Java Objects Inside Out

Aleksey Shipilëv
shade@redhat.com
@shipilev

Safe Harbor / Тихая Гавань

Anything on this or any subsequent slides may be a lie. Do not base your decisions on this talk. If you do, ask for professional help.

Всё что угодно на этом слайде, как и на всех следующих, может быть враньём. Не принимайте решений на основании этого доклада. Если всё-таки решите принять, то наймите профессионалов.

Intro

Intro: Mood

The overwhelming majority of things in this talk are **implementation-specific to Hotspot JVM**

But, choices are driven by technical considerations, so they:

- ...solve similar challenges in JVM impls
- ...frequently converge to same solutions
- ...somewhat transfer to other VMs and languages

Intro: Text Version

These slides are mostly for visual aids.
For consistent narrative and discussion look here:

`https://shipilev.net/jvm/objects-inside-out`

Methodology

Heap Dumps: HPROF format

CLASS DUMP

u4	instance size (in bytes)
u2	Number of static fields:
	ID: static field name string ID
	u1: type of field: (See Basic Type)
	value: value of entry (u1, u2, u4, or u8 based on type of field)
u2	Number of instance fields (not including super class's)
	ID: field name string ID
	u1: type of field: (See Basic Type)

Heap Dumps: HPROF format

CLASS DUMP

u4	instance size (in bytes)
u2	Number of static fields:
	ID: static field name string ID
	u1: type of field: (See Basic Type)
	value: value of entry (u1, u
u2	Number of instance fields (not including super class's)
	ID: field name string ID
	u1: type of field: (See Basic Type)

Field order is unspecified, does not to have to match runtime

Heap Dumps: HPROF format

CLASS DUMP

u4	instance size (in bytes)
u2	Number of static fields:

ID	static field name string ID
u1	type of field: (See Basic Type)
value	value of entry (u1, u

u2	Number of instance fields (not including super class's)

ID	field name string ID
u1	type of field: (See Basic Type)

This is a sum of all field sizes (a blatant lie![a])

[a]JDK-8005604, abandoned

Field order is unspecified, does not to have to match runtime

Heap Dumps: HPROF format, #2

INSTANCE DUMP

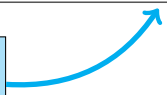
ID	object ID
u4	stack trace serial number
ID	class object ID
u4	number of bytes that follow
[value]*	instance field values (this class, followed by super class, etc)

Heap Dumps: HPROF format, #2

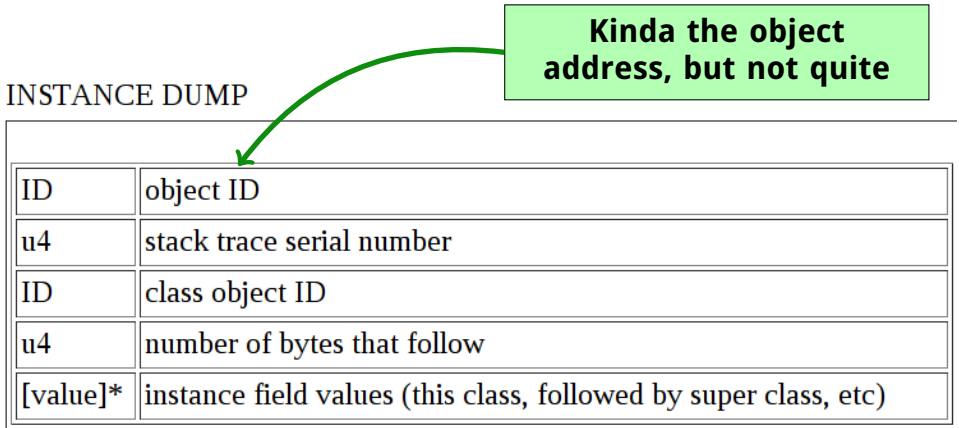
INSTANCE DUMP

ID	object ID
u4	stack trace serial number
ID	class object ID
u4	number of bytes that follow
[value]*	instance field values (this class, followed by super class, etc)

Assumes superclass fields are separate



Heap Dumps: HPROF format, #2



MXBeans: Idea

```
long tid = Thread.currentThread().getId();
var bean = (ThreadMXBean)ManagementFactory.getThreadMXBean();

long before = bean.getThreadAllocatedBytes(tid);
new Object();
long after = bean.getThreadAllocatedBytes(tid);

System.out.println("size = " + (after - before));
```

MXBeans: Testing...

```
# java Test
size = 64
```

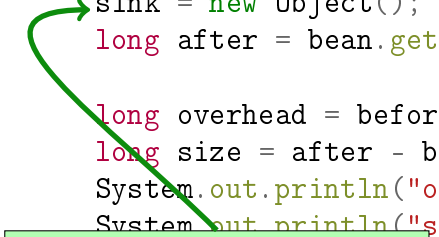
MXBeans: Testing...

```
# java Test
size = 64
```

Wait, what? A single `new Object()` is 64 bytes?

MXBeans: Better Idea

```
long tid = Thread.currentThread().getId();
var bean = (ThreadMXBean)ManagementFactory.getThreadMXBean();

long before1 = bean.getThreadAllocatedBytes(tid);
long before2 = bean.getThreadAllocatedBytes(tid);
sink = new Object();
long after = bean.getThreadAllocatedBytes(tid);

long overhead = before2 - before1;
long size = after - before2 - overhead;
System.out.println("overhead = " + overhead);
System.out.println("size = " + size);
```



MXBeans: Better Idea

Perform empty «measurement»

```
long tid = Thread.currentThread().getId();
var bean = (ThreadMXBean)ManagementFactory.getThreadMXBean();

long before1 = bean.getThreadAllocatedBytes(tid);
long before2 = bean.getThreadAllocatedBytes(tid);
sink = new Object();
long after = bean.getThreadAllocatedBytes(tid);

long overhead = before2 - before1;
long size = after - before2 - overhead;
System.out.println("overhead = " + overhead);
System.out.println("size = " + size);
```

Prevent dead code elimination

MXBeans: Better Idea

```
long tid = Thread.currentThread().getId();
var bean = (ThreadMXBean)ManagementFactory.getThreadMXBean();

long before1 = bean.getThreadAllocatedBytes(tid);
long before2 = bean.getThreadAllocatedBytes(tid);
sink = new Object();
long after = bean.getThreadAlloca

long overhead = before2 - before1;
long size = after - before2 - overhead;
System.out.println("overhead = " + overhead);
System.out.println("size = " + size);
```

Perform empty «measurement»

Subtract the overhead, *hoping* it is constant

Prevent dead code elimination

MXBeans: Overheads

```
# java Test
overhead = 48
size = 16
```

[1] https://bugs.openjdk.java.net/browse/JDK-8231209

MXBeans: Overheads

```
# java Test
overhead = 48
size = 16
```

Can be fixed:[1]

```
# jdk8u282
overhead = 0
size = 16
```

```
# jdk11.0.9
overhead = 0
size = 16
```

```
# jdk17-ea
overhead = 0
size = 16
```

[1] https://bugs.openjdk.java.net/browse/JDK-8231209

Diagnostic VM Flags: Example

Luckily, we can ask the JVM to dump the field layouts!

Diagnostic VM Flags: Example

Luckily, we can ask the JVM to dump the field layouts!

```
# jdk/bin/java -XX:+PrintFieldLayout
Error: VM option 'PrintFieldLayout' is notproduct and...
 ... is available only in debug version of VM.
Error: Could not create the Java Virtual Machine.
```

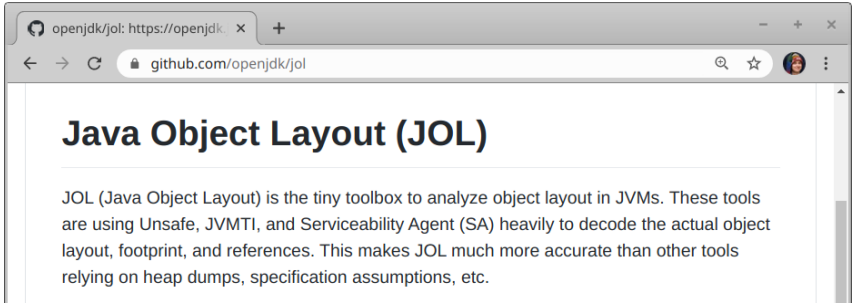
Err...

