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

Class ValueLayout

java.lang.Object
    jdk.incubator.foreign.ValueLayout

All Implemented Interfaces:

`Constable`, `MemoryLayout`

Direct Known Subclasses:

`ValueLayout.OfAddress`, `ValueLayout.OfBoolean`, `ValueLayout.OfByte`, `ValueLayout.OfChar`, `ValueLayout.OfDouble`, `ValueLayout.OfFloat`, `ValueLayout.OfInt`, `ValueLayout.OfLong`, `ValueLayout.OfShort`

```
public sealed class ValueLayout
extends Object
implements MemoryLayout
permits ValueLayout.OfAddress, ValueLayout.OfByte, ValueLayout.OfBoolean, ValueLayout.OfChar,
ValueLayout.OfShort, ValueLayout.OfInt, ValueLayout.OfLong, ValueLayout.OfFloat, ValueLayout.OfDouble
```

A value layout. A value layout is used to model the memory layout associated with values of basic data types, such as *integral* types (either signed or unsigned) and *floating-point* types. Each value layout has a size, an alignment (in bits), a byte order, and a *carrier*, that is, the Java type that should be used when accessing a memory region using the value layout.

This class defines useful value layout constants for Java primitive types and addresses. The layout constants in this class make implicit alignment and byte-ordering assumption: all layout constants in this class are byte-aligned, and their byte order is set to the platform default, thus making it easy to work with other APIs, such as arrays and `ByteBuffer`.

This is a value-based class; programmers should treat instances that are equal as interchangeable and should not use instances for synchronization, or unpredictable behavior may occur. For example, in a future release, synchronization may fail. The `equals` method should be used for comparisons.

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.

Implementation Requirements:

This class is immutable and thread-safe.

Nested Class Summary

Nested Classes

Modifier and Type	Class	Description
static final class	**ValueLayout.OfAddress**	A value layout whose carrier is `MemoryAddress.class`.
static final class	**ValueLayout.OfBoolean**	A value layout whose carrier is `boolean.class`.
static final class	**ValueLayout.OfByte**	A value layout whose carrier is `byte.class`.
static final class	**ValueLayout.OfChar**	A value layout whose carrier is `char.class`.
static final class	**ValueLayout.OfDouble**	A value layout whose carrier is `double.class`.
static final class	**ValueLayout.OfFloat**	A value layout whose carrier is `float.class`.
static final class	**ValueLayout.OfInt**	A value layout whose carrier is `int.class`.
static final class	**ValueLayout.OfLong**	A value layout whose carrier is `long.class`.
static final class	**ValueLayout.OfShort**	A value layout whose carrier is `short.class`.

Nested classes/interfaces declared in interface jdk.incubator.foreign.MemoryLayout

`MemoryLayout.PathElement`

Field Summary

Fields

Modifier and Type	Field	Description
static final `ValueLayout.OfAddress`	**ADDRESS**	A value layout constant whose size is the same as that of a machine address (`size_t`), bit alignment set to 8, and byte order set to `ByteOrder.nativeOrder()`.

static final ValueLayout.OfBoolean	JAVA_BOOLEAN	A value layout constant whose size is the same as that of a Java boolean, bit alignment set to 8, and byte order set to ByteOrder.nativeOrder().
static final ValueLayout.OfByte	JAVA_BYTE	A value layout constant whose size is the same as that of a Java byte, bit alignment set to 8, and byte order set to ByteOrder.nativeOrder().
static final ValueLayout.OfChar	JAVA_CHAR	A value layout constant whose size is the same as that of a Java char, bit alignment set to 8, and byte order set to ByteOrder.nativeOrder().
static final ValueLayout.OfDouble	JAVA_DOUBLE	A value layout constant whose size is the same as that of a Java double, bit alignment set to 8, and byte order set to ByteOrder.nativeOrder().
static final ValueLayout.OfFloat	JAVA_FLOAT	A value layout constant whose size is the same as that of a Java float, bit alignment set to 8, and byte order set to ByteOrder.nativeOrder().
static final ValueLayout.OfInt	JAVA_INT	A value layout constant whose size is the same as that of a Java int, bit alignment set to 8, and byte order set to ByteOrder.nativeOrder().
static final ValueLayout.OfLong	JAVA_LONG	A value layout constant whose size is the same as that of a Java long, bit alignment set to 8, and byte order set to ByteOrder.nativeOrder().
static final ValueLayout.OfShort	JAVA_SHORT	A value layout constant whose size is the same as that of a Java short, bit alignment set to 8, and byte order set to ByteOrder.nativeOrder().

