyBuffer

public abstract [DoubleBuffer](http://docs.google.com/java/nio/DoubleBuffer.html) **asReadOnlyBuffer**()

Creates a new, read-only double buffer that shares this buffer's content.

The content of the new buffer will be that of this buffer. Changes to this buffer's content will be visible in the new buffer; the new buffer itself, however, will be read-only and will not allow the shared content to be modified. The two buffers' position, limit, and mark values will be independent.

The new buffer's capacity, limit, position, and mark values will be identical to those of this buffer.

If this buffer is itself read-only then this method behaves in exactly the same way as the [duplicate](http://docs.google.com/java/nio/DoubleBuffer.html#duplicate()) method.

**Returns:**The new, read-only double buffer

### get

public abstract double **get**()

Relative *get* method. Reads the double at this buffer's current position, and then increments the position.

**Returns:**The double at the buffer's current position **Throws:** [BufferUnderflowException](http://docs.google.com/java/nio/BufferUnderflowException.html) - If the buffer's current position is not smaller than its limit

### put

public abstract [DoubleBuffer](http://docs.google.com/java/nio/DoubleBuffer.html) **put**(double d)

Relative *put* method  *(optional operation)*.

Writes the given double into this buffer at the current position, and then increments the position.

**Parameters:**d - The double to be written **Returns:**This buffer **Throws:** [BufferOverflowException](http://docs.google.com/java/nio/BufferOverflowException.html) - If this buffer's current position is not smaller than its limit [ReadOnlyBufferException](http://docs.google.com/java/nio/ReadOnlyBufferException.html) - If this buffer is read-only

### get

public abstract double **get**(int index)

Absolute *get* method. Reads the double at the given index.

**Parameters:**index - The index from which the double will be read **Returns:**The double at the given index **Throws:** [IndexOutOfBoundsException](http://docs.google.com/java/lang/IndexOutOfBoundsException.html) - If index is negative or not smaller than the buffer's limit

### put

public abstract [DoubleBuffer](http://docs.google.com/java/nio/DoubleBuffer.html) **put**(int index,  
 double d)

Absolute *put* method  *(optional operation)*.

Writes the given double into this buffer at the given index.

**Parameters:**index - The index at which the double will be writtend - The double value to be written **Returns:**This buffer **Throws:** [IndexOutOfBoundsException](http://docs.google.com/java/lang/IndexOutOfBoundsException.html) - If index is negative or not smaller than the buffer's limit [ReadOnlyBufferException](http://docs.google.com/java/nio/ReadOnlyBufferException.html) - If this buffer is read-only

### get

public [DoubleBuffer](http://docs.google.com/java/nio/DoubleBuffer.html) **get**(double[] dst,  
 int offset,  
 int length)

Relative bulk *get* method.

This method transfers doubles from this buffer into the given destination array. If there are fewer doubles remaining in the buffer than are required to satisfy the request, that is, if length > remaining(), then no doubles are transferred and a [BufferUnderflowException](http://docs.google.com/java/nio/BufferUnderflowException.html) is thrown.

Otherwise, this method copies length doubles from this buffer into the given array, starting at the current position of this buffer and at the given offset in the array. The position of this buffer is then incremented by length.

In other words, an invocation of this method of the form src.get(dst, off, len) has exactly the same effect as the loop

for (int i = off; i < off + len; i++)  
 dst[i] = src.get();

except that it first checks that there are sufficient doubles in this buffer and it is potentially much more efficient.

**Parameters:**dst - The array into which doubles are to be writtenoffset - The offset within the array of the first double to be written; must be non-negative and no larger than dst.lengthlength - The maximum number of doubles to be written to the given array; must be non-negative and no larger than dst.length - offset **Returns:**This buffer **Throws:** [BufferUnderflowException](http://docs.google.com/java/nio/BufferUnderflowException.html) - If there are fewer than length doubles remaining in this buffer [IndexOutOfBoundsException](http://docs.google.com/java/lang/IndexOutOfBoundsException.html) - If the preconditions on the offset and length parameters do not hold

### get

public [DoubleBuffer](http://docs.google.com/java/nio/DoubleBuffer.html) **get**(double[] dst)

Relative bulk *get* method.

