Module jdk.incubator.foreign
Package jdk.incubator.foreign

Class MemoryAccess

java.lang.Object
    jdk.incubator.foreign.MemoryAccess

```
public final class MemoryAccess
extends Object
```

This class defines ready-made static accessors which can be used to dereference memory segments in many ways.

The most primitive accessors (see `getIntAtOffset(MemorySegment, long, ByteOrder)`) take a segment, an offset (expressed in bytes) and a byte order. The final address at which the dereference will occur will be computed by offsetting the base address by the specified offset, as if by calling `MemoryAddress.addOffset(long)` on the specified base address.

In cases where no offset is required, overloads are provided (see `getInt(MemorySegment, ByteOrder)`) so that clients can omit the offset coordinate.

To help dereferencing in array-like use cases (e.g. where the layout of a given memory segment is a sequence layout of given size an element count), higher-level overloads are also provided (see `getIntAtIndex(MemorySegment, long, ByteOrder)`), which take a segment and a *logical* element index. The formula to obtain the byte offset `O` from an index `I` is given by `O = I * S` where `S` is the size (expressed in bytes) of the element to be dereferenced.

In cases where native byte order is preferred, overloads are provided (see `getIntAtOffset(MemorySegment, long)`) so that clients can omit the byte order parameter.

Unless otherwise specified, passing a `null` argument, or an array argument containing one or more `null` elements to a method in this class causes a `NullPointerException` to be thrown.

Method Summary

All Methods | Static Methods | Concrete Methods

Modifier and Type	Method	Description
static MemoryAddress	getAddress(MemorySegment segment)	Reads a memory address from given segment, with byte order set to ByteOrder.nativeOrder().
static MemoryAddress	getAddressAtIndex(MemorySegment segment, long index)	Reads a memory address from given segment and element index, with byte order set to ByteOrder.nativeOrder().
static MemoryAddress	getAddressAtOffset(MemorySegment segment, long offset)	Reads a memory address from given segment and offset, with byte order set to ByteOrder.nativeOrder().
static byte	getByte(MemorySegment segment)	Reads a byte from given segment.
static byte	getByteAtOffset(MemorySegment segment, long offset)	Reads a byte from given segment and offset.
static char	getChar(MemorySegment segment)	Reads a char from given segment, with byte order set to ByteOrder.nativeOrder().
static char	getChar(MemorySegment segment, ByteOrder order)	Reads a char from given segment, with given byte order.
static char	getCharAtIndex(MemorySegment segment, long index)	Reads a char from given segment and element index, with byte order set to ByteOrder.nativeOrder().
static char	getCharAtIndex(MemorySegment segment, long index, ByteOrder order)	Reads a char from given segment and element index, with given byte order.
static char	getCharAtOffset(MemorySegment segment, long offset)	Reads a char from given segment and offset, with byte order set to ByteOrder.nativeOrder().
static char	getCharAtOffset(MemorySegment segment, long offset, ByteOrder order)	Reads a char from given segment and offset with given byte order.
static double	getDouble(MemorySegment segment)	Reads a double from given segment, with byte order set to ByteOrder.nativeOrder().
static double	getDouble(MemorySegment segment, ByteOrder order)	Reads a double from given segment, with given byte order.
static double	getDoubleAtIndex(MemorySegment segment, long index)	Reads a double from given segment and element index, with byte order set to ByteOrder.nativeOrder().
static double	getDoubleAtIndex(MemorySegment segment, long index, ByteOrder order)	Reads a double from given segment and element index, with given byte order.
static double	getDoubleAtOffset(MemorySegment segment, long offset)	Reads a double from given segment and offset, with byte order set to ByteOrder.nativeOrder().

static double	**getDoubleAtOffset**(**MemorySegment** segment, long offset, **ByteOrder** order)	Reads a double from given segment and offset with given byte order.
static float	**getFloat**(**MemorySegment** segment)	Reads a float from given segment, with byte order set to `ByteOrder.nativeOrder()`.
static float	**getFloat**(**MemorySegment** segment, **ByteOrder** order)	Reads a float from given segment, with given byte order.
static float	**getFloatAtIndex**(**MemorySegment** segment, long index)	Reads a float from given segment and element index, with byte order set to `ByteOrder.nativeOrder()`.
static float	**getFloatAtIndex**(**MemorySegment** segment, long index, **ByteOrder** order)	Reads a float from given segment and element index, with given byte order.
static float	**getFloatAtOffset**(**MemorySegment** segment, long offset)	Reads a float from given segment and offset, with byte order set to `ByteOrder.nativeOrder()`.
static float	**getFloatAtOffset**(**MemorySegment** segment, long offset, **ByteOrder** order)	Reads a float from given segment and offset with given byte order.
static int	**getInt**(**MemorySegment** segment)	Reads an int from given segment, with byte order set to `ByteOrder.nativeOrder()`.
static int	**getInt**(**MemorySegment** segment, **ByteOrder** order)	Reads an int from given segment, with given byte order.
static int	**getIntAtIndex**(**MemorySegment** segment, long index)	Reads an int from given segment and element index, with byte order set to `ByteOrder.nativeOrder()`.
static int	**getIntAtIndex**(**MemorySegment** segment, long index, **ByteOrder** order)	Reads an int from given segment and element index, with given byte order.
static int	**getIntAtOffset**(**MemorySegment** segment, long offset)	Reads an int from given segment and offset, with byte order set to `ByteOrder.nativeOrder()`.
static int	**getIntAtOffset**(**MemorySegment** segment, long offset, **ByteOrder** order)	Reads an int from given segment and offset with given byte order.
static long	**getLong**(**MemorySegment** segment)	Reads a long from given segment, with byte order set to `ByteOrder.nativeOrder()`.
static long	**getLong**(**MemorySegment** segment, **ByteOrder** order)	Reads a long from given segment, with given byte order.
static long	**getLongAtIndex**(**MemorySegment** segment, long index)	Reads a long from given segment and element index, with byte order set to `ByteOrder.nativeOrder()`.
static long	**getLongAtIndex**(**MemorySegment** segment, long index, **ByteOrder** order)	Reads a long from given segment and element index, with given byte order.
static long	**getLongAtOffset**(**MemorySegment** segment, long offset)	Reads a long from given segment and offset, with byte order set to `ByteOrder.nativeOrder()`.
static long	**getLongAtOffset**(**MemorySegment** segment, long offset, **ByteOrder** order)	Reads a long from given segment and offset with given byte order.
static short	**getShort**(**MemorySegment** segment)	Reads a short from given segment, with byte order set to `ByteOrder.nativeOrder()`.
static short	**getShort**(**MemorySegment** segment, **ByteOrder** order)	Reads a short from given segment, with given byte order.
static short	**getShortAtIndex**(**MemorySegment** segment, long index)	Reads a short from given segment and element index, with byte order set to `ByteOrder.nativeOrder()`.
static short	**getShortAtIndex**(**MemorySegment** segment, long index, **ByteOrder** order)	Reads a short from given segment and element index, with given byte order.
static short	**getShortAtOffset**(**MemorySegment** segment, long offset)	Reads a short from given segment and offset, with byte order set to `ByteOrder.nativeOrder()`.
static short	**getShortAtOffset**(**MemorySegment** segment, long offset, **ByteOrder** order)	Reads a short from given segment and offset with given byte order.
static void	**setAddress**(**MemorySegment** segment, **Addressable** value)	Writes a memory address at given segment, with byte order set to `ByteOrder.nativeOrder()`.
static void	**setAddressAtIndex**(**MemorySegment** segment, long index, **Addressable** value)	Writes a memory address at given segment and element index, with byte order set to `ByteOrder.nativeOrder()`.
static void	**setAddressAtOffset**(**MemorySegment** segment, long offset, **Addressable** value)	Writes a memory address at given segment and offset, with byte order set to `ByteOrder.nativeOrder()`.
static void	**setByte**(**MemorySegment** segment, byte value)	Writes a byte at given segment.