Method Summary

All Methods | Instance Methods | Concrete Methods

Modifier and Type	Method	Description
final long	bitAlignment()	Returns the alignment constraint associated with this layout, expressed in bits.
long	bitSize()	Returns the layout size, in bits.
long	byteSize()	Returns the layout size, in bytes.
Class<?>	carrier()	Returns the carrier associated with this value layout.
Optional<DynamicConstantDesc<Val	describeConstable()	Returns an Optional containing the nominal descriptor for this layout, if one can be constructed, or an empty Optional if one cannot be constructed.
boolean	equals(Object other)	Indicates whether some other object is "equal to" this one.
int	hashCode()	Returns a hash code value for the object.
boolean	hasSize()	Returns true if this layout has a specified size.
boolean	isPadding()	Returns true, if this layout is a padding layout.
final Optional<String>	name()	Returns the *name* (if any) associated with this layout.
ByteOrder	order()	Returns the value's byte order.
String	toString()	Returns a string representation of the object.
ValueLayout	withBitAlignment(long alignmentBits)	Creates a new layout which features the desired alignment constraint.
ValueLayout	withName(String name)	Creates a new layout which features the desired layout *name*.
ValueLayout	withOrder(ByteOrder order)	Returns a new value layout with given byte order.

Methods declared in class java.lang.Object

clone, finalize, getClass, notify, notifyAll, toString, wait, wait, wait

Methods declared in interface jdk.incubator.foreign.MemoryLayout

bitAlignment, bitOffset, bitOffsetHandle, bitSize, byteAlignment, byteOffset, byteOffsetHandle, byteSize, hasSize, isPadding, map, name, select, sliceHandle, varHandle

Field Details

ADDRESS

```
public static final ValueLayout.OfAddress ADDRESS
```

A value layout constant whose size is the same as that of a machine address (`size_t`), bit alignment set to 8, and byte order set to `ByteOrder.nativeOrder()`. Equivalent to the following code:

```
MemoryLayout.valueLayout(MemoryAddress.class, ByteOrder.nativeOrder()).withBitAlignment(8);
```

JAVA_BYTE

```
public static final ValueLayout.OfByte JAVA_BYTE
```

A value layout constant whose size is the same as that of a Java `byte`, bit alignment set to 8, and byte order set to `ByteOrder.nativeOrder()`. Equivalent to the following code:

```
MemoryLayout.valueLayout(byte.class, ByteOrder.nativeOrder()).withBitAlignment(8);
```

JAVA_BOOLEAN

```
public static final ValueLayout.OfBoolean JAVA_BOOLEAN
```

A value layout constant whose size is the same as that of a Java `boolean`, bit alignment set to 8, and byte order set to `ByteOrder.nativeOrder()`. Equivalent to the following code:

```
MemoryLayout.valueLayout(boolean.class, ByteOrder.nativeOrder()).withBitAlignment(8);
```

JAVA_CHAR

```
public static final ValueLayout.OfChar JAVA_CHAR
```

A value layout constant whose size is the same as that of a Java `char`, bit alignment set to 8, and byte order set to `ByteOrder.nativeOrder()`. Equivalent to the following code:

```
MemoryLayout.valueLayout(char.class, ByteOrder.nativeOrder()).withBitAlignment(8);
```

JAVA_SHORT

```
public static final ValueLayout.OfShort JAVA_SHORT
```

A value layout constant whose size is the same as that of a Java `short`, bit alignment set to 8, and byte order set to `ByteOrder.nativeOrder()`. Equivalent to the following code:

```
MemoryLayout.valueLayout(short.class, ByteOrder.nativeOrder()).withBitAlignment(8);
```