This method transfers doubles from this buffer into the given destination array. An invocation of this method of the form src.get(a) behaves in exactly the same way as the invocation

src.get(a, 0, a.length)

**Returns:**This buffer **Throws:** [BufferUnderflowException](http://docs.google.com/java/nio/BufferUnderflowException.html) - If there are fewer than length doubles remaining in this buffer

### put

public [DoubleBuffer](http://docs.google.com/java/nio/DoubleBuffer.html) **put**([DoubleBuffer](http://docs.google.com/java/nio/DoubleBuffer.html) src)

Relative bulk *put* method  *(optional operation)*.

This method transfers the doubles remaining in the given source buffer into this buffer. If there are more doubles remaining in the source buffer than in this buffer, that is, if src.remaining() > remaining(), then no doubles are transferred and a [BufferOverflowException](http://docs.google.com/java/nio/BufferOverflowException.html) is thrown.

Otherwise, this method copies *n* = src.remaining() doubles from the given buffer into this buffer, starting at each buffer's current position. The positions of both buffers are then incremented by *n*.

In other words, an invocation of this method of the form dst.put(src) has exactly the same effect as the loop

while (src.hasRemaining())  
 dst.put(src.get());

except that it first checks that there is sufficient space in this buffer and it is potentially much more efficient.

**Parameters:**src - The source buffer from which doubles are to be read; must not be this buffer **Returns:**This buffer **Throws:** [BufferOverflowException](http://docs.google.com/java/nio/BufferOverflowException.html) - If there is insufficient space in this buffer for the remaining doubles in the source buffer [IllegalArgumentException](http://docs.google.com/java/lang/IllegalArgumentException.html) - If the source buffer is this buffer [ReadOnlyBufferException](http://docs.google.com/java/nio/ReadOnlyBufferException.html) - If this buffer is read-only

### put

public [DoubleBuffer](http://docs.google.com/java/nio/DoubleBuffer.html) **put**(double[] src,  
 int offset,  
 int length)

Relative bulk *put* method  *(optional operation)*.

This method transfers doubles into this buffer from the given source array. If there are more doubles to be copied from the array than remain in this buffer, that is, if length > remaining(), then no doubles are transferred and a [BufferOverflowException](http://docs.google.com/java/nio/BufferOverflowException.html) is thrown.

Otherwise, this method copies length doubles from the given array into this buffer, starting at the given offset in the array and at the current position of this buffer. The position of this buffer is then incremented by length.

In other words, an invocation of this method of the form dst.put(src, off, len) has exactly the same effect as the loop

for (int i = off; i < off + len; i++)  
 dst.put(a[i]);

except that it first checks that there is sufficient space in this buffer and it is potentially much more efficient.

**Parameters:**src - The array from which doubles are to be readoffset - The offset within the array of the first double to be read; must be non-negative and no larger than array.lengthlength - The number of doubles to be read from the given array; must be non-negative and no larger than array.length - offset **Returns:**This buffer **Throws:** [BufferOverflowException](http://docs.google.com/java/nio/BufferOverflowException.html) - If there is insufficient space in this buffer [IndexOutOfBoundsException](http://docs.google.com/java/lang/IndexOutOfBoundsException.html) - If the preconditions on the offset and length parameters do not hold [ReadOnlyBufferException](http://docs.google.com/java/nio/ReadOnlyBufferException.html) - If this buffer is read-only

### put

public final [DoubleBuffer](http://docs.google.com/java/nio/DoubleBuffer.html) **put**(double[] src)

Relative bulk *put* method  *(optional operation)*.

This method transfers the entire content of the given source double array into this buffer. An invocation of this method of the form dst.put(a) behaves in exactly the same way as the invocation

dst.put(a, 0, a.length)

**Returns:**This buffer **Throws:** [BufferOverflowException](http://docs.google.com/java/nio/BufferOverflowException.html) - If there is insufficient space in this buffer [ReadOnlyBufferException](http://docs.google.com/java/nio/ReadOnlyBufferException.html) - If this buffer is read-only

### hasArray

public final boolean **hasArray**()

Tells whether or not this buffer is backed by an accessible double array.