static void	**setByteAtOffset**(**MemorySegment** segment, long offset, byte value)	Writes a byte at given segment and offset.
static void	**setChar**(**MemorySegment** segment, char value)	Writes a char at given segment, with byte order set to `ByteOrder.nativeOrder()`.
static void	**setChar**(**MemorySegment** segment, **ByteOrder** order, char value)	Writes a char at given segment, with given byte order.
static void	**setCharAtIndex**(**MemorySegment** segment, long index, char value)	Writes a char at given segment and element index, with byte order set to `ByteOrder.nativeOrder()`.
static void	**setCharAtIndex**(**MemorySegment** segment, long index, **ByteOrder** order, char value)	Writes a char at given segment and element index, with given byte order.
static void	**setCharAtOffset**(**MemorySegment** segment, long offset, char value)	Writes a char at given segment and offset, with byte order set to `ByteOrder.nativeOrder()`.
static void	**setCharAtOffset**(**MemorySegment** segment, long offset, **ByteOrder** order, char value)	Writes a char at given segment and offset with given byte order.
static void	**setDouble**(**MemorySegment** segment, double value)	Writes a double at given segment, with byte order set to `ByteOrder.nativeOrder()`.
static void	**setDouble**(**MemorySegment** segment, **ByteOrder** order, double value)	Writes a double at given segment, with given byte order.
static void	**setDoubleAtIndex**(**MemorySegment** segment, long index, double value)	Writes a double at given segment and element index, with byte order set to `ByteOrder.nativeOrder()`.
static void	**setDoubleAtIndex**(**MemorySegment** segment, long index, **ByteOrder** order, double value)	Writes a double at given segment and element index, with given byte order.
static void	**setDoubleAtOffset**(**MemorySegment** segment, long offset, double value)	Writes a double at given segment and offset, with byte order set to `ByteOrder.nativeOrder()`.
static void	**setDoubleAtOffset**(**MemorySegment** segment, long offset, **ByteOrder** order, double value)	Writes a double at given segment and offset with given byte order.
static void	**setFloat**(**MemorySegment** segment, float value)	Writes a float at given segment, with byte order set to `ByteOrder.nativeOrder()`.
static void	**setFloat**(**MemorySegment** segment, **ByteOrder** order, float value)	Writes a float at given segment, with given byte order.
static void	**setFloatAtIndex**(**MemorySegment** segment, long index, float value)	Writes a float at given segment and element index, with byte order set to `ByteOrder.nativeOrder()`.
static void	**setFloatAtIndex**(**MemorySegment** segment, long index, **ByteOrder** order, float value)	Writes a float at given segment and element index, with given byte order.
static void	**setFloatAtOffset**(**MemorySegment** segment, long offset, float value)	Writes a float at given segment and offset, with byte order set to `ByteOrder.nativeOrder()`.
static void	**setFloatAtOffset**(**MemorySegment** segment, long offset, **ByteOrder** order, float value)	Writes a float at given segment and offset with given byte order.
static void	**setInt**(**MemorySegment** segment, int value)	Writes an int at given segment, with byte order set to `ByteOrder.nativeOrder()`.
static void	**setInt**(**MemorySegment** segment, **ByteOrder** order, int value)	Writes an int at given segment, with given byte order.
static void	**setIntAtIndex**(**MemorySegment** segment, long index, int value)	Writes an int at given segment and element index, with byte order set to `ByteOrder.nativeOrder()`.
static void	**setIntAtIndex**(**MemorySegment** segment, long index, **ByteOrder** order, int value)	Writes an int at given segment and element index, with given byte order.
static void	**setIntAtOffset**(**MemorySegment** segment, long offset, int value)	Writes an int at given segment and offset, with byte order set to `ByteOrder.nativeOrder()`.
static void	**setIntAtOffset**(**MemorySegment** segment, long offset, **ByteOrder** order, int value)	Writes an int at given segment and offset with given byte order.
static void	**setLong**(**MemorySegment** segment, long value)	Writes a long at given segment, with byte order set to `ByteOrder.nativeOrder()`.
static void	**setLong**(**MemorySegment** segment, **ByteOrder** order, long value)	Writes a long at given segment, with given byte order.
static void	**setLongAtIndex**(**MemorySegment** segment, long index, long value)	Writes a long at given segment and element index, with byte order set to `ByteOrder.nativeOrder()`.
static void	**setLongAtIndex**(**MemorySegment** segment, long index, **ByteOrder** order, long value)	Writes a long at given segment and element index, with given byte order.
static void	**setLongAtOffset**(**MemorySegment** segment, long offset, long value)	Writes a long at given segment and offset, with byte order set to `ByteOrder.nativeOrder()`.

static void	**setLongAtOffset**(**MemorySegment** segment, long offset, **ByteOrder** order, long value)	Writes a long at given segment and offset with given byte order.
static void	**setShort**(**MemorySegment** segment, short value)	Writes a short at given segment, with byte order set to `ByteOrder.nativeOrder()`.
static void	**setShort**(**MemorySegment** segment, **ByteOrder** order, short value)	Writes a short at given segment, with given byte order.
static void	**setShortAtIndex**(**MemorySegment** segment, long index, short value)	Writes a short at given segment and element index, with byte order set to `ByteOrder.nativeOrder()`.
static void	**setShortAtIndex**(**MemorySegment** segment, long index, **ByteOrder** order, short value)	Writes a short at given segment and element index, with given byte order.
static void	**setShortAtOffset**(**MemorySegment** segment, long offset, short value)	Writes a short at given segment and offset, with byte order set to `ByteOrder.nativeOrder()`.
static void	**setShortAtOffset**(**MemorySegment** segment, long offset, **ByteOrder** order, short value)	Writes a short at given segment and offset with given byte order.