Diagnostic VM Flags: Example, Fixed

Yay, debug JVM builds work!

```
# jdk-ea.fastdebug/bin/java -XX:+PrintFieldLayout
Layout of class java/util/ResourceBundle$3
Instance fields:
 @0 12/- RESERVED
 @12 "val$providerName" Ljava/lang/String; 4/4 REGULAR
 @16 "val$loader" Ljava/lang/ClassLoader; 4/4 REGULAR
Static fields:
 @0 112/- RESERVED
Instance size = 24 bytes
```

VM-specific Tools: Java Object Layout (JOL)



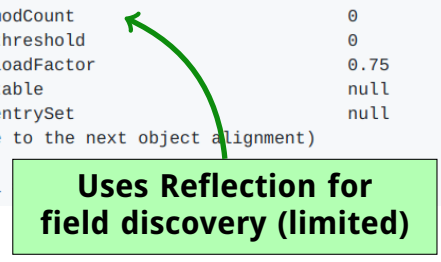
`https://github.com/openjdk/jol`

VM-specific Tools: Java Object Layout (JOL)

```
java.util.HashMap object internals:
OFFSET  SIZE                      TYPE DESCRIPTION                               VALUE
     0     4                           (object header)                           01 00 00 00 (1)
     4     4                           (object header)                           00 00 00 00 (0)
     8     4                           (object header)                           b8 b2 01 00 (1112
    12     4              java.util.Set AbstractMap.keySet                        null
    16     4       java.util.Collection AbstractMap.values                        null
    20     4                        int HashMap.size                              0
    24     4                        int HashMap.modCount                          0
    28     4                        int HashMap.threshold                         0
    32     4                      float HashMap.loadFactor                        0.75
    36     4   java.util.HashMap.Node[] HashMap.table                             null
    40     4              java.util.Set HashMap.entrySet                          null
    44     4                           (loss due to the next object alignment)
Instance size: 48 bytes
Space losses: 0 bytes internal + 4 bytes external = 4 bytes total
```

VM-specific Tools: Java Object Layout (JOL)

```
java.util.HashMap object internals:
OFFSET  SIZE                      TYPE DESCRIPTION                               VALUE
     0     4                           (object header)                           01 00 00 00 (1)
     4     4                           (object header)                           00 00 00 00 (0)
     8     4                           (object header)                           b8 b2 01 00 (1112
    12     4              java.util.Set AbstractMap.keySet                        null
    16     4       java.util.Collection AbstractMap.values                        null
    20     4                        int HashMap.size                              0
    24     4                        int HashMap.modCount                          0
    28     4                        int HashMap.threshold                         0
    32     4                      float HashMap.loadFactor                        0.75
    36     4   java.util.HashMap.Node[] HashMap.table                             null
    40     4              java.util.Set HashMap.entrySet                          null
    44     4                           (loss due to the next object alignment)
Instance size: 48 bytes
Space losses: 0 bytes internal + 4 bytes external
```



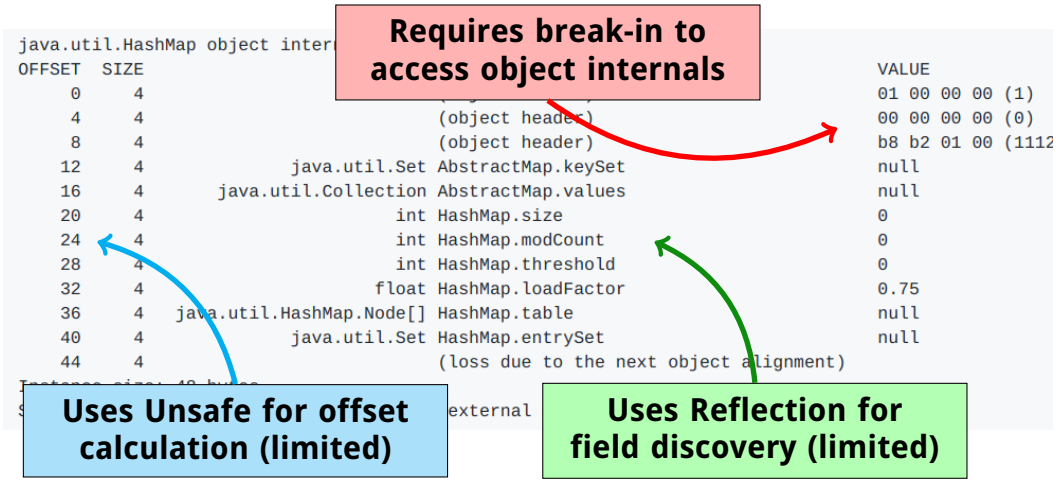
VM-specific Tools: Java Object Layout (JOL)

```
java.util.HashMap object internals:
OFFSET  SIZE                      TYPE DESCRIPTION                               VALUE
     0     4                           (object header)                           01 00 00 00 (1)
     4     4                           (object header)                           00 00 00 00 (0)
     8     4                           (object header)                           b8 b2 01 00 (1112
    12     4              java.util.Set AbstractMap.keySet                        null
    16     4       java.util.Collection AbstractMap.values                        null
    20     4                        int HashMap.size                              0
    24     4                        int HashMap.modCount                          0
    28     4                        int HashMap.threshold                         0
    32     4                      float HashMap.loadFactor                        0.75
    36     4   java.util.HashMap.Node[] HashMap.table                             null
    40     4              java.util.Set HashMap.entrySet                          null
    44     4                           (loss due to the next object alignment)
Instances size: 48 bytes
S                                                                    external
```

Uses Unsafe for offset calculation (limited)

Uses Reflection for field discovery (limited)

VM-specific Tools: Java Object Layout (JOL)



Data Type Representation

Data Type Representation: Bits Per Type

Type	Model			
	32-bit		64-bit	
	Min	Act	Min	Act
boolean	1		1	
byte	8		8	
short/char	16		16	
int/float	32		32	
long/double	64		64	
reference				

Data Type Representation: Bits Per Type

Type	Model			
	32-bit		64-bit	
	Min	Act	Min	Act
boolean	1		1	
byte	8	8	8	8
short/char	16	16	16	16
int/float	32	32	32	32
long/double	64	64	64	64
reference				

Size Exceptions: boolean

Suppose `boolean` field takes a single bit,
then `boolean` store looks like:

```
movb %r1, (loc)   # read the entire byte
...something...   # create mask for a bit
or %r1, mask      # set the 1-st bit
movb (loc), %r1   # write the byte back
```

Size Exceptions: boolean

Suppose `boolean` field takes a single bit,
then `boolean` store looks like:

```
movb %r1, (loc)   # read the entire byte
...something...   # create mask for a bit
or %r1, mask      # set the 1-st bit
movb (loc), %r1   # write the byte back
```

Hardware can only address full bytes

Size Exceptions: boolean

Suppose `boolean` field takes a single bit,
then `boolean` store looks like:

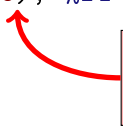
```
movb %r1, (loc)   # read the entire byte
...something...   # create mask for a bit
or %r1, mask      # set the 1-st bit
movb (loc), %r1   # write the byte back
```

Computing the mask is funky (omitted here)

Size Exceptions: boolean

Suppose `boolean` field takes a single bit,
then `boolean` store looks like:

```
movb %r1, (loc)   # read the entire byte
...something...   # create mask for a bit
or %r1, mask      # set the 1-st bit
movb (loc), %r1   # write the byte back
```



Just overwrote adjacent bits! Big NO-NO.

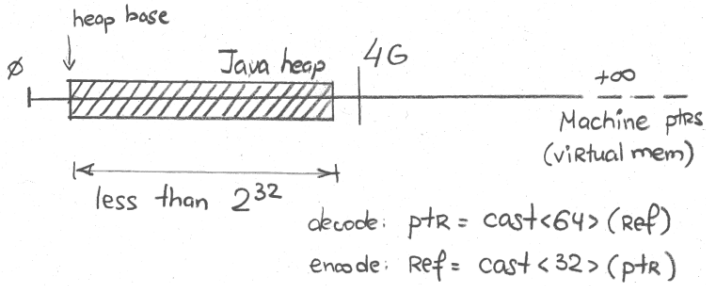
Size Exceptions: Bits Per Type

Type	Model			
	32-bit		64-bit	
	Min	Act	Min	Act
boolean	1	8	1	8
byte	8	8	8	8
short/char	16	16	16	16
int/float	32	32	32	32
long/double	64	64	64	64
reference				

