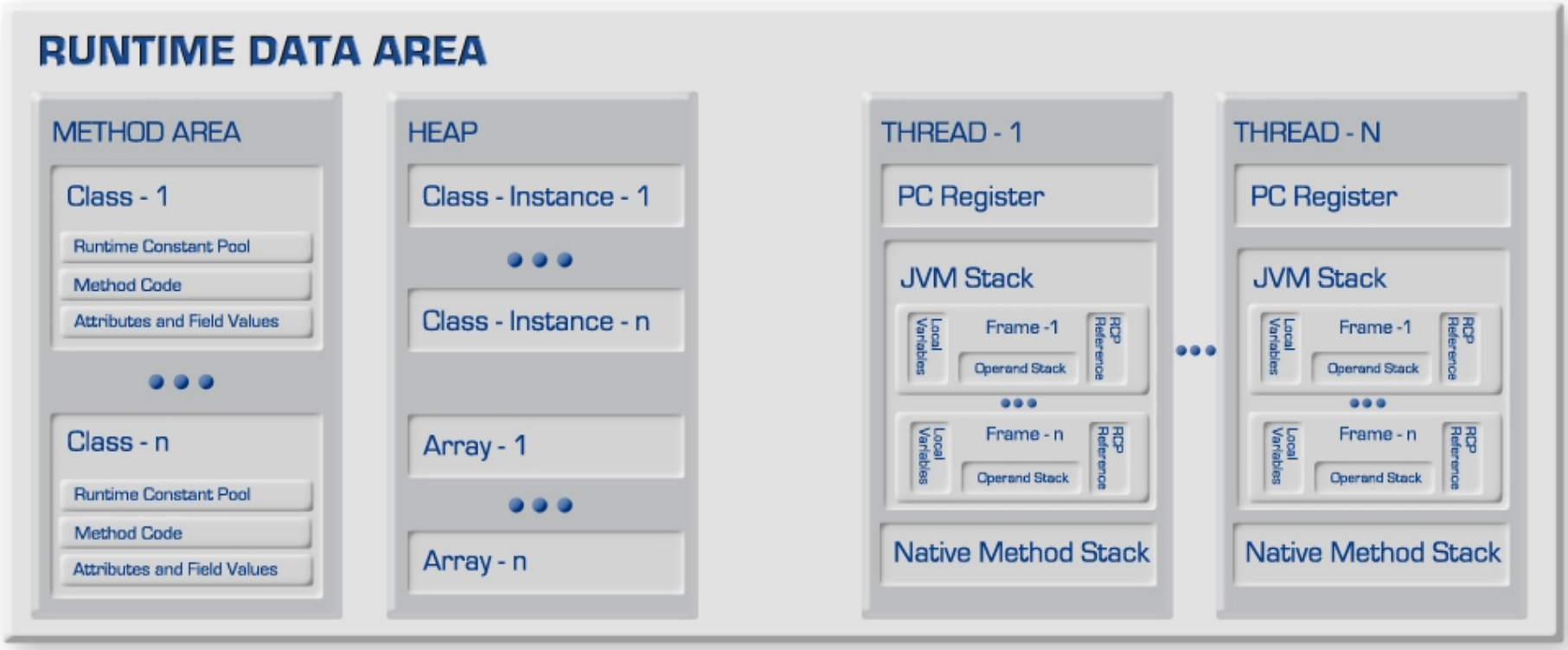
Java memory management and memory leaks

Mateusz Herbut
herbi-blog.blogspot.com
mherbi@gmail.com

Agenda

- JVM memory sections
- How to measure object size
- OutOfMemoryError
- Should we always enlarge the heap
- Usefull tools
- Case study
- Collections in Java

Java memory sections – short review



Graphics source: [1]

HotSpot heap areas

-Xms<size> initial Java heap size
-Xmx<size> maximum Java heap size
-Xss<size> java thread stack size
-XX:PermSize<size> Perm size
-XX:MaxPermSize<size> Perm size max value
-Xmn<size> Eden size

-XX:NewRatio<ratio> Tenured/Young ratio
-XX:SurvivorRatio<ratio> Eden/Survivor ration
-XX:MinHeapFreeRatio<ratio> Min percentage of heap free after GC to avoid expansion
-XX:MaxHeapFreeRatio<ratio> Max percentage of heap free after GC to avoid shrinking.

Graphics source: [1]

How to measure object size?