Methods declared in class java.lang.Object

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

Method Details

getByteAtOffset

```
public static byte getByteAtOffset(MemorySegment segment,
                                   long offset)
```

Reads a byte from given segment and offset.

Parameters:

`segment` - the segment to be dereferenced.

`offset` - offset in bytes (relative to `segment`). The final address of this read operation can be expressed as `segment.address().addOffset(offset)`.

Returns:

a byte value read from `segment`.

setByteAtOffset

```
public static void setByteAtOffset(MemorySegment segment,
                                   long offset,
                                   byte value)
```

Writes a byte at given segment and offset.

Parameters:

`segment` - the segment to be dereferenced.

`offset` - offset in bytes (relative to `segment`). The final address of this read operation can be expressed as `segment.address().addOffset(offset)`.

`value` - the byte value to be written.

getCharAtOffset

```
public static char getCharAtOffset(MemorySegment segment,
                                   long offset)
```

Reads a char from given segment and offset, with byte order set to `ByteOrder.nativeOrder()`.

This is equivalent to the following code:

```
getCharAtOffset(segment, offset, ByteOrder.nativeOrder());
```

Parameters:

`segment` - the segment to be dereferenced.

`offset` - offset in bytes (relative to `segment`). The final address of this read operation can be expressed as `segment.address().addOffset(offset)`.

Returns:

a char value read from `segment`.

setCharAtOffset

```
public static void setCharAtOffset(MemorySegment segment,
                                   long offset,
                                   char value)
```

Writes a char at given segment and offset, with byte order set to `ByteOrder.nativeOrder()`.

This is equivalent to the following code:

```
setCharAtOffset(segment, offset, ByteOrder.nativeOrder(), value);
```

Parameters:

`segment` - the segment to be dereferenced.

`offset` - offset in bytes (relative to `segment`). The final address of this read operation can be expressed as `segment.address().addOffset(offset)`.

`value` - the char value to be written.

getShortAtOffset

```
public static short getShortAtOffset(MemorySegment segment,
                                     long offset)
```

Reads a short from given segment and offset, with byte order set to `ByteOrder.nativeOrder()`.

This is equivalent to the following code:

```
getShortAtOffset(segment, offset, ByteOrder.nativeOrder());
```

Parameters:

`segment` - the segment to be dereferenced.

`offset` - offset in bytes (relative to `segment`). The final address of this read operation can be expressed as `segment.address().addOffset(offset)`.

Returns:

a short value read from `segment`.

setShortAtOffset

```
public static void setShortAtOffset(MemorySegment segment,
                                    long offset,
                                    short value)
```

Writes a short at given segment and offset, with byte order set to `ByteOrder.nativeOrder()`.

This is equivalent to the following code:

```
setShortAtOffset(segment, offset, ByteOrder.nativeOrder(), value);
```

Parameters:

`segment` - the segment to be dereferenced.

`offset` - offset in bytes (relative to `segment`). The final address of this read operation can be expressed as `segment.address().addOffset(offset)`.

`value` - the short value to be written.

getIntAtOffset

```
public static int getIntAtOffset(MemorySegment segment,
                                 long offset)
```

Reads an int from given segment and offset, with byte order set to `ByteOrder.nativeOrder()`.

This is equivalent to the following code:

```
getIntAtOffset(segment, offset, ByteOrder.nativeOrder());
```

Parameters:

`segment` - the segment to be dereferenced.

`offset` - offset in bytes (relative to `segment`). The final address of this read operation can be expressed as `segment.address().addOffset(offset)`.

Returns:

an int value read from `segment`.

setIntAtOffset

```
public static void setIntAtOffset(MemorySegment segment,
                                  long offset,
                                  int value)
```

Writes an int at given segment and offset, with byte order set to `ByteOrder.nativeOrder()`.

This is equivalent to the following code:

```
setIntAtOffset(segment, offset, ByteOrder.nativeOrder(), value);
```

Parameters:

`segment` - the segment to be dereferenced.

`offset` - offset in bytes (relative to `segment`). The final address of this read operation can be expressed as `segment.address().addOffset(offset)`.

`value` - the int value to be written.

getFloatAtOffset

```
public static float getFloatAtOffset(MemorySegment segment,
                                     long offset)
```

Reads a float from given segment and offset, with byte order set to `ByteOrder.nativeOrder()`.

This is equivalent to the following code:

```
getFloatAtOffset(segment, offset, ByteOrder.nativeOrder());
```

Parameters:

`segment` - the segment to be dereferenced.

`offset` - offset in bytes (relative to `segment`). The final address of this read operation can be expressed as `segment.address().addOffset(offset)`.

Returns:

a float value read from `segment`.

setFloatAtOffset

```
public static void setFloatAtOffset(MemorySegment segment,
                                    long offset,
                                    float value)
```

Writes a float at given segment and offset, with byte order set to `ByteOrder.nativeOrder()`.

This is equivalent to the following code:

```
setFloatAtOffset(segment, offset, ByteOrder.nativeOrder(), value);
```

Parameters:

`segment` - the segment to be dereferenced.

`offset` - offset in bytes (relative to `segment`). The final address of this read operation can be expressed as `segment.address().addOffset(offset)`.

`value` - the float value to be written.

getLongAtOffset

```
public static long getLongAtOffset(MemorySegment segment,
                                   long offset)
```

Reads a long from given segment and offset, with byte order set to `ByteOrder.nativeOrder()`.

This is equivalent to the following code:

```
getLongAtOffset(segment, offset, ByteOrder.nativeOrder());
```

Parameters:

`segment` - the segment to be dereferenced.

`offset` - offset in bytes (relative to `segment`). The final address of this read operation can be expressed as `segment.address().addOffset(offset)`.

Returns:

a long value read from `segment`.

setLongAtOffset

```
public static void setLongAtOffset(MemorySegment segment,
                                   long offset,
                                   long value)
```

Writes a long at given segment and offset, with byte order set to `ByteOrder.nativeOrder()`.

This is equivalent to the following code:

```
setLongAtOffset(segment, offset, ByteOrder.nativeOrder(), value);
```

Parameters:

`segment` - the segment to be dereferenced.

`offset` - offset in bytes (relative to `segment`). The final address of this read operation can be expressed as `segment.address().addOffset(offset)`.

`value` - the long value to be written.

getDoubleAtOffset

```
public static double getDoubleAtOffset(MemorySegment segment,
                                       long offset)
```

Reads a double from given segment and offset, with byte order set to `ByteOrder.nativeOrder()`.