Size Exceptions: Bits Per Type

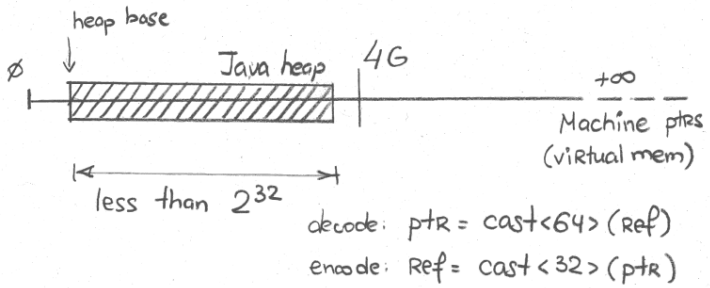
Type	Model			
	32-bit		64-bit	
	Min	Act	Min	Act
boolean	1	8	1	8
byte	8	8	8	8
short/char	16	16	16	16
int/float	32	32	32	32
long/double	64	64	64	64
reference		32		

Size Exceptions: References[2]



[2] https://shipilev.net/jvm/anatomy-quarks/23-compressed-references/

Size Exceptions: References[2]




```
# java -XX:+PrintFlagsFinal | grep CompressedOops
bool PrintCompressedOopsMode     = false   {diagnostic}
bool UseCompressedOops           := true    {lp64_product}
```

[2] https://shipilev.net/jvm/anatomy-quarks/23-compressed-references/

Size Exceptions: Compressed References, #2

```
# java
# java -Xmx8g
# java -Xmx30g

"Node" object internals:
OFF  SZ      TYPE DESCRIPTION
  0  12           ...
 12   4    Object Node.left
 16   4    Object Node.right
 20   4           ...
Instance size: 24 bytes
```

```
# java -Xmx40g
# java -XX:-UseCompressedOops
# java -XX:+UseZGC

"Node" object internals:
OFF  SZ      TYPE DESCRIPTION
  0  16           ...
 16   8    Object Node.left
 24   8    Object Node.right
Instance size: 32 bytes
```

Size Exceptions: Bits Per Type

Type	Model			
	32-bit		64-bit	
	Min	Act	Min	Act
boolean	1	8	1	8
byte	8	8	8	8
short/char	16	16	16	16
int/float	32	32	32	32
long/double	64	64	64	64
reference		32		32/64

Array Element Sizes: Another Wrinkle

```
# java -jar jol-cli.jar ...

# Running 64-bit HotSpot VM
# Field sizes by type: 4, 1, 1, 2, 2, 4, 4, 8, 8 [bytes]
# Array element sizes: 4, 1, 1, 2, 2, 4, 4, 8, 8 [bytes]
```

Nothing requires array element sizes to match field sizes. In Hotspot JVMs, they generally match.

Header, Mark Word

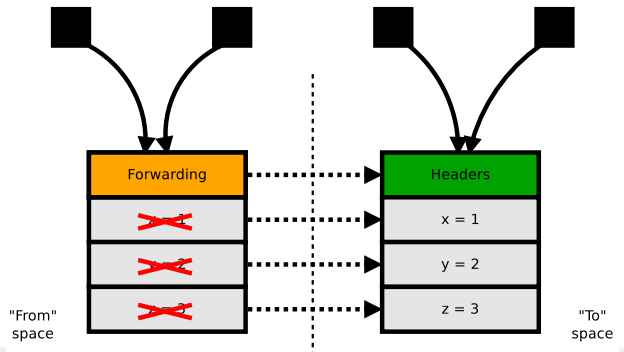
Header, Mark Word: Simple Object

```
# jdk8-64/java -jar jol-cli.jar internals java.lang.Object
# Running 64-bit HotSpot VM.

java.lang.Object object internals:
OFF  SZ   TYPE  DESCRIPTION       VALUE
  0   4         (object header)   05 00 00 00 # Mark word
  4   4         (object header)   00 00 00 00 # Mark word
  8   4         (object header)   00 10 00 00 # Class word
 12   4         (object alignment loss)
Instance size: 16 bytes
```

- Every Java object carriers a header: *mark* and *class* words
- Object header bits are very precious resource

GC Data: Marking/Forwarding

When GC moves the object, it needs to record a new location:



mark word of «old» object is as good place as any

GC Data: Observation

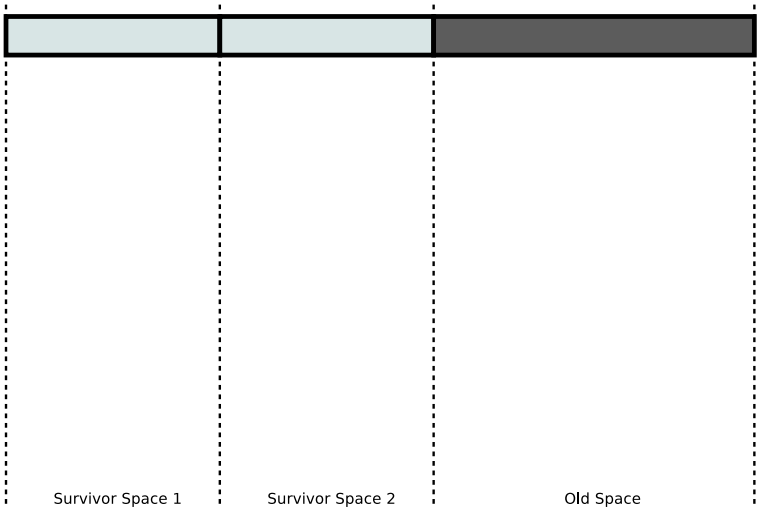
Opaque to application code, only VM sees it:

```
#  Internal Error (/home/shade/trunks/jdk/src/hotspot/share/gc/shenandoah/shenandoahV
#  Error: After Updating References, Reachable; Object should be in active region

Object:
  0x00000000e81d7ec0 - klass 0x000000080008e270 java.lang.invoke.MethodType
    not allocated after mark start
    not after update watermark
        marked strong
    not marked weak
    not in collection set
  mark: marked(0x00000000fffbfddb)
  region: |    7|T  |BTE     e81c0000,     e8200000,     e8200000|TAMS     e8200000|U

Forwardee:
  0x00000000fffbfdd8 - safe print, no details
  region: | 1534|R  |BTE     fff80000,     fffc0000,     fffc0000|TAMS     fff80000|U
```

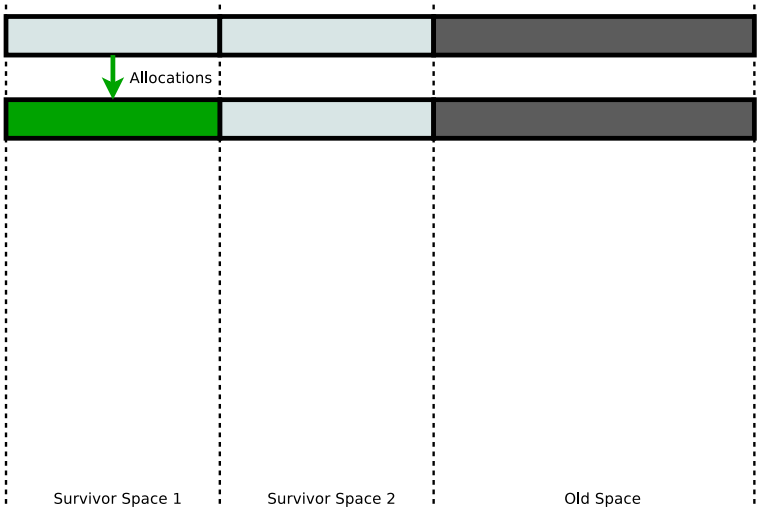
GC Data: 32-bit VMs Improve Footprint

```
# Running 32-bit HotSpot VM
java.lang.Object object internals:
OFF  SZ   TYPE DESCRIPTION     VALUE
  0   4        (object header) ...   # Mark word
  4   4        (object header) ...   # Class word
Instance size: 8 bytes

# Running 64-bit HotSpot VM
java.lang.Object object internals:
OFF  SZ   TYPE DESCRIPTION     VALUE
  0   8        (object header) ...   # Mark word
  8   4        (object header) ...   # Class word
 12   4        (object alignment loss)
Instance size: 16 bytes
```