- Shallow size = [reference to the class definition] + space for superclass fields + space for instance fields + [alignment]
- Retained size = shallow size of our object + shallow size of all objects eligible by GC after removing our object
- Deep size = shallow size of our object + shallow size of all referenced (directly and indirectly) objects

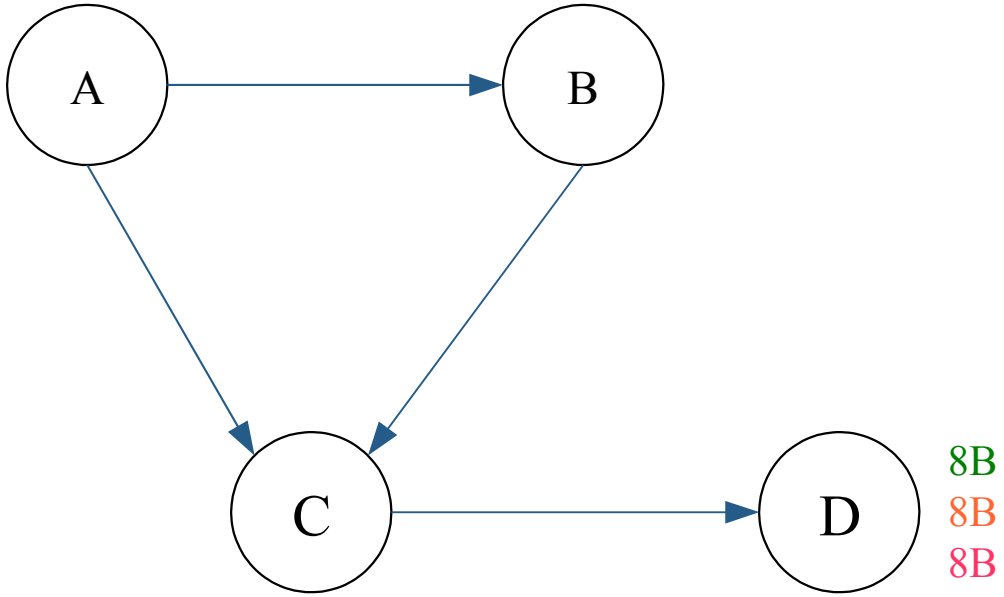
Graphics source: [2]

How to measure object size?

- Shallow size = [reference to the class definition] + space for superclass fields + space for instance fields + [alignment]
- Retained size = shallow size of our object + shallow size of all objects eligible by GC after removing our object
- Deep size = shallow size of our object + shallow size of all referenced (directly and indirectly) objects

Let's assume that:

- Reference to the class definition = 8B
- Reference to other objects = 4B
- Alignment = 0B

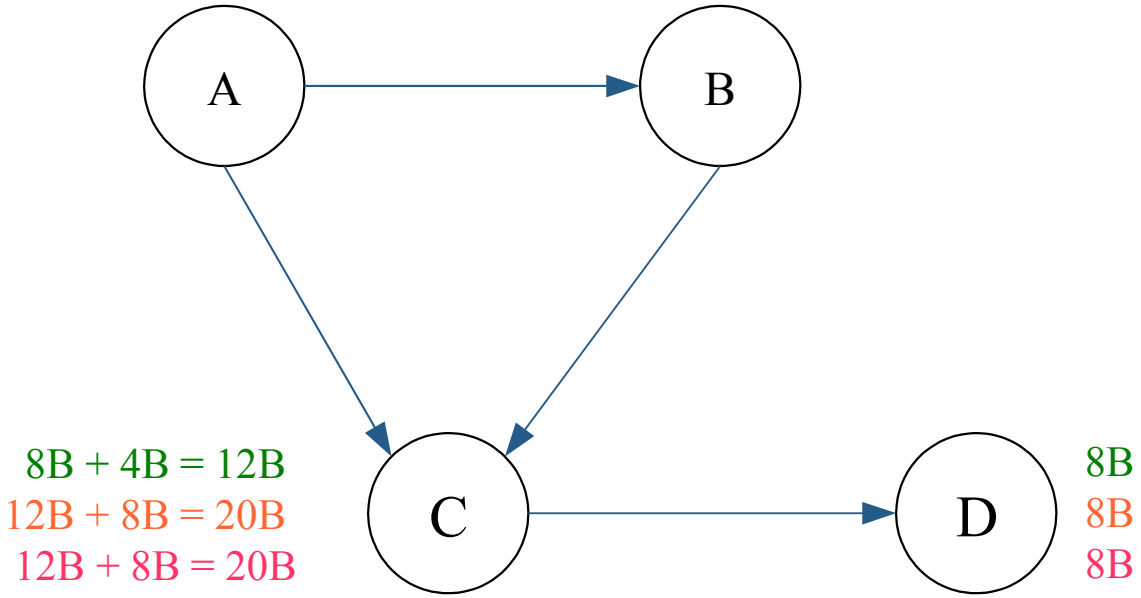


How to measure object size?