This is equivalent to the following code:

```
getDoubleAtOffset(segment, offset, ByteOrder.nativeOrder());
```

Parameters:

`segment` - the segment to be dereferenced.

`offset` - offset in bytes (relative to `segment`). The final address of this read operation can be expressed as `segment.address().addOffset(offset)`.

Returns:

a double value read from `segment`.

setDoubleAtOffset

```
public static void setDoubleAtOffset(MemorySegment segment,
                                     long offset,
                                     double value)
```

Writes a double at given segment and offset, with byte order set to `ByteOrder.nativeOrder()`.

This is equivalent to the following code:

```
setDoubleAtOffset(segment, offset, ByteOrder.nativeOrder(), value);
```

Parameters:

`segment` - the segment to be dereferenced.

`offset` - offset in bytes (relative to `segment`). The final address of this read operation can be expressed as `segment.address().addOffset(offset)`.

`value` - the double value to be written.

getAddressAtOffset

```
public static MemoryAddress getAddressAtOffset(MemorySegment segment,
                                               long offset)
```

Reads a memory address from given segment and offset, with byte order set to `ByteOrder.nativeOrder()`.

This is equivalent (e.g. on a 64-bit platform) to the following code:

```
VarHandle handle = MemoryHandles.asAddressHandle(MemoryHandles.varHandle(long.class, ByteOrder.nativeOrder()));
MemoryAddress value = (MemoryAddress)handle.get(segment, offset);
```

Parameters:

`segment` - the segment to be dereferenced.

`offset` - offset in bytes (relative to `segment`). The final address of this read operation can be expressed as `segment.address().addOffset(offset)`.

Returns:

a memory address read from `segment`.

setAddressAtOffset

```
public static void setAddressAtOffset(MemorySegment segment,
                                       long offset,
                                       Addressable value)
```

Writes a memory address at given segment and offset, with byte order set to `ByteOrder.nativeOrder()`.

This is equivalent (e.g. on a 64-bit platform) to the following code:

```
VarHandle handle = MemoryHandles.asAddressHandle(MemoryHandles.varHandle(long.class, ByteOrder.nativeOrder()));
handle.set(segment, offset, value.address());
```

Parameters:

`segment` - the segment to be dereferenced.

`offset` - offset in bytes (relative to `segment`). The final address of this read operation can be expressed as `segment.address().addOffset(offset)`.

`value` - the memory address to be written (expressed as an `Addressable` instance).

getCharAtOffset

```
public static char getCharAtOffset(MemorySegment segment,
                                   long offset,
                                   ByteOrder order)
```

Reads a char from given segment and offset with given byte order.

This is equivalent to the following code:

```
VarHandle handle = MemoryHandles.varHandle(char.class, 1, order);
char value = (char)handle.get(segment, offset);
```

Parameters:

`segment` - the segment to be dereferenced.

`offset` - offset in bytes (relative to `segment`). The final address of this read operation can be expressed as `segment.address().addOffset(offset)`.

`order` - the specified byte order.

Returns:

a char value read from `segment`.

setCharAtOffset

```
public static void setCharAtOffset(MemorySegment segment,
                                   long offset,
                                   ByteOrder order,
                                   char value)
```

Writes a char at given segment and offset with given byte order.

This is equivalent to the following code:

```
VarHandle handle = MemoryHandles.varHandle(char.class, 1, order);
handle.set(segment, offset, value);
```

Parameters:

`segment` - the segment to be dereferenced.

`offset` - offset in bytes (relative to `segment`). The final address of this read operation can be expressed as `segment.address().addOffset(offset)`.

`order` - the specified byte order.

`value` - the char value to be written.

getShortAtOffset

```
public static short getShortAtOffset(MemorySegment segment,
                                     long offset,
                                     ByteOrder order)
```

Reads a short from given segment and offset with given byte order.

This is equivalent to the following code:

```
VarHandle handle = MemoryHandles.varHandle(short.class, 1, order);
short value = (short)handle.get(segment, offset);
```

Parameters:

`segment` - the segment to be dereferenced.

`offset` - offset in bytes (relative to `segment`). The final address of this read operation can be expressed as `segment.address().addOffset(offset)`.

`order` - the specified byte order.

Returns:

a short value read from `segment`.

setShortAtOffset

```
public static void setShortAtOffset(MemorySegment segment,
                                     long offset,
                                     ByteOrder order,
                                     short value)
```

Writes a short at given segment and offset with given byte order.

This is equivalent to the following code:

```
VarHandle handle = MemoryHandles.varHandle(short.class, 1, order);
handle.set(segment, offset, value);
```

Parameters:

`segment` - the segment to be dereferenced.

`offset` - offset in bytes (relative to `segment`). The final address of this read operation can be expressed as `segment.address().addOffset(offset)`.

`order` - the specified byte order.

`value` - the short value to be written.

getIntAtOffset

```
public static int getIntAtOffset(MemorySegment segment,
                                 long offset,
                                 ByteOrder order)
```

Reads an int from given segment and offset with given byte order.

This is equivalent to the following code:

```
VarHandle handle = MemoryHandles.varHandle(int.class, 1, order);
int value = (int)handle.get(segment, offset);
```

Parameters:

`segment` - the segment to be dereferenced.

`offset` - offset in bytes (relative to `segment`). The final address of this read operation can be expressed as `segment.address().addOffset(offset)`.

`order` - the specified byte order.

Returns:

an int value read from `segment`.

setIntAtOffset

```
public static void setIntAtOffset(MemorySegment segment,
                                  long offset,
                                  ByteOrder order,
                                  int value)
```

Writes an int at given segment and offset with given byte order.

This is equivalent to the following code:

```
VarHandle handle = MemoryHandles.varHandle(int.class, 1, order);
handle.set(segment, offset, value);
```

Parameters:

`segment` - the segment to be dereferenced.

`offset` - offset in bytes (relative to `segment`). The final address of this read operation can be expressed as `segment.address().addOffset(offset)`.

`order` - the specified byte order.

`value` - the int value to be written.

getFloatAtOffset

```
public static float getFloatAtOffset(MemorySegment segment,
                                     long offset,
                                     ByteOrder order)
```

Reads a float from given segment and offset with given byte order.

This is equivalent to the following code:

```
VarHandle handle = MemoryHandles.varHandle(float.class, 1, order);
float value = (float)handle.get(segment, offset);
```

Parameters:

`segment` - the segment to be dereferenced.

`offset` - offset in bytes (relative to `segment`). The final address of this read operation can be expressed as `segment.address().addOffset(offset)`.

`order` - the specified byte order.