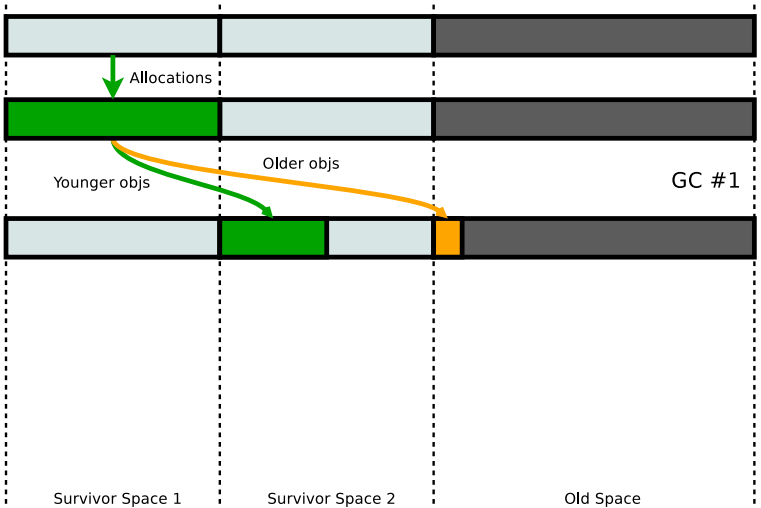
Object Ages: Object Lifecycle



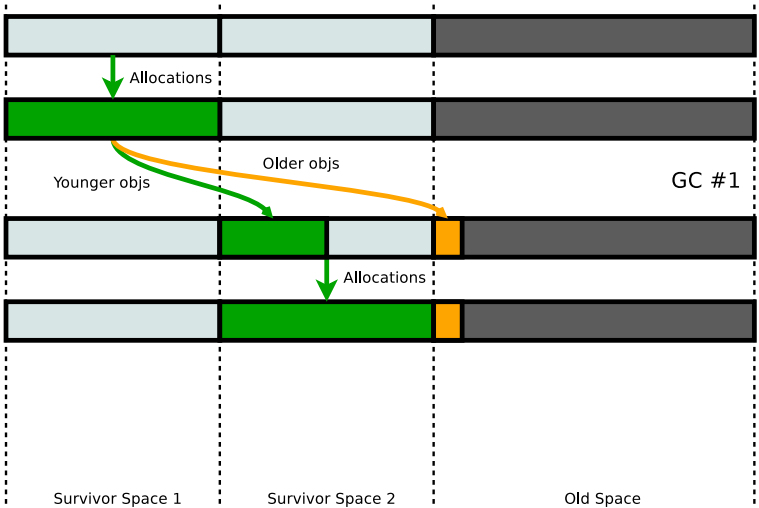
Object Ages: Object Lifecycle



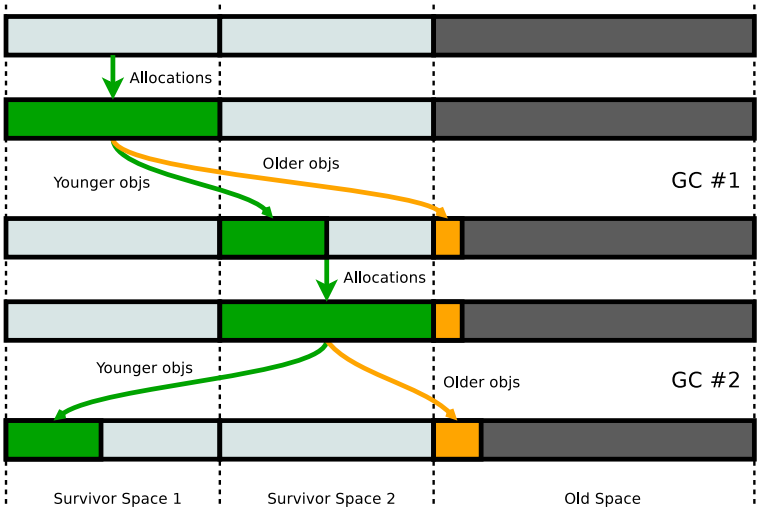
Object Ages: Object Lifecycle



Object Ages: Object Lifecycle



Object Ages: Object Lifecycle



Object Ages: Observation

```
# jdk8-32/bin/java -cp jol-samples.jar \
#    org.openjdk.jol.samples.JOLSample_19_Promotion

# Running 32-bit HotSpot VM

Move 1: (object header)  09 00 00 00 (00001001 ...)
Move 2: (object header)  11 00 00 00 (00010001 ...)
Move 3: (object header)  19 00 00 00 (00011001 ...)
Move 4: (object header)  21 00 00 00 (00100001 ...)
Move 5: (object header)  29 00 00 00 (00101001 ...)
Move 6: (object header)  31 00 00 00 (00110001 ...)
Move 7: (object header)  31 00 00 00 (00110001 ...)
# <moves stop, object in old>
```

Object Ages: Mark Word

- Current budget is 4 bits $\Rightarrow$ age $\in [0; 2^4)$

```
# java -XX:+PrintFlagsFinal | grep Tenuring
uintx InitialTenuringThreshold  = 7  {product}
uintx MaxTenuringThreshold      = 15 {product}
```

- No matter how hard you try...

```
$ java -XX:MaxTenuringThreshold=30
MaxTenuringThreshold of 30 is invalid; must be between 0 and 15
Error: Could not create the Java Virtual Machine.
```

Identity Hash Code: Problem

Every Java object implements `hashCode`,
even `java.lang.Object`!

- Hashcode **must** be idempotent
  `java.lang.Object`: okay, that one is easy
- Hashcode *should* be well-distributed
  `java.lang.Object`: err… well…

Identity Hash Code: Approaches

Approach	Idempotent?	Well-distributed?
Constant	YES	NO

Identity Hash Code: Approaches

Approach	Idempotent?	Well-distributed?
Constant	YES	NO
Counter	No	No

Identity Hash Code: Approaches

Approach	Idempotent?	Well-distributed?
Constant	YES	NO
Counter	No	No
Object address	No, not really	No, not really

Identity Hash Code: Approaches

Approach	Idempotent?	Well-distributed?
Constant	YES	NO
Counter	No	No
Object address	No, not really	No, not really
Random value	No	Yes, good PRNG

Identity Hash Code: Approaches

Approach	Idempotent?	Well-distributed?	Mode
Constant	YES	NO	2
Counter	No	No	3
Object address	No, not really	No, not really	1, 4
Random value	No	Yes, good PRNG	0, 5

Hotspot VM actually implements all of them,
selectable with `-XX:hashCode=#`

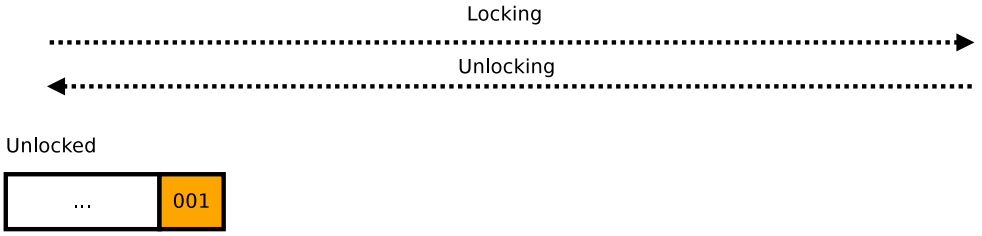
Identity Hash Code: Need to Store It

```
# Running 64-bit HotSpot VM.

"a" object internals:
OFF  SZ   TYPE DESCRIPTION         VALUE
  0   4        (object header)     01 00 00 00
  4   4        (object header)     00 00 00 00
               ...

System.identityHashCode(a): 5ccddd20

"a" object internals:
OFF  SZ   TYPE DESCRIPTION         VALUE
  0   4        (object header)     01 20 dd cd
  4   4        (object header)     5c 00 00 00
               ...
```

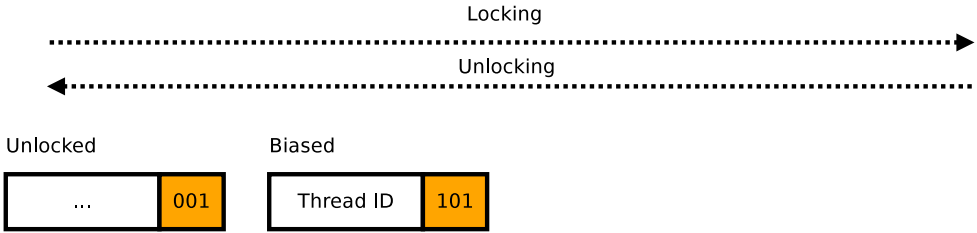
Identity Hash Code: Need to Store (Some Of) It

```
# Running 32-bit HotSpot VM.

"a" object internals:
OFF  SZ   TYPE DESCRIPTION         VALUE
  0   4        (object header)     01 00 00 00
              ...

System.identityHashCode(a): 12ddf17

"a" object internals:
OFF  SZ   TYPE DESCRIPTION         VALUE
  0   4        (object header)     81 8b ef 96
              ...
```