If this method returns true then the [array](http://docs.google.com/java/nio/DoubleBuffer.html#array()) and [arrayOffset](http://docs.google.com/java/nio/DoubleBuffer.html#arrayOffset()) methods may safely be invoked.

**Specified by:**[hasArray](http://docs.google.com/java/nio/Buffer.html#hasArray()) in class [Buffer](http://docs.google.com/java/nio/Buffer.html) **Returns:**true if, and only if, this buffer is backed by an array and is not read-only

### array

public final double[] **array**()

Returns the double array that backs this buffer  *(optional operation)*.

Modifications to this buffer's content will cause the returned array's content to be modified, and vice versa.

Invoke the [hasArray](http://docs.google.com/java/nio/DoubleBuffer.html#hasArray()) method before invoking this method in order to ensure that this buffer has an accessible backing array.

**Specified by:**[array](http://docs.google.com/java/nio/Buffer.html#array()) in class [Buffer](http://docs.google.com/java/nio/Buffer.html) **Returns:**The array that backs this buffer **Throws:** [ReadOnlyBufferException](http://docs.google.com/java/nio/ReadOnlyBufferException.html) - If this buffer is backed by an array but is read-only [UnsupportedOperationException](http://docs.google.com/java/lang/UnsupportedOperationException.html) - If this buffer is not backed by an accessible array

### arrayOffset

public final int **arrayOffset**()

Returns the offset within this buffer's backing array of the first element of the buffer  *(optional operation)*.

If this buffer is backed by an array then buffer position *p* corresponds to array index *p* + arrayOffset().

Invoke the [hasArray](http://docs.google.com/java/nio/DoubleBuffer.html#hasArray()) method before invoking this method in order to ensure that this buffer has an accessible backing array.

**Specified by:**[arrayOffset](http://docs.google.com/java/nio/Buffer.html#arrayOffset()) in class [Buffer](http://docs.google.com/java/nio/Buffer.html) **Returns:**The offset within this buffer's array of the first element of the buffer **Throws:** [ReadOnlyBufferException](http://docs.google.com/java/nio/ReadOnlyBufferException.html) - If this buffer is backed by an array but is read-only [UnsupportedOperationException](http://docs.google.com/java/lang/UnsupportedOperationException.html) - If this buffer is not backed by an accessible array

### compact

public abstract [DoubleBuffer](http://docs.google.com/java/nio/DoubleBuffer.html) **compact**()

Compacts this buffer  *(optional operation)*.

The doubles between the buffer's current position and its limit, if any, are copied to the beginning of the buffer. That is, the double at index *p* = position() is copied to index zero, the double at index *p* + 1 is copied to index one, and so forth until the double at index limit() - 1 is copied to index *n* = limit() - 1 - *p*. The buffer's position is then set to *n+1* and its limit is set to its capacity. The mark, if defined, is discarded.

The buffer's position is set to the number of doubles copied, rather than to zero, so that an invocation of this method can be followed immediately by an invocation of another relative *put* method.

**Returns:**This buffer **Throws:** [ReadOnlyBufferException](http://docs.google.com/java/nio/ReadOnlyBufferException.html) - If this buffer is read-only

### isDirect

public abstract boolean **isDirect**()

Tells whether or not this double buffer is direct.

**Specified by:**[isDirect](http://docs.google.com/java/nio/Buffer.html#isDirect()) in class [Buffer](http://docs.google.com/java/nio/Buffer.html) **Returns:**true if, and only if, this buffer is direct

### toString

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

Returns a string summarizing the state of this buffer.

**Overrides:**[toString](http://docs.google.com/java/lang/Object.html#toString()) in class [Object](http://docs.google.com/java/lang/Object.html) **Returns:**A summary string

### hashCode

public int **hashCode**()

Returns the current hash code of this buffer.

The hash code of a double buffer depends only upon its remaining elements; that is, upon the elements from position() up to, and including, the element at limit() - 1.

Because buffer hash codes are content-dependent, it is inadvisable to use buffers as keys in hash maps or similar data structures unless it is known that their contents will not change.