Returns:

a float value read from `segment`.

setFloatAtOffset

```
public static void setFloatAtOffset(MemorySegment segment,
                                    long offset,
                                    ByteOrder order,
                                    float value)
```

Writes a float at given segment and offset with given byte order.

This is equivalent to the following code:

```
VarHandle handle = MemoryHandles.varHandle(float.class, 1, order);
handle.set(segment, offset, value);
```

Parameters:

`segment` - the segment to be dereferenced.

`offset` - offset in bytes (relative to `segment`). The final address of this read operation can be expressed as `segment.address().addOffset(offset)`.

`order` - the specified byte order.

`value` - the float value to be written.

getLongAtOffset

```
public static long getLongAtOffset(MemorySegment segment,
                                   long offset,
                                   ByteOrder order)
```

Reads a long from given segment and offset with given byte order.

This is equivalent to the following code:

```
VarHandle handle = MemoryHandles.varHandle(long.class, 1, order);
long value = (long)handle.get(segment, offset);
```

Parameters:

`segment` - the segment to be dereferenced.

`offset` - offset in bytes (relative to `segment`). The final address of this read operation can be expressed as `segment.address().addOffset(offset)`.

`order` - the specified byte order.

Returns:

a long value read from `segment`.

setLongAtOffset

```
public static void setLongAtOffset(MemorySegment segment,
                                   long offset,
                                   ByteOrder order,
                                   long value)
```

Writes a long at given segment and offset with given byte order.

This is equivalent to the following code:

```
VarHandle handle = MemoryHandles.varHandle(long.class, 1, order);
handle.set(segment, offset, value);
```

Parameters:

`segment` - the segment to be dereferenced.

`offset` - offset in bytes (relative to `segment`). The final address of this read operation can be expressed as `segment.address().addOffset(offset)`.

`order` - the specified byte order.

`value` - the long value to be written.

getDoubleAtOffset

```
public static double getDoubleAtOffset(MemorySegment segment,
                                       long offset,
                                       ByteOrder order)
```

Reads a double from given segment and offset with given byte order.

This is equivalent to the following code:

```
VarHandle handle = MemoryHandles.varHandle(double.class, 1, order);
double value = (double)handle.get(segment, offset);
```

Parameters:

`segment` - the segment to be dereferenced.

`offset` - offset in bytes (relative to `segment`). The final address of this read operation can be expressed as `segment.address().addOffset(offset)`.

`order` - the specified byte order.

Returns:

a double value read from `segment`.

setDoubleAtOffset

```
public static void setDoubleAtOffset(MemorySegment segment,
                                     long offset,
                                     ByteOrder order,
                                     double value)
```

Writes a double at given segment and offset with given byte order.

This is equivalent to the following code:

```
VarHandle handle = MemoryHandles.varHandle(double.class, 1, order);
handle.set(segment, offset, value);
```

Parameters:

`segment` - the segment to be dereferenced.

`offset` - offset in bytes (relative to `segment`). The final address of this read operation can be expressed as `segment.address().addOffset(offset)`.

`order` - the specified byte order.

`value` - the double value to be written.

getByte

```
public static byte getByte(MemorySegment segment)
```

Reads a byte from given segment.

This is equivalent to the following code:

```
byte value = getByteAtOffset(segment, 0L);
```

Parameters:

`segment` - the segment to be dereferenced.

Returns:

a byte value read from `segment`.

setByte

```
public static void setByte(MemorySegment segment,
                           byte value)
```

Writes a byte at given segment.

This is equivalent to the following code:

```
setByteAtOffset(segment, 0L, value);
```

Parameters:

`segment` - the segment to be dereferenced.

`value` - the byte value to be written.

getChar

```
public static char getChar(MemorySegment segment)
```

Reads a char from given segment, with byte order set to `ByteOrder.nativeOrder()`.

This is equivalent to the following code:

```
char value = getCharAtOffset(segment, 0L);
```

Parameters:

`segment` - the segment to be dereferenced.

Returns:

a char value read from `segment`.

setChar

```
public static void setChar(MemorySegment segment,
                           char value)
```

Writes a char at given segment, with byte order set to `ByteOrder.nativeOrder()`.

This is equivalent to the following code:

```
setCharAtOffset(segment, 0L, value);
```

Parameters:

`segment` - the segment to be dereferenced.

`value` - the char value to be written.

getShort

```
public static short getShort(MemorySegment segment)
```

Reads a short from given segment, with byte order set to `ByteOrder.nativeOrder()`.

This is equivalent to the following code:

```
short value = getShortAtOffset(segment, 0L);
```

Parameters:

`segment` - the segment to be dereferenced.

Returns:

a short value read from `segment`.

setShort

```
public static void setShort(MemorySegment segment,
                            short value)
```

Writes a short at given segment, with byte order set to `ByteOrder.nativeOrder()`.

This is equivalent to the following code:

```
setShortAtOffset(segment, 0L, value);
```

Parameters:

`segment` - the segment to be dereferenced.

`value` - the short value to be written.

getInt

```
public static int getInt(MemorySegment segment)
```

Reads an int from given segment, with byte order set to `ByteOrder.nativeOrder()`.

This is equivalent to the following code:

```
int value = getIntAtOffset(segment, 0L);
```

Parameters:

`segment` - the segment to be dereferenced.

Returns:

an int value read from `segment`.

setInt

```
public static void setInt(MemorySegment segment,
                          int value)
```

Writes an int at given segment, with byte order set to `ByteOrder.nativeOrder()`.

This is equivalent to the following code:

```
setIntAtOffset(segment, 0L, value);
```

Parameters:

`segment` - the segment to be dereferenced.

`value` - the int value to be written.

getFloat

```
public static float getFloat(MemorySegment segment)
```

Reads a float from given segment, with byte order set to `ByteOrder.nativeOrder()`.

This is equivalent to the following code:

```
float value = getFloatAtOffset(segment, 0L);
```

Parameters:

`segment` - the segment to be dereferenced.

Returns:

a float value read from `segment`.

setFloat

```
public static void setFloat(MemorySegment segment,
                            float value)
```

Writes a float at given segment, with byte order set to `ByteOrder.nativeOrder()`.

This is equivalent to the following code:

```
setFloatAtOffset(segment, 0L, value);
```

Parameters:

`segment` - the segment to be dereferenced.

`value` - the float value to be written.

getLong

```
public static long getLong(MemorySegment segment)
```

Reads a long from given segment, with byte order set to `ByteOrder.nativeOrder()`.

This is equivalent to the following code:

```
long value = getLongAtOffset(segment, 0L);
```

Parameters:

`segment` - the segment to be dereferenced.

Returns:

a long value read from `segment`.

setLong

```
public static void setLong(MemorySegment segment,
                           long value)
```

Writes a long at given segment, with byte order set to `ByteOrder.nativeOrder()`.