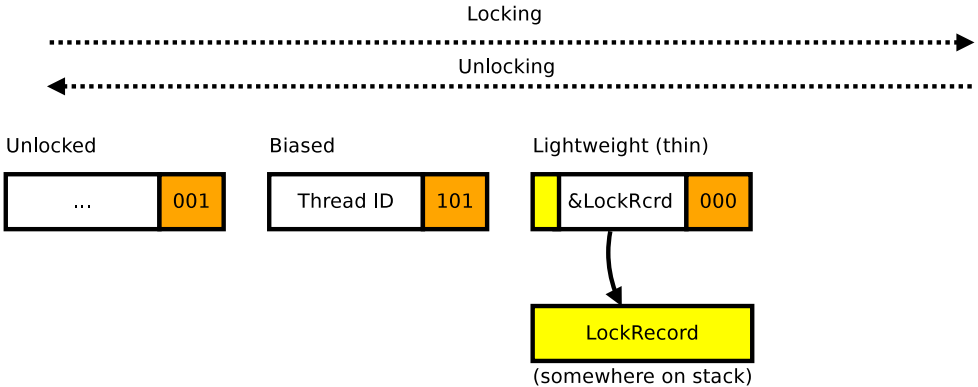
Locking: Hierarchy



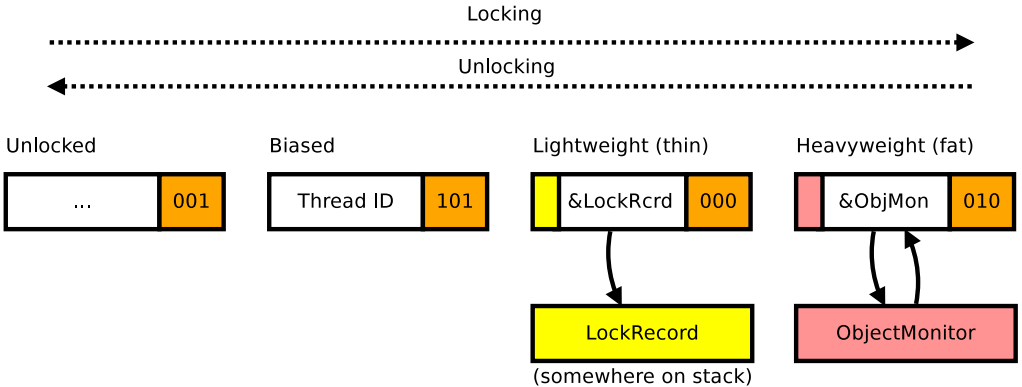
Locking: Hierarchy



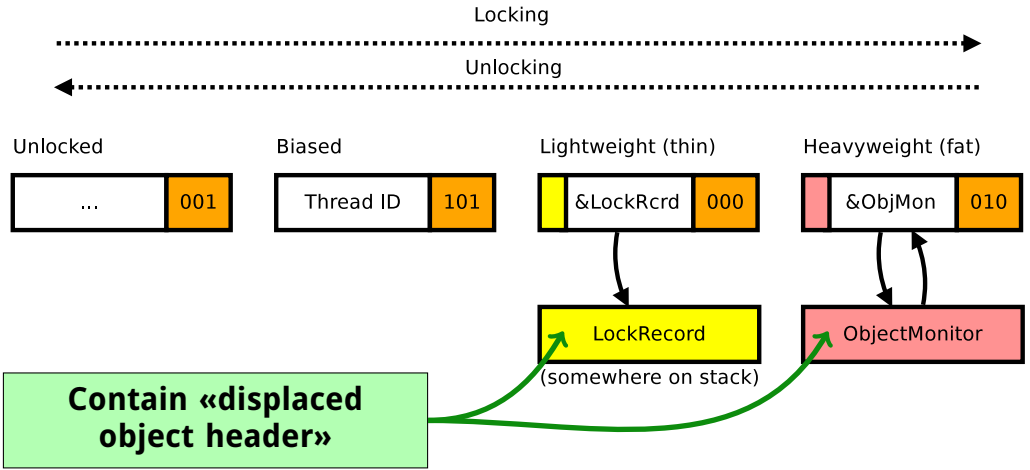
Locking: Hierarchy



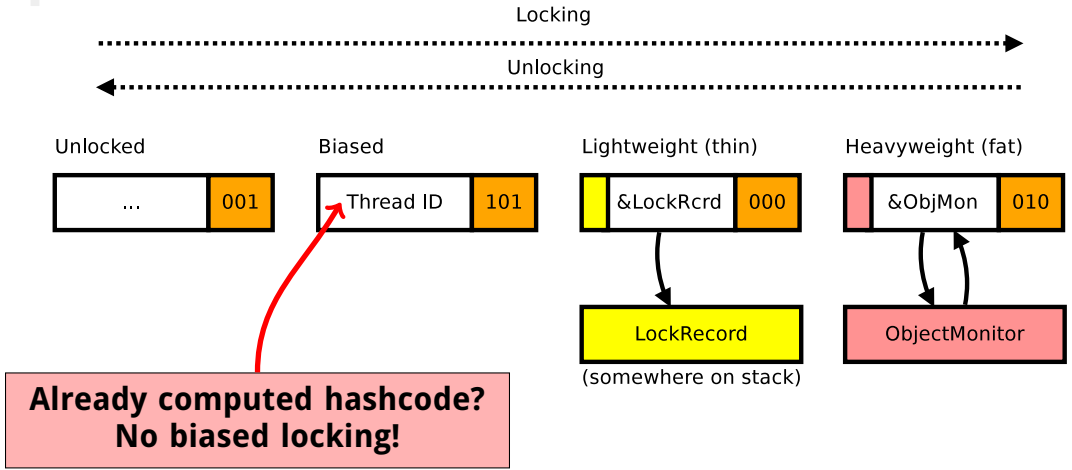
Locking: Hierarchy



Locking: Hierarchy



Locking: Hierarchy



Locking: Identity Hash Disables Biased Locking

```
@Setup
public void setup(Blackhole bh) {
  if (hc) bh.consume(obj.hashCode());
}

@Benchmark
public int test() {
  synchronized (obj) { return 42; }
}
```

Benchmark	hc	Score, ns/op
BiasedFailsIHC.test	true	9.628 ± 0.009
BiasedFailsIHC.test	false	1.735 ± 0.009

Locking: Locking Slows Down Identity Hash

```
@Setup
public void setup() {
  if (sync) {
    Runnable r = () -> { synchronized (obj) { obj.wait(); }};
    new Thread(r).start();
  }
}

@Benchmark
public int test() {  return obj.hashCode(); }
```

Benchmark	sync	Score, ns/op	
SyncSlowsIHC.test	true	22.914	± 0.538
SyncSlowsIHC.test	false	1.425	± 0.024

Class Word

Class Word: Here It Is!

```
# Running 64-bit HotSpot VM
java.lang.Object object internals:
OFF  SZ   TYPE DESCRIPTION     VALUE
  0   8        (object header) ...           # Mark word
  8   4        (object header) 00 10 00 00   # Class word
 12   4        (object alignment loss)
Instance size: 16 bytes
```

- Points to **native** `Klass` instance
- Used to poll runtime type information, determining the object size, targets of virtual/interface calls

Runtime Type Info: Example Typecheck

```
Object o = new MyClass();

@Benchmark
public Class<?> test() {
  return (MyClass)o;
}
```

```
mov    0x10(%r1), %r2 ; getfield "o"
mov     0x8(%r2), %r3 ; get o.<classword>, Klass*
movabs $constant, %r4 ; load known Klass* for MyClass
cmp    %r3, %r4       ; type check
jne    SLOWPATH       ; type check failed
... r2 is definitely instance of MyClass
```

Runtime Type Info: Object::getClass Example

```
Object o = new MyClass();

@Benchmark
public Class<?> test() {
  return o.getClass();
}
```

```
mov 0x10(%r1), %r2 ; getfield "o"
mov  0x8(%r2), %r3 ; get o.<classword>, Klass*
mov 0x70(%r3), %r4 ; get Klass._java_mirror, OopHandle
mov     (%r4), %r5 ; deref OopHandle, java.lang.Class
```

Determining The Object Size: GC example

GCs frequently traverse «self-parsable heap»[3] like this:

```
object cur = heap_start;
while (cur < heap_used) {
  do_object(cur);
  cur = cur + cur->size();
}
```

[3] https://shipilev.net/jvm/anatomy-quarks/5-tlabs-and-heap-parsability/

Determining The Object Size: GC example

GCs frequently traverse «self-parsable heap»[3] like this:

```
object cur = heap_start;
while (cur < heap_used) {
  do_object(cur);
  cur = cur + cur->size();
}
```

...where:

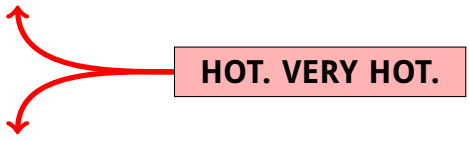
```
int object::size() {
  int lh = _klass->_layout_helper;
  return ((lh & 0x1) == 0) ? lh : decode(lh);
}
```