**Overrides:**[hashCode](http://docs.google.com/java/lang/Object.html#hashCode()) in class [Object](http://docs.google.com/java/lang/Object.html) **Returns:**The current hash code of this buffer**See Also:**[Object.equals(java.lang.Object)](http://docs.google.com/java/lang/Object.html#equals(java.lang.Object)), [Hashtable](http://docs.google.com/java/util/Hashtable.html)

### equals

public boolean **equals**([Object](http://docs.google.com/java/lang/Object.html) ob)

Tells whether or not this buffer is equal to another object.

Two double buffers are equal if, and only if,

1. They have the same element type,
2. They have the same number of remaining elements, and
3. The two sequences of remaining elements, considered independently of their starting positions, are pointwise equal.

A double buffer is not equal to any other type of object.

**Overrides:**[equals](http://docs.google.com/java/lang/Object.html#equals(java.lang.Object)) in class [Object](http://docs.google.com/java/lang/Object.html) **Parameters:**ob - The object to which this buffer is to be compared **Returns:**true if, and only if, this buffer is equal to the given object**See Also:**[Object.hashCode()](http://docs.google.com/java/lang/Object.html#hashCode()), [Hashtable](http://docs.google.com/java/util/Hashtable.html)

### compareTo

public int **compareTo**([DoubleBuffer](http://docs.google.com/java/nio/DoubleBuffer.html) that)

Compares this buffer to another.

Two double buffers are compared by comparing their sequences of remaining elements lexicographically, without regard to the starting position of each sequence within its corresponding buffer.

A double buffer is not comparable to any other type of object.

**Specified by:**[compareTo](http://docs.google.com/java/lang/Comparable.html#compareTo(T)) in interface [Comparable](http://docs.google.com/java/lang/Comparable.html)<[DoubleBuffer](http://docs.google.com/java/nio/DoubleBuffer.html)> **Parameters:**that - the object to be compared. **Returns:**A negative integer, zero, or a positive integer as this buffer is less than, equal to, or greater than the given buffer

### order

public abstract [ByteOrder](http://docs.google.com/java/nio/ByteOrder.html) **order**()

Retrieves this buffer's byte order.

The byte order of a double buffer created by allocation or by wrapping an existing double array is the [native order](http://docs.google.com/java/nio/ByteOrder.html#nativeOrder()) of the underlying hardware. The byte order of a double buffer created as a [view](http://docs.google.com/ByteBuffer.html#views) of a byte buffer is that of the byte buffer at the moment that the view is created.

**Returns:**This buffer's byte order

| | [**Overview**](http://docs.google.com/overview-summary.html) | [**Package**](http://docs.google.com/package-summary.html) | **Class** | [**Use**](http://docs.google.com/class-use/DoubleBuffer.html) | [**Tree**](http://docs.google.com/package-tree.html) | [**Deprecated**](http://docs.google.com/deprecated-list.html) | [**Index**](http://docs.google.com/index-files/index-1.html) | [**Help**](http://docs.google.com/help-doc.html) | | --- | --- | --- | --- | --- | --- | --- | --- | | | ***Java™ Platform***  ***Standard Ed. 6*** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| [**PREV CLASS**](http://docs.google.com/java/nio/CharBuffer.html)   [**NEXT CLASS**](http://docs.google.com/java/nio/FloatBuffer.html) | [**FRAMES**](http://docs.google.com/index.html?java/nio/DoubleBuffer.html)    [**NO FRAMES**](http://docs.google.com/DoubleBuffer.html)     [**All Classes**](http://docs.google.com/allclasses-noframe.html) |
| SUMMARY: NESTED | FIELD | CONSTR | [METHOD](#3znysh7) | DETAIL: FIELD | CONSTR | [METHOD](#3dy6vkm) |

[Submit a bug or feature](http://bugs.sun.com/services/bugreport/index.jsp)

For further API reference and developer documentation, see [Java SE Developer Documentation](http://docs.google.com/webnotes/devdocs-vs-specs.html). That documentation contains more detailed, developer-targeted descriptions, with conceptual overviews, definitions of terms, workarounds, and working code examples.

Copyright 2006 Sun Microsystems, Inc. All rights reserved. Use is subject to [license terms](http://docs.google.com/legal/license.html). Also see the [documentation redistribution policy](http://java.sun.com/docs/redist.html).