This is equivalent to the following code:

```
setLongAtOffset(segment, 0L, value);
```

Parameters:

`segment` - the segment to be dereferenced.

`value` - the long value to be written.

getDouble

```
public static double getDouble(MemorySegment segment)
```

Reads a double from given segment, with byte order set to `ByteOrder.nativeOrder()`.

This is equivalent to the following code:

```
double value = getDoubleAtOffset(segment, 0L);
```

Parameters:

`segment` - the segment to be dereferenced.

Returns:

a double value read from `segment`.

setDouble

```
public static void setDouble(MemorySegment segment,
                             double value)
```

Writes a double at given segment, with byte order set to `ByteOrder.nativeOrder()`.

This is equivalent to the following code:

```
setDoubleAtOffset(segment, 0L, value);
```

Parameters:

`segment` - the segment to be dereferenced.

`value` - the double value to be written.

getAddress

```
public static MemoryAddress getAddress(MemorySegment segment)
```

Reads a memory address from given segment, with byte order set to `ByteOrder.nativeOrder()`.

This is equivalent to the following code:

```
MemoryAddress value = getAddressAtOffset(segment, 0L);
```

Parameters:

`segment` - the segment to be dereferenced.

Returns:

a memory address read from `segment`.

setAddress

```
public static void setAddress(MemorySegment segment,
                              Addressable value)
```

Writes a memory address at given segment, with byte order set to `ByteOrder.nativeOrder()`.

This is equivalent to the following code:

```
setAddressAtOffset(segment, 0L, value);
```

Parameters:

`segment` - the segment to be dereferenced.

`value` - the memory address to be written (expressed as an `Addressable` instance).

getChar

```
public static char getChar(MemorySegment segment,
                           ByteOrder order)
```

Reads a char from given segment, with given byte order.

This is equivalent to the following code:

```
char value = getCharAtOffset(segment, 0L, order);
```

Parameters:

`segment` - the segment to be dereferenced.

`order` - the specified byte order.

Returns:

a char value read from `segment`.

setChar

```
public static void setChar(MemorySegment segment,
                           ByteOrder order,
                           char value)
```

Writes a char at given segment, with given byte order.

This is equivalent to the following code:

```
setCharAtOffset(segment, 0L, order, value);
```

Parameters:

`segment` - the segment to be dereferenced.

`order` - the specified byte order.

`value` - the char value to be written.

getShort

```
public static short getShort(MemorySegment segment,
                             ByteOrder order)
```

Reads a short from given segment, with given byte order.

This is equivalent to the following code:

```
short value = getShortAtOffset(segment, 0L, order);
```

Parameters:

`segment` - the segment to be dereferenced.

`order` - the specified byte order.

Returns:

a short value read from `segment`.

setShort

```
public static void setShort(MemorySegment segment,
                            ByteOrder order,
                            short value)
```

Writes a short at given segment, with given byte order.

This is equivalent to the following code:

```
setShortAtOffset(segment, 0L, order, value);
```

Parameters:

`segment` - the segment to be dereferenced.

`order` - the specified byte order.

`value` - the short value to be written.

getInt

```
public static int getInt(MemorySegment segment,
                         ByteOrder order)
```

Reads an int from given segment, with given byte order.

This is equivalent to the following code:

```
int value = getIntAtOffset(segment, 0L, order);
```

Parameters:

`segment` - the segment to be dereferenced.

`order` - the specified byte order.

Returns:

an int value read from `segment`.

setInt

```
public static void setInt(MemorySegment segment,
                          ByteOrder order,
                          int value)
```

Writes an int at given segment, with given byte order.

This is equivalent to the following code:

```
setIntAtOffset(segment, 0L, order, value);
```

Parameters:

`segment` - the segment to be dereferenced.

`order` - the specified byte order.

`value` - the int value to be written.

getFloat

```
public static float getFloat(MemorySegment segment,
                             ByteOrder order)
```

Reads a float from given segment, with given byte order.

This is equivalent to the following code:

```
float value = getFloatAtOffset(segment, 0L, order);
```

Parameters:

`segment` - the segment to be dereferenced.

`order` - the specified byte order.

Returns:

a float value read from `segment`.

setFloat

```
public static void setFloat(MemorySegment segment,
                            ByteOrder order,
                            float value)
```

Writes a float at given segment, with given byte order.

This is equivalent to the following code:

```
        setFloatAtOffset(segment, 0L, order, value);
```

Parameters:

`segment` - the segment to be dereferenced.

`order` - the specified byte order.

`value` - the float value to be written.

getLong

```
public static long getLong(MemorySegment segment,
                           ByteOrder order)
```

Reads a long from given segment, with given byte order.

This is equivalent to the following code:

```
        long value = getLongAtOffset(segment, 0L, order);
```

Parameters:

`segment` - the segment to be dereferenced.

`order` - the specified byte order.

Returns:

a long value read from `segment`.

setLong

```
public static void setLong(MemorySegment segment,
                           ByteOrder order,
                           long value)
```

Writes a long at given segment, with given byte order.

This is equivalent to the following code:

```
        setLongAtOffset(segment, 0L, order, value);
```

Parameters:

`segment` - the segment to be dereferenced.

`order` - the specified byte order.

`value` - the long value to be written.

getDouble

```
public static double getDouble(MemorySegment segment,
                               ByteOrder order)
```

Reads a double from given segment, with given byte order.

This is equivalent to the following code:

```
        double value = getDoubleAtOffset(segment, 0L, order);
```

Parameters:

`segment` - the segment to be dereferenced.

`order` - the specified byte order.

Returns:

a double value read from `segment`.

setDouble

```
public static void setDouble(MemorySegment segment,
                             ByteOrder order,
                             double value)
```

Writes a double at given segment, with given byte order.

This is equivalent to the following code:

```
setDoubleAtOffset(segment, 0L, order, value);
```

Parameters:

`segment` - the segment to be dereferenced.

`order` - the specified byte order.

`value` - the double value to be written.

getCharAtIndex

```
public static char getCharAtIndex(MemorySegment segment,
                                  long index)
```

Reads a char from given segment and element index, with byte order set to `ByteOrder.nativeOrder()`.

This is equivalent to the following code:

```
char value = getCharAtOffset(segment, 2 * index);
```

Parameters:

`segment` - the segment to be dereferenced.

`index` - element index (relative to `segment`). The final address of this read operation can be expressed as `segment.address().addOffset(index * 2)`.

Returns:

a char value read from `segment` at the element index specified by `index`.

setCharAtIndex

```
public static void setCharAtIndex(MemorySegment segment,
                                  long index,
                                  char value)
```

Writes a char at given segment and element index, with byte order set to `ByteOrder.nativeOrder()`.