[3] https://shipilev.net/jvm/anatomy-quarks/5-tlabs-and-heap-parsability/

Determining The Object Size: GC example

GCs frequently traverse «self-parsable heap»[3] like this:

```
object cur = heap_start;
while (cur < heap_used) {
  do_object(cur);
  cur = cur + cur->size();
}
```



...where:

```
int object::size() {
  int lh = _klass->_layout_helper;
  return ((lh & 0x1) == 0) ? lh : decode(lh);
}
```

[3] https://shipilev.net/jvm/anatomy-quarks/5-tlabs-and-heap-parsability/

Compressed Class Ptrs: Here It Is!

```
# Running 64-bit HotSpot VM
java.lang.Object object internals:
OFF  SZ   TYPE DESCRIPTION     VALUE
  0   8        (object header) ...           # Mark word
  8   4        (object header) 00 10 00 00   # Class word
 12   4        (object alignment loss)
Instance size: 16 bytes
```

- Points to **native** `Klass` instance
- With a few encoding tricks, can be **compressed**

Compressed Class Ptrs: Dependencies

```
# jdk11u/bin/java -XX:+UseCompressedOops ...
bool UseCompressedOops           = true
bool UseCompressedClassPointers  = true

# jdk11u/bin/java -XX:-UseCompressedOops ...
bool UseCompressedOops           = false
bool UseCompressedClassPointers  = false
```

Compressed Class Ptrs: Dependencies

```
# jdk11u/bin/java -XX:+UseCompressedOops ...
bool UseCompressedOops           = true
bool UseCompressedClassPointers  = true

# jdk11u/bin/java -XX:-UseCompressedOops ...
bool UseCompressedOops           = false
bool UseCompressedClassPointers  = false
```

Compressed Class Ptrs: Dependencies, #2

```
# java
# java -Xmx8g
# java -Xmx30g

j.l.Integer object internals:
OFF  SZ  TYPE DESCRIPTION
 0   8       (mark word)
 8   4       (class word)
12   4   int Integer.value
Instance size: 16 bytes
```

```
# java -Xmx40g
# java -XX:-UseCompressedOops
# java -XX:+UseZGC

j.l.Integer object internals:
OFF  SZ  TYPE DESCRIPTION
  0   8       (mark word)
  8   8       (class word)
 16   4   int Integer.value
 20   4       (alignment loss)
Instance size: 24 bytes
```

Compressed Class Ptrs: JDK 15+

```
# jdk-ea/bin/java -XX:-UseCompressedOops ...
bool UseCompressedOops           = false
bool UseCompressedClassPointers  = true
```

This is actually platform-dependent, but definitely works on x86_64, AArch64, POWER and others...[4]

[4] https://bugs.openjdk.java.net/browse/JDK-8241825

Compressed Class Ptrs: JDK 15+, #2

```
# java
# java -Xmx8g
# java -Xmx30g

j.l.Integer object internals:
OFF  SZ   TYPE DESCRIPTION
  0   8        (mark word)
  8   4        (class word)
 12   4    int Integer.value
Instance size: 16 bytes
```

```
# java -Xmx40g
# java -XX:-UseCompressedOops
# java -XX:+UseZGC

j.l.Integer object internals:
OFF  SZ   TYPE DESCRIPTION
  0   8        (mark word)
  8   4        (class word)
 12   4    int Integer.value
Instance size: 16 bytes
```

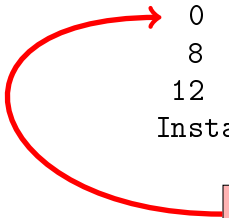
Object Alignment

Object Alignment: Hardware Wrinkles Again

```
# Running 64-bit HotSpot VM.
# Objects are 8 bytes aligned.

java.lang.Object object internals:
OFF  SZ   TYPE  DESCRIPTION       VALUE
  0   8         (object header)   ... # Mark word
  8   4         (object header)   ... # Class word
 12   4         (object alignment loss)
Instance size: 16 bytes
```

Object Alignment: Hardware Wrinkles Again

```
# Running 64-bit HotSpot VM.
# Objects are 8 bytes aligned.

java.lang.Object object internals:
OFF  SZ   TYPE  DESCRIPTION       VALUE
  0   8         (object header)   ... # Mark word
  8   4         (object header)   ... # Class word
 12   4         (object alignment loss)
Instance size: 16 bytes
```



WARNING: Atomic access required

Object Alignment: Hardware Wrinkles Again

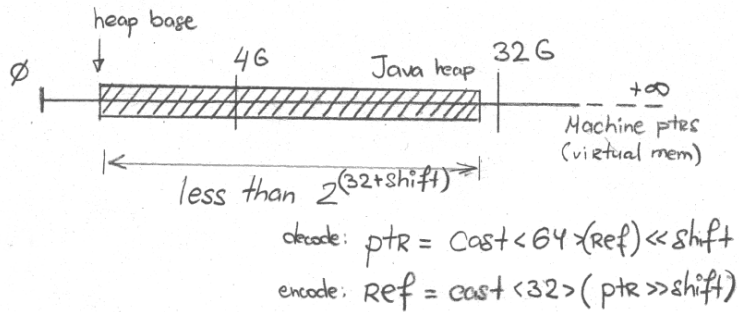
```
# Running 64-bit HotSpot VM.
# Objects are 8 bytes aligned.

java.lang.Object object internals:
OFF  SZ   TYPE  DESCRIPTION       VALUE
  0   8         (object header)   ... # Mark word
  8   4         (object header)   ... # Class word
 12   4         (object alignment loss)
Instance size: 16 bytes
```

Object must be aligned by mark word size

WARNING: Atomic access required

Object Alignment: Comp Refs, Larger Heaps[5]

Non-zero alignment means that
lower reference bits are always zero.



[5] https://shipilev.net/jvm/anatomy-quarks/24-object-alignment/

Object Alignment: Hiding Fields

```
# Running 64-bit HotSpot VM.
# Objects are 8 bytes aligned.

java.lang.Object object internals:
OFF  SZ   TYPE DESCRIPTION        VALUE
  0  12        (object header)    ...
 12   4        (object alignment loss)
Instance size: 16 bytes

java.lang.Integer object internals:
OFF  SZ   TYPE DESCRIPTION        VALUE
  0  12        (object header)    ...
 12   4    int Integer.value      0
Instance size: 16 bytes
```

Object Alignment: Small Fields Blowup

```
public class A {
  int a1;
}

public class B {
  int b1;
  boolean b2; // takes 1 byte?
}
```

Object Alignment: Small Fields Blowup, #2

```
A object internals:
OFF  SZ      TYPE DESCRIPTION       VALUE
  0  12           (object header)   ...
 12   4       int A.a1              0
Instance size: 16 bytes

B object internals:
OFF  SZ      TYPE DESCRIPTION       VALUE
  0  12           (object header)   ...
 12   4       int B.b1              0
 16   1   boolean B.b2              false
 17   7           (object alignment loss)
Instance size: 24 bytes
```

Field Alignments

Field Alignments: `java.lang.Long` Example

```
# Running 64-bit HotSpot VM.
# Objects are 8 bytes aligned.

java.lang.Long object internals:
OFF  SZ   TYPE DESCRIPTION               VALUE
  0  12        (object header)           ...
 12   4        (alignment/padding gap)
 16   8   long Long.value                0
Instance size: 24 bytes
```

The field needs to be aligned by 8 itself

Field Alignments: Hiding Fields

```
public class LongIntCarrier {
  long value;
  int ninja;
}
```

```
LongIntCarrier object internals:
OFF  SZ   TYPE DESCRIPTION              VALUE
  0  12        (object header)         ...
 12   4    int LongIntCarrier.ninja    0
 16   8   long LongIntCarrier.value    0
Instance size: 24 bytes
```

Arrays

Arrays: Additional Headers

Array length is not a part of data type,
so it needs a separate header:

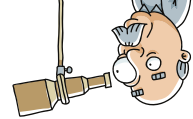
```
"new byte[8]" object internals:
OFF   SZ   TYPE DESCRIPTION       VALUE
  0    4        (object header)   ...           # Mark word
  4    4        (object header)   ...           # Mark word
  8    4        (object header)   ...           # Class word
 12    4        (object header)   08 00 00 00   # Array length
 16    8   byte [B.<elements>     N/A
Instance size: 24 bytes
```

Arrays: Bounds Checks

```
private int[] a = new int[100];

@Benchmark
public int test() {
  return a[42];
}
```