- Shallow size = [reference to the class definition] + space for superclass fields + space for instance fields + [alignment]
- Retained size = shallow size of our object + shallow size of all objects eligible by GC after removing our object
- Deep size = shallow size of our object + shallow size of all referenced (directly and indirectly) objects

Let's assume that:

- Reference to the class definition = 8B
- Reference to other objects = 4B
- Alignment = 0B

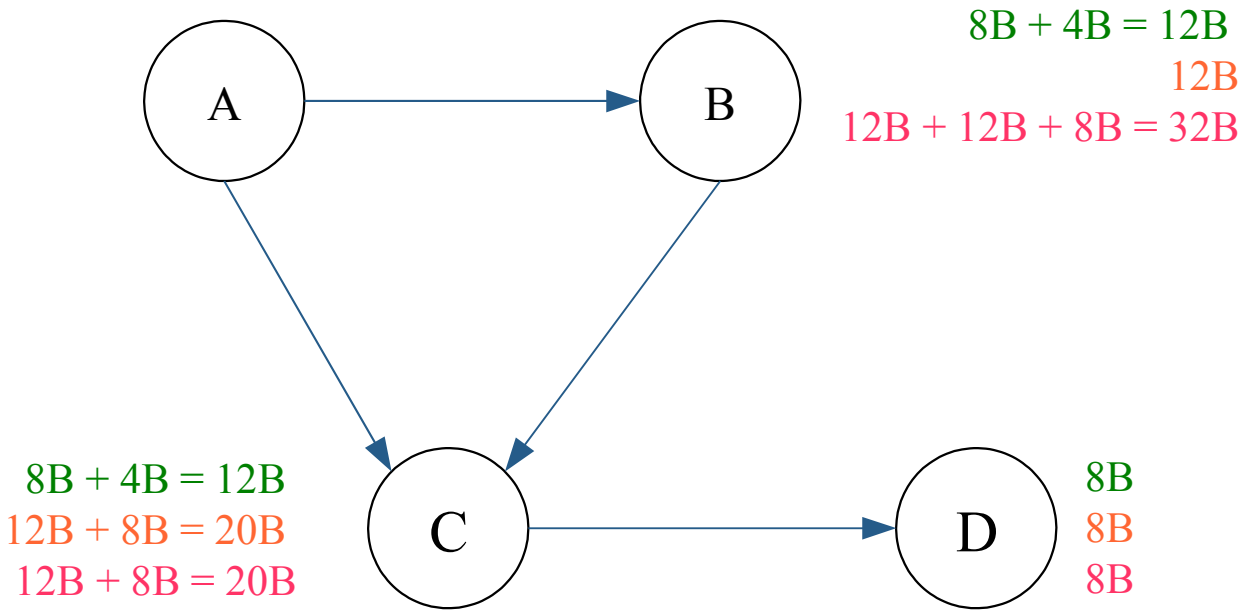


How to measure object size?

- Shallow size = [reference to the class definition] + space for superclass fields + space for instance fields + [alignment]
- Retained size = shallow size of our object + shallow size of all objects eligible by GC after removing our object
- Deep size = shallow size of our object + shallow size of all referenced (directly and indirectly) objects

Let's assume that:

- Reference to the class definition = 8B
- Reference to other objects = 4B
- Alignment = 0B