JAVA_INT

```
public static final ValueLayout.OfInt JAVA_INT
```

A value layout constant whose size is the same as that of a Java `int`, bit alignment set to 8, and byte order set to `ByteOrder.nativeOrder()`. Equivalent to the following code:

```
MemoryLayout.valueLayout(int.class, ByteOrder.nativeOrder()).withBitAlignment(8);
```

JAVA_LONG

```
public static final ValueLayout.OfLong JAVA_LONG
```

A value layout constant whose size is the same as that of a Java `long`, bit alignment set to 8, and byte order set to `ByteOrder.nativeOrder()`. Equivalent to the following code:

```
MemoryLayout.valueLayout(long.class, ByteOrder.nativeOrder()).withBitAlignment(8);
```

JAVA_FLOAT

```
public static final ValueLayout.OfFloat JAVA_FLOAT
```

A value layout constant whose size is the same as that of a Java `float`, bit alignment set to 8, and byte order set to `ByteOrder.nativeOrder()`. Equivalent to the following code:

```
MemoryLayout.valueLayout(float.class, ByteOrder.nativeOrder()).withBitAlignment(8);
```

JAVA_DOUBLE

```
public static final ValueLayout.OfDouble JAVA_DOUBLE
```

A value layout constant whose size is the same as that of a Java `double`, bit alignment set to 8, and byte order set to `ByteOrder.nativeOrder()`. Equivalent to the following code:

```
MemoryLayout.valueLayout(double.class, ByteOrder.nativeOrder()).withBitAlignment(8);
```

Method Details

order

```
public ByteOrder order()
```

Returns the value's byte order.

Returns:
the value's byte order

withOrder

```
public ValueLayout withOrder(ByteOrder order)
```

Returns a new value layout with given byte order.

Parameters:
`order` - the desired byte order.

Returns:
a new value layout with given byte order.

toString

```
public String toString()
```

Description copied from class: `Object`
Returns a string representation of the object.

Specified by:
`toString` in interface `MemoryLayout`

Overrides:
`toString` in class `Object`

Returns:
a string representation of the object.

equals

```
public boolean equals(Object other)
```

Description copied from class: `Object`
Indicates whether some other object is "equal to" this one.

The `equals` method implements an equivalence relation on non-null object references:

- It is *reflexive*: for any non-null reference value `x`, `x.equals(x)` should return `true`.
- It is *symmetric*: for any non-null reference values `x` and `y`, `x.equals(y)` should return `true` if and only if `y.equals(x)` returns `true`.
- It is *transitive*: for any non-null reference values `x`, `y`, and `z`, if `x.equals(y)` returns `true` and `y.equals(z)` returns `true`, then `x.equals(z)` should return `true`.
- It is *consistent*: for any non-null reference values `x` and `y`, multiple invocations of `x.equals(y)` consistently return `true` or consistently return `false`, provided no information used in `equals` comparisons on the objects is modified.
- For any non-null reference value `x`, `x.equals(null)` should return `false`.

An equivalence relation partitions the elements it operates on into *equivalence classes*; all the members of an equivalence class are equal to each other. Members of an equivalence class are substitutable for each other, at least for some purposes.

Specified by:

`equals` in interface `MemoryLayout`

Parameters:

`other` - the reference object with which to compare.

Returns:

`true` if this object is the same as the obj argument; `false` otherwise.

See Also:

`Object.hashCode()`, `HashMap`

carrier

```
public Class<?> carrier()
```

Returns the carrier associated with this value layout.

Returns:

the carrier associated with this value layout

hashCode

```
public int hashCode()
```

Description copied from class: `Object`

Returns a hash code value for the object. This method is supported for the benefit of hash tables such as those provided by `HashMap`.

The general contract of `hashCode` is:

- Whenever it is invoked on the same object more than once during an execution of a Java application, the `hashCode` method must consistently return the same integer, provided no information used in `equals` comparisons on the object is modified. This integer need not remain consistent from one execution of an application to another execution of the same application.
- If two objects are equal according to the `equals` method, then calling the `hashCode` method on each of the two objects must produce the same integer result.
- It is *not* required that if two objects are unequal according to the `equals` method, then calling the `hashCode` method on each of the two objects must produce distinct integer results. However, the programmer should be aware that producing distinct integer results for unequal objects may improve the performance of hash tables.