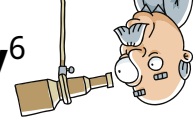
```
mov   0x10(%r1),%r2    ; get field "a"
mov   0x10(%r2),%r3    ; get a.<arraylength>, at 0x10
cmp   $0x2a,%r3        ; compare arraylength with 42
jbe   SLOWPATH         ; below or equal? jump to slowpath
mov   0xc0(%r2),%r4    ; read element at (24 + 4*42) = 0xc0
```

Arrays: Array Base Is Aligned



```
# java -Xmx40g

"new long[1]" object internals:
OFF  SZ   TYPE DESCRIPTION       VALUE
  0   8        (object header)   ...           # Mark word
  8   8        (object header)   ...           # Class word
 16   4        (object header)   01 00 00 00   # Array length
 20   4        (alignment/padding gap)
 24   8   long [J.<elements>     ...
Instance size: 32 bytes
```

Arrays: Array Base Is Aligned Unnecessarily[6]



```
# java -Xmx40g

"new byte[8]" object internals:
OFF  SZ   TYPE DESCRIPTION       VALUE
  0   8        (object header)   ...           # Mark word
  8   8        (object header)   ...           # Class word
 16   4        (object header)   08 00 00 00   # Array length
 20   4        (alignment/padding gap)
 24   8   byte [B.<elements>     ...
Instance size: 32 bytes
```

[6] Must be a bug: `https://bugs.openjdk.java.net/browse/JDK-8139457`

Arrays: Small Arrays Are Not Small

```
"new A()" object internals:
OFF  SZ   TYPE DESCRIPTION
  0  12        (object header)  # Mark + Class
 12   4    int A.a
Instance size: 16 bytes

"new int[1]" object internals:
OFF  SZ   TYPE DESCRIPTION
  0  12        (object header)  # Mark + Class
 12   4        (object header)  # Array length
 16   4    int [I.<elements>
 20   4        (object alignment loss)
Instance size: 24 bytes
```

Field Packing

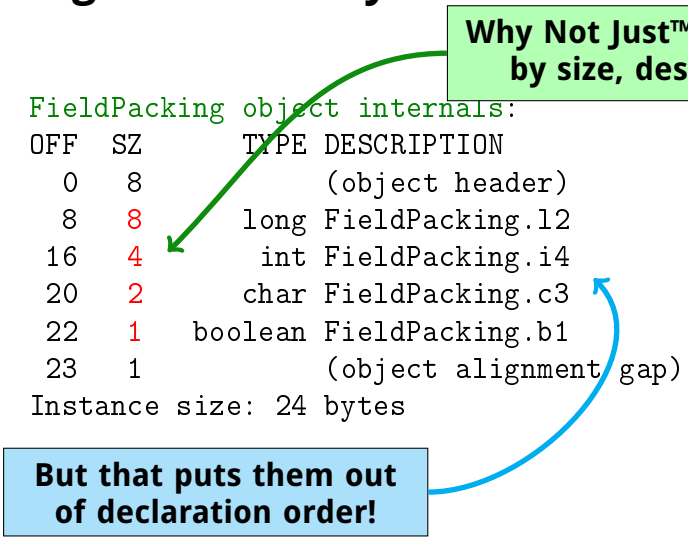
Field Packing: Consider A Class

```
public class FieldPacking {
  boolean b1;
  long    l2;
  char    c3;
  int     i4;
}
```

Field Packing: Naive Layout

```
FieldPacking object internals:
OFF  SZ      TYPE DESCRIPTION
  0   8           (object header)
  8   1   boolean FieldPacking.b1
  9   7           (alignment/padding gap)
 16   8      long FieldPacking.l2
 24   2      char FieldPacking.c3
 26   2           (alignment/padding gap)
 28   4       int FieldPacking.i4
Instance size: 32 bytes
```

Field Packing: Smarter Layout



Field Packing: C-style Padding Is Unreliable

```
public class LongPadding {
  long l01, l02, l03, l04, l05, l06, l07, l08; // 64 bytes
  byte pleaseHelpMe;
  long l11, l12, l13, l14, l15, l16, l17, l18; // 64 bytes
}
```

For example, to avoid «false sharing» on adjacent fields...

Field Packing: C-style Padding Is Unreliable, #2

```
LongPadding object internals:
OFF  SZ    TYPE DESCRIPTION
  0  12         (object header)
 12   1    byte LongPadding.pleaseHelpMe
 13   3         (alignment/padding gap)
 16   8    long LongPadding.l01
                ...
136   8    long LongPadding.l18
Instance size: 144 bytes
```

That’s optimization! The one we did not want…

Field Packing: C-style Padding, Trying Again

```
public class BytePadding {
  byte p000, p001, p002, p003, p004, p005, p006, p007;
  ...
  byte p056, p057, p058, p059, p060, p061,  , p063;
  byte pleaseHelpMe;
  byte p100, p101, p102, p103, p104, p105, p106, p107;
  ...
  byte p156, p157, p158, p159, p160, p161, p162, p163;
}
```

Field Packing: C-style Padding, Trying Again, #2

```
BytePadding object internals:
OFF  SZ    TYPE DESCRIPTION
  0  12         (object header)
...
 75   1    byte BytePadding.p063
 76   1    byte BytePadding.pleaseHelpMe
 77   1    byte BytePadding.p100
...
141   3         (object alignment loss)
Instance size: 144 bytes
```

«Works», but still relies heavily on layouter

Field Packing: @Contended

```
public class Thread implements Runnable {
  ...
  @jdk.internal.vm.annotation.Contended("tlr")
  long threadLocalRandomSeed;

  @jdk.internal.vm.annotation.Contended("tlr")
  int threadLocalRandomProbe;

  @jdk.internal.vm.annotation.Contended("tlr")
  int threadLocalRandomSecondarySeed;
  ...
}
```

Field Packing: @Contended, #2

```
java.lang.Thread object internals:
OFF  SIZE      TYPE DESCRIPTION
  0    12           (object header)
...
100     4  j.l.UEH Thread.uncaughtExceptionHandler
104   128           (alignment/padding gap)
232     8      long Thread.threadLocalRandomSeed
240     4       int Thread.threadLocalRandomProbe
244     4       int Thread.threadLocalRandomSecondarySeed
248   128           (object alignment loss)
Instance size: 376 bytes
```

Layouter knows these gaps should be there

Superclass Gaps: What Are Expected Sizes?

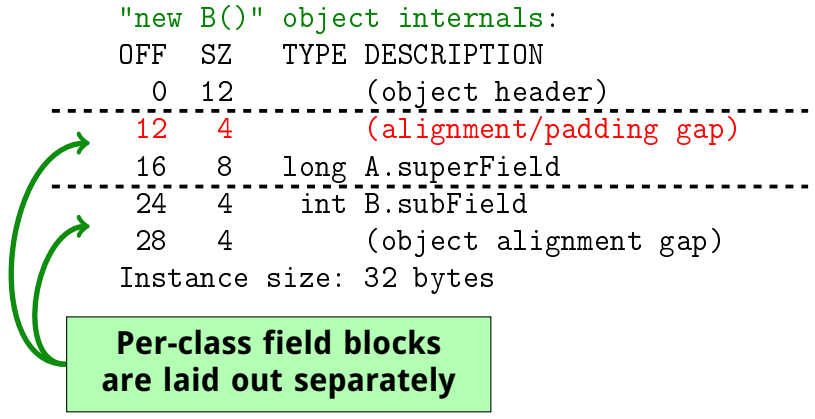
```
class T {
  long sameField1;
  int  sameField2;
}
```

```
class A {
  long superField;
}

class B extends A {
  int subField;
}
```

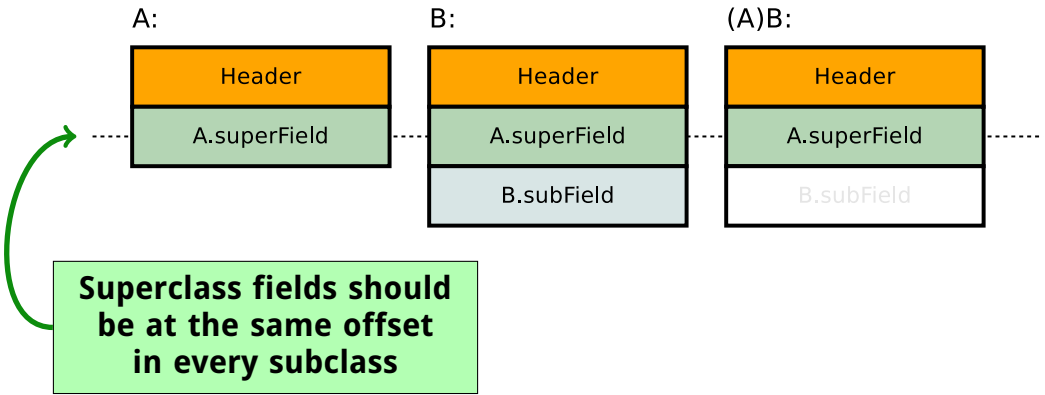
Superclass Gaps: Awwww...

```
"new B()" object internals:
OFF  SZ   TYPE DESCRIPTION
  0  12        (object header)
 12   4        (alignment/padding gap)
 16   8   long A.superField
 24   4    int B.subField
 28   4        (object alignment gap)
Instance size: 32 bytes
```

Superclass Gaps: Awwww...