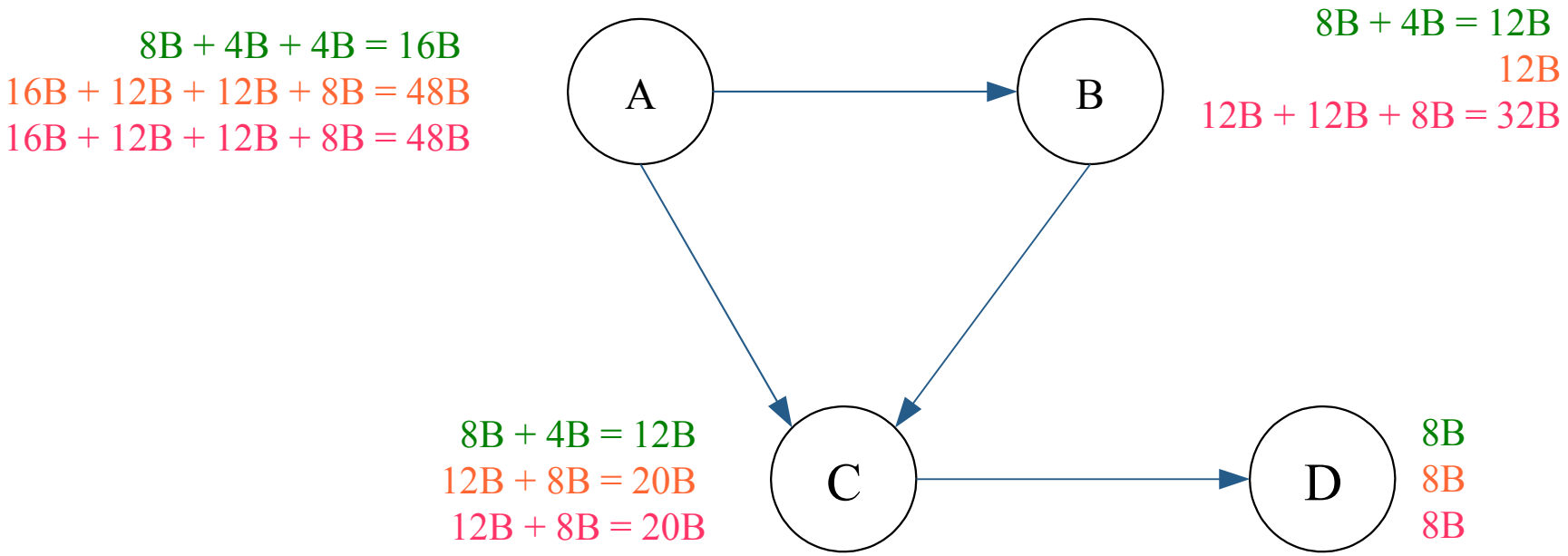


How to measure object size?

- Shallow size = [reference to the class definition] + space for superclass fields + space for instance fields + [alignment]
- Retained size = shallow size of our object + shallow size of all objects eligible by GC after removing our object
- Deep size = shallow size of our object + shallow size of all referenced (directly and indirectly) objects

Let's assume that:

- Reference to the class definition = 8B
- Reference to other objects = 4B
- Alignment = 0B



JVM alignment – what's that?

```
//8B for object header
public class ClassWithLong {
     //8B for long
     private long long_ = 258L;
}
//total = 16 B
```

JVM alignment – what's that?

```
//8B for object header
public class ClassWithLong {
     //8B for long
     private long long_ = 258L;
}
//total = 16 B
```

```
//8B for object header
public class ClassWithInt {
          //4B for int
          private int int_ = 258;
}
//total = ?
```

JVM alignment – what's that?

```
//8B for object header
public class ClassWithLong {
     //8B for long
     private long long_ = 258L;
}
//total = 16 B
```

```
//8B for object header
public class ClassWithInt {
          //4B for int
          private int int_ = 258;
}
//total = 12B + 4B of alignment = 16B
```

JVM alignment – what's that?

```
//8B for object header
public class ClassWithLong {
     //8B for long
     private long long_ = 258L;
}
//total = 16 B
```

```
//8B for object header
public class ClassWithInt {
          //4B for int
          private int int_ = 258;
}
//total = 12B + 4B of alignment = 16B
```

```
//8B for object header
public class ClassWithBoolean {

//1B for boolean
private boolean bool = false;

}

//total = ?
```

JVM alignment – what's that?

```
//8B for object header
public class ClassWithLong {
     //8B for long
     private long long_ = 258L;
}
//total = 16 B
```

```
//8B for object header
public class ClassWithInt {
          //4B for int
          private int int_ = 258;
}
//total = 12B + 4B of alignment = 16B
```

```
//8B for object header
public class ClassWithBoolean {

//1B for boolean
private boolean bool = false;

}

//total = 9B + 7B of alignment = 16B
```

JVM alignment – what's that?

```
//8B for object header
public class ClassWithLong {
     //8B for long
     private long long_ = 258L;
}
//total = 16 B
```

```
//8B for object header
public class ClassWithInt {
          //4B for int
          private int int_ = 258;
}
//total = 12B + 4B of alignment = 16B
```

```
//8B for object header
public class ClassWithBoolean {

//1B for boolean
private boolean bool = false;

}

//total = 9B + 7B of alignment = 16B
```

```
//8B for object header
//8B for superClass fields
public class ClassExtends16B
  extends ClassWithLong {

//8B for long
private long long2 = 258L;

}

//total = 24 B
```

JVM alignment – what's that?

```
//8B for object header
public class ClassWithLong {