Specified by:

`hashCode` in interface `MemoryLayout`

Returns:

a hash code value for this object.

See Also:

`Object.equals(java.lang.Object)`, `System.identityHashCode(java.lang.Object)`

describeConstable

```
public Optional<DynamicConstantDesc<ValueLayout>> describeConstable()
```

Description copied from interface: `MemoryLayout`

Returns an `Optional` containing the nominal descriptor for this layout, if one can be constructed, or an empty `Optional` if one cannot be constructed.

Specified by:

`describeConstable` in interface `Constable`

Specified by:

`describeConstable` in interface `MemoryLayout`

Returns:

an `Optional` containing the nominal descriptor for this layout, if one can be constructed, or an empty `Optional` if one cannot be constructed

withName

`public ValueLayout withName(String name)`

Creates a new layout which features the desired layout *name*.

Specified by:

`withName` in interface `MemoryLayout`

Parameters:

`name` - the layout name.

Returns:

a new layout which is the same as this layout, except for the *name* associated with it.

See Also:

`MemoryLayout.name()`

withBitAlignment

`public ValueLayout withBitAlignment(long alignmentBits)`

Creates a new layout which features the desired alignment constraint.

Specified by:

`withBitAlignment` in interface `MemoryLayout`

Parameters:

`alignmentBits` - the layout alignment constraint, expressed in bits.

Returns:

a new layout which is the same as this layout, except for the alignment constraint associated with it.

name

`public final Optional<String> name()`

Description copied from interface: `MemoryLayout`

Returns the *name* (if any) associated with this layout.

Specified by:

`name` in interface `MemoryLayout`

Returns:

the *name* (if any) associated with this layout

See Also:

`MemoryLayout.withName(String)`

bitAlignment

`public final long bitAlignment()`

Description copied from interface: `MemoryLayout`

Returns the alignment constraint associated with this layout, expressed in bits. Layout alignment defines a power of two `A` which is the bit-wise alignment of the layout. If `A <= 8` then `A/8` is the number of bytes that must be aligned for any pointer that correctly points to this layout. Thus:

- `A=8` means unaligned (in the usual sense), which is common in packets.
- `A=64` means word aligned (on LP64), `A=32` int aligned, `A=16` short aligned, etc.
- `A=512` is the most strict alignment required by the x86/SV ABI (for AVX-512 data).

If no explicit alignment constraint was set on this layout (see `MemoryLayout.withBitAlignment(long)`), then this method returns the natural alignment constraint (in bits) associated with this layout.

Specified by:

`bitAlignment` in interface `MemoryLayout`

Returns:

the layout alignment constraint, in bits.

byteSize

`public long byteSize()`

Description copied from interface: `MemoryLayout`

Returns the layout size, in bytes.

Specified by:

`byteSize` in interface `MemoryLayout`

Returns:

the layout size, in bytes

hasSize

```
public boolean hasSize()
```

Description copied from interface: `MemoryLayout`

Returns `true` if this layout has a specified size. A layout does not have a specified size if it is (or contains) a sequence layout whose size is unspecified (see `SequenceLayout.elementCount()`). Value layouts (see `ValueLayout`) and padding layouts (see `MemoryLayout.paddingLayout(long)`) *always* have a specified size, therefore this method always returns `true` in these cases.

Specified by:

`hasSize` in interface `MemoryLayout`

Returns:

`true`, if this layout has a specified size.

bitSize

```
public long bitSize()
```

Description copied from interface: `MemoryLayout`

Returns the layout size, in bits.

Specified by:

`bitSize` in interface `MemoryLayout`

Returns:

the layout size, in bits

isPadding

```
public boolean isPadding()
```

Description copied from interface: `MemoryLayout`

Returns true, if this layout is a padding layout.

Specified by:

`isPadding` in interface `MemoryLayout`

Returns:

true, if this layout is a padding layout