Superclass Gaps: But Why?



Superclass Gaps: Hierarchy Padding Trick

```
class Squeeze1 {
  long l01, l02, l03, l04, l05, l06, l07, l08;
}

class Carrier extends Squeeze1 {
  short stocks;
}

class Squeeze2 extends Carrier {
  long l11, l12, l13, l14, l15, l16, l17, l18;
}

class PublicType extends Squeeze2 {};
```

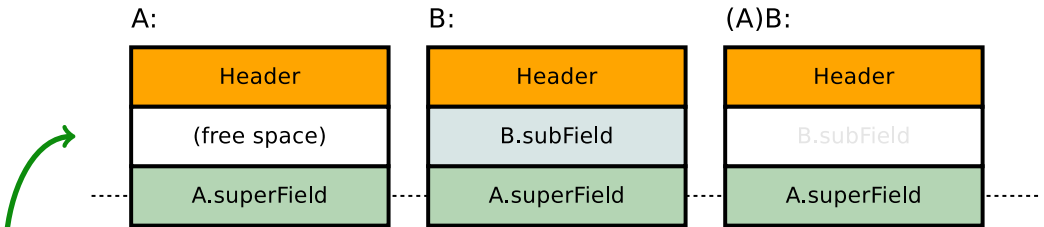
Superclass Gaps: Bug ⇒ Feature

```
"new PublicType()" object internals:
OFF  SZ   TYPE DESCRIPTION
  0  12        (object header)
 12   4        (alignment/padding gap)
 16   8   long Squeeze1.l01
                ...
 72   8   long Squeeze1.l08
 80   2  short Carrier.stocks
 82   6        (alignment/padding gap)
 88   8   long Squeeze2.l11
                ...
144   8   long Squeeze2.l18
Instance size: 152 bytes
```

Superclass Gaps: Bug ⇒ Feature

```
"new PublicType()" object internals:
OFF  SZ   TYPE DESCRIPTION
  0  12        (object header)
 12   4        (alignment/padding gap)
 16   8   long Squeeze1.l01
                ...
 72   8   long Squeeze1.l08
 80   2  short Carrier.stocks
 82   6        (alignment/padding gap)
 88   8   long Squeeze2.l11
                ...
144   8   long Squeeze2.l18
Instance size: 152 bytes
```

Superclass Gaps: But Why Not Better?

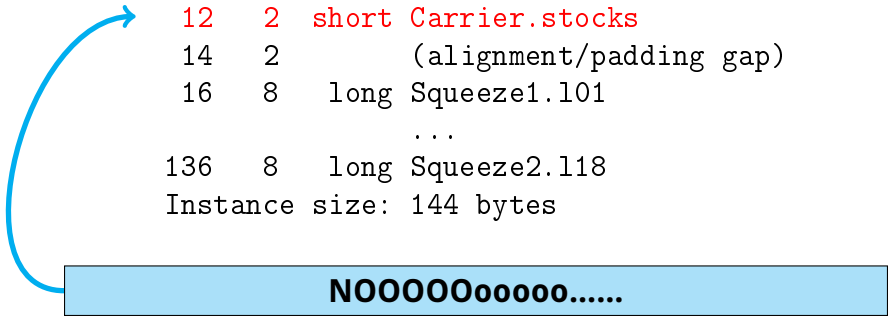


Superclass Gaps: Re-done in JDK 15+

```
"new B()" object internals:
OFF  SZ   TYPE DESCRIPTION
  0  12        (object header)
 12   4    int B.subField
 16   8   long A.superField
Instance size: 24 bytes
```

Superclass Gaps: Hierarchy Padding Collapse

```
"new PublicType()" object internals:
OFF  SZ   TYPE DESCRIPTION
  0  12        (object header)
 12   2  short Carrier.stocks
 14   2        (alignment/padding gap)
 16   8   long Squeeze1.l01
                ...
136   8   long Squeeze2.l18
Instance size: 144 bytes
```



Superclass Gaps: Hierarchy Padding Revisited

```
class Squeeze1 {
  byte p000, p001, p002, p003, p004...
}

class Carrier extends Squeeze1 {
  short stocks;
}

class Squeeze2 extends Ca
  byte p100, p101, p102, p103, p104...
}

class PublicType extends Squeeze2 {};
```

Types with smallest allocation size

Superclass Gaps: Hierarchy Padding Revisited, #2

```
"new PublicType()" object internals:
OFF  SZ   TYPE DESCRIPTION
  0  12        (object header)
                ...
 75   1   byte Squeeze1.p063
 76   2  short Carrier.stocks
 78   1   byte Squeeze2.p100
                ...
142   2        (object alignment loss)
Instance size: 144 bytes
```

Still «works», but still relies on layouter specifics

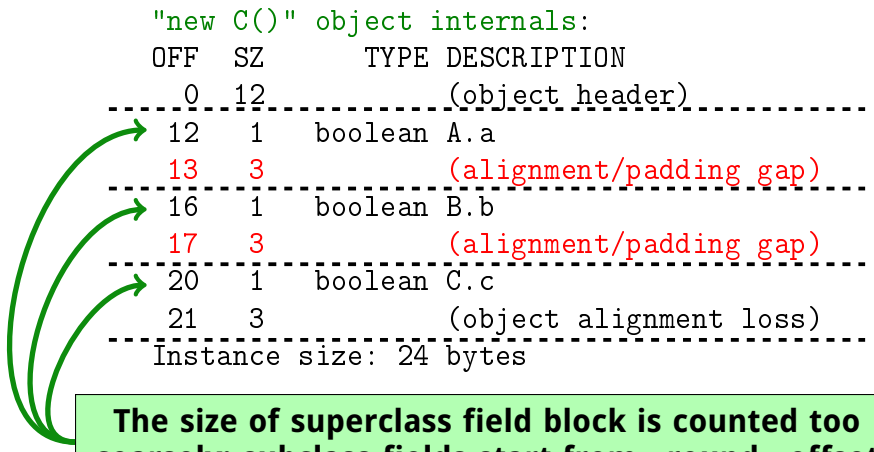
Hierarchy Gaps: What Is A Size Of «C»?

```
static class A {
  boolean a;
}

static class B extends A {
  boolean b;
}

static class C extends B {
  boolean c;
}
```

Hierarchy Gaps: Gaps, Gaps, Gaps

```
"new C()" object internals:
OFF  SZ      TYPE DESCRIPTION
  0  12           (object header)
 12   1   boolean A.a
 13   3           (alignment/padding gap)
 16   1   boolean B.b
 17   3           (alignment/padding gap)
 20   1   boolean C.c
 21   3           (object alignment loss)
Instance size: 24 bytes
```

Hierarchy Gaps: Gaps, Gaps, Gaps



The size of superclass field block is counted too coarsely: subclass fields start from «round» offset

Hierarchy Gaps: Fixed in JDK 15+

```
"new C()" object internals:
OFF  SZ      TYPE DESCRIPTION
  0  12           (object header)
 12   1   boolean A.a
 13   1   boolean B.b
 14   1   boolean C.c
 15   1           (object alignment loss)
Instance size: 16 bytes
```

Conclusion

Conclusion: Object Sizes Vary A Lot

- JVM bitness: *32- or 64-bit JVM?*
- JVM heap size: *compressed oops enabled, or not?*
- JVM features: *compressed oops implemented, or not?*
- JVM minor updates: *different metadata sizes?*
- JVM major updates: *improved field layout?*

Corollary: Important to use runtime tools to check what is going on for your **target** VM

Conclusion: Tools

- JOL (from OpenJDK Code Tools)
  `https://openjdk.java.net/projects/code-tools/jol/`
- JAMM (heavily used by Apache Cassandra)
  `https://github.com/jbellis/jamm`
- EhCache sizeOf
  `https://github.com/ehcache/sizeof`

...and if it would ever be done, JVM support for `sizeOf`:
`https://openjdk.java.net/jeps/8249196`