This is equivalent to the following code:

```
setCharAtOffset(segment, 2 * index, value);
```

Parameters:

`segment` - the segment to be dereferenced.

`index` - element index (relative to `segment`). The final address of this read operation can be expressed as `segment.address().addOffset(index * 2)`.

`value` - the char value to be written.

getShortAtIndex

```
public static short getShortAtIndex(MemorySegment segment,
                                    long index)
```

Reads a short from given segment and element index, with byte order set to `ByteOrder.nativeOrder()`.

This is equivalent to the following code:

```
short value = getShortAtOffset(segment, 2 * index);
```

Parameters:

`segment` - the segment to be dereferenced.

`index` - element index (relative to `segment`). The final address of this read operation can be expressed as `segment.address().addOffset(index * 2)`.

Returns:

a short value read from `segment` at the element index specified by `index`.

setShortAtIndex

```
public static void setShortAtIndex(MemorySegment segment,
                                   long index,
                                   short value)
```

Writes a short at given segment and element index, with byte order set to `ByteOrder.nativeOrder()`.

This is equivalent to the following code:

```
setShortAtOffset(segment, 2 * index, value);
```

Parameters:

`segment` - the segment to be dereferenced.

`index` - element index (relative to `segment`). The final address of this read operation can be expressed as `segment.address().addOffset(index * 2)`.

`value` - the short value to be written.

getIntAtIndex

```
public static int getIntAtIndex(MemorySegment segment,
                                long index)
```

Reads an int from given segment and element index, with byte order set to `ByteOrder.nativeOrder()`.

This is equivalent to the following code:

```
int value = getIntAtOffset(segment, 4 * index);
```

Parameters:

`segment` - the segment to be dereferenced.

`index` - element index (relative to `segment`). The final address of this read operation can be expressed as `segment.address().addOffset(index * 4)`.

Returns:

an int value read from `segment` at the element index specified by `index`.

setIntAtIndex

```
public static void setIntAtIndex(MemorySegment segment,
                                 long index,
                                 int value)
```

Writes an int at given segment and element index, with byte order set to `ByteOrder.nativeOrder()`.

This is equivalent to the following code:

```
setIntAtOffset(segment, 4 * index, value);
```

Parameters:

`segment` - the segment to be dereferenced.

`index` - element index (relative to `segment`). The final address of this read operation can be expressed as `segment.address().addOffset(index * 4)`.

`value` - the int value to be written.

getFloatAtIndex

```
public static float getFloatAtIndex(MemorySegment segment,
                                    long index)
```

Reads a float from given segment and element index, with byte order set to `ByteOrder.nativeOrder()`.

This is equivalent to the following code:

```
float value = getFloatAtOffset(segment, 4 * index);
```

Parameters:

`segment` - the segment to be dereferenced.

`index` - element index (relative to `segment`). The final address of this read operation can be expressed as `segment.address().addOffset(index * 4)`.

Returns:

a float value read from `segment` at the element index specified by `index`.

setFloatAtIndex

```
public static void setFloatAtIndex(MemorySegment segment,
                                   long index,
                                   float value)
```

Writes a float at given segment and element index, with byte order set to `ByteOrder.nativeOrder()`.

This is equivalent to the following code:

```
setFloatAtOffset(segment, 4 * index, value);
```

Parameters:

`segment` - the segment to be dereferenced.

`index` - element index (relative to `segment`). The final address of this read operation can be expressed as `segment.address().addOffset(index * 4)`.

`value` - the float value to be written.

getLongAtIndex

```
public static long getLongAtIndex(MemorySegment segment,
                                  long index)
```

Reads a long from given segment and element index, with byte order set to `ByteOrder.nativeOrder()`.

This is equivalent to the following code:

```
return getLongAtOffset(segment, 8 * index);
```

Parameters:

`segment` - the segment to be dereferenced.

`index` - element index (relative to `segment`). The final address of this read operation can be expressed as `segment.address().addOffset(index * 8)`.

Returns:

a long value read from `segment` at the element index specified by `index`.

setLongAtIndex

```
public static void setLongAtIndex(MemorySegment segment,
                                  long index,
                                  long value)
```

Writes a long at given segment and element index, with byte order set to `ByteOrder.nativeOrder()`.

This is equivalent to the following code:

```
setLongAtOffset(segment, 8 * index, value);
```

Parameters:

`segment` - the segment to be dereferenced.

`index` - element index (relative to `segment`). The final address of this read operation can be expressed as `segment.address().addOffset(index * 8)`.

`value` - the long value to be written.

getDoubleAtIndex

```
public static double getDoubleAtIndex(MemorySegment segment,
                                      long index)
```

Reads a double from given segment and element index, with byte order set to `ByteOrder.nativeOrder()`.

This is equivalent to the following code:

```
return getDoubleAtOffset(segment, 8 * index);
```

Parameters:

`segment` - the segment to be dereferenced.

`index` - element index (relative to `segment`). The final address of this read operation can be expressed as `segment.address().addOffset(index * 8)`.

Returns:

a double value read from `segment` at the element index specified by `index`.

setDoubleAtIndex

```
public static void setDoubleAtIndex(MemorySegment segment,
                                     long index,
                                     double value)
```

Writes a double at given segment and element index, with byte order set to `ByteOrder.nativeOrder()`.

This is equivalent to the following code:

```
        setDoubleAtOffset(segment, 8 * index, value);
```

Parameters:

`segment` - the segment to be dereferenced.

`index` - element index (relative to `segment`). The final address of this read operation can be expressed as `segment.address().addOffset(index * 8)`.

`value` - the double value to be written.

getAddressAtIndex

```
public static MemoryAddress getAddressAtIndex(MemorySegment segment,
                                              long index)
```

Reads a memory address from given segment and element index, with byte order set to `ByteOrder.nativeOrder()`.

This is equivalent to the following code:

```
        return getAddressAtOffset(segment, index * MemoryLayouts.ADDRESS.byteSize());
```

Parameters:

`segment` - the segment to be dereferenced.

`index` - element index (relative to `segment`). The final address of this read operation can be expressed as `segment.address().addOffset(index * 8)`.

Returns:

a memory address read from `segment` at the element index specified by `index`.

setAddressAtIndex

```
public static void setAddressAtIndex(MemorySegment segment,
                                     long index,
                                     Addressable value)
```

Writes a memory address at given segment and element index, with byte order set to `ByteOrder.nativeOrder()`.

This is equivalent to the following code:

```
        setAddressAtOffset(segment, index * MemoryLayouts.ADDRESS.byteSize(), value);
```

Parameters:

`segment` - the segment to be dereferenced.

`index` - element index (relative to `segment`). The final address of this read operation can be expressed as `segment.address().addOffset(index * 8)`.

`value` - the memory address to be written (expressed as an `Addressable` instance).

getCharAtIndex

```
public static char getCharAtIndex(MemorySegment segment,
                                  long index,
                                  ByteOrder order)
```

Reads a char from given segment and element index, with given byte order.

This is equivalent to the following code:

```
        char value = getCharAtOffset(segment, 2 * index, order);
```

Parameters:

`segment` - the segment to be dereferenced.

`index` - element index (relative to `segment`). The final address of this read operation can be expressed as `segment.address().addOffset(index * 2)`.

`order` - the specified byte order.

Returns:

a char value read from `segment` at the element index specified by `index`.

setCharAtIndex

```
public static void setCharAtIndex(MemorySegment segment,
                                  long index,
                                  ByteOrder order,
                                  char value)
```

Writes a char at given segment and element index, with given byte order.

This is equivalent to the following code:

```
        setCharAtOffset(segment, 2 * index, order, value);
```

Parameters:

`segment` - the segment to be dereferenced.

`index` - element index (relative to `segment`). The final address of this read operation can be expressed as `segment.address().addOffset(index * 2)`.

`order` - the specified byte order.

`value` - the char value to be written.

getShortAtIndex

```
public static short getShortAtIndex(MemorySegment segment,
                                    long index,
                                    ByteOrder order)
```

Reads a short from given segment and element index, with given byte order.

This is equivalent to the following code:

```
        short value = getShortAtOffset(segment, 2 * index, order);
```

Parameters:

`segment` - the segment to be dereferenced.

`index` - element index (relative to `segment`). The final address of this read operation can be expressed as `segment.address().addOffset(index * 2)`.

`order` - the specified byte order.

Returns:

a short value read from `segment` at the element index specified by `index`.

setShortAtIndex

```
public static void setShortAtIndex(MemorySegment segment,
                                   long index,
                                   ByteOrder order,
                                   short value)
```

Writes a short at given segment and element index, with given byte order.

This is equivalent to the following code:

```
        setShortAtOffset(segment, 2 * index, order, value);
```

Parameters:

`segment` - the segment to be dereferenced.

`index` - element index (relative to `segment`). The final address of this read operation can be expressed as `segment.address().addOffset(index * 2)`.

`order` - the specified byte order.

`value` - the short value to be written.

getIntAtIndex

```
public static int getIntAtIndex(MemorySegment segment,
                                long index,
                                ByteOrder order)
```

Reads an int from given segment and element index, with given byte order.

This is equivalent to the following code:

```
int value = getIntAtOffset(segment, 4 * index, order);
```

Parameters:

`segment` - the segment to be dereferenced.

`index` - element index (relative to `segment`). The final address of this read operation can be expressed as `segment.address().addOffset(index * 4)`.

`order` - the specified byte order.

Returns:

an int value read from `segment` at the element index specified by `index`.

setIntAtIndex

```
public static void setIntAtIndex(MemorySegment segment,
                                 long index,
                                 ByteOrder order,
                                 int value)
```

Writes an int at given segment and element index, with given byte order.

This is equivalent to the following code:

```
setIntAtOffset(segment, 4 * index, order, value);
```

Parameters:

`segment` - the segment to be dereferenced.

`index` - element index (relative to `segment`). The final address of this read operation can be expressed as `segment.address().addOffset(index * 4)`.

`order` - the specified byte order.

`value` - the int value to be written.

getFloatAtIndex

```
public static float getFloatAtIndex(MemorySegment segment,
                                    long index,
                                    ByteOrder order)
```

Reads a float from given segment and element index, with given byte order.

This is equivalent to the following code:

```
float value = getFloatAtOffset(segment, 4 * index, order);
```

Parameters:

`segment` - the segment to be dereferenced.

`index` - element index (relative to `segment`). The final address of this read operation can be expressed as `segment.address().addOffset(index * 4)`.

`order` - the specified byte order.

Returns:

a float value read from `segment` at the element index specified by `index`.

setFloatAtIndex

```
public static void setFloatAtIndex(MemorySegment segment,
                                   long index,
                                   ByteOrder order,
                                   float value)
```

Writes a float at given segment and element index, with given byte order.

This is equivalent to the following code:

```
setFloatAtOffset(segment, 4 * index, order, value);
```

Parameters:

`segment` - the segment to be dereferenced.

`index` - element index (relative to `segment`). The final address of this read operation can be expressed as `segment.address().addOffset(index * 4)`.

`order` - the specified byte order.

`value` - the float value to be written.

getLongAtIndex

```
public static long getLongAtIndex(MemorySegment segment,
                                  long index,
                                  ByteOrder order)
```

Reads a long from given segment and element index, with given byte order.

This is equivalent to the following code:

```
        return getLongAtOffset(segment, 8 * index, order);
```

Parameters:

`segment` - the segment to be dereferenced.

`index` - element index (relative to `segment`). The final address of this read operation can be expressed as `segment.address().addOffset(index * 8)`.

`order` - the specified byte order.

Returns:

a long value read from `segment` at the element index specified by `index`.

setLongAtIndex

```
public static void setLongAtIndex(MemorySegment segment,
                                  long index,
                                  ByteOrder order,
                                  long value)
```

Writes a long at given segment and element index, with given byte order.

This is equivalent to the following code:

```
        setLongAtOffset(segment, 8 * index, order, value);
```

Parameters:

`segment` - the segment to be dereferenced.

`index` - element index (relative to `segment`). The final address of this read operation can be expressed as `segment.address().addOffset(index * 8)`.

`order` - the specified byte order.

`value` - the long value to be written.

getDoubleAtIndex

```
public static double getDoubleAtIndex(MemorySegment segment,
                                      long index,
                                      ByteOrder order)
```

Reads a double from given segment and element index, with given byte order.

This is equivalent to the following code:

```
        return getDoubleAtOffset(segment, 8 * index, order);
```

Parameters:

`segment` - the segment to be dereferenced.

`index` - element index (relative to `segment`). The final address of this read operation can be expressed as `segment.address().addOffset(index * 8)`.

`order` - the specified byte order.

Returns:

a double value read from `segment` at the element index specified by `index`.

setDoubleAtIndex

```
public static void setDoubleAtIndex(MemorySegment segment,
                                     long index,
                                     ByteOrder order,
                                     double value)
```

Writes a double at given segment and element index, with given byte order.

This is equivalent to the following code:

```
setDoubleAtOffset(segment, 8 * index, order, value);
```

Parameters:

`segment` - the segment to be dereferenced.

`index` - element index (relative to `segment`). The final address of this read operation can be expressed as `segment.address().addOffset(index * 8)`.

`order` - the specified byte order.

`value` - the double value to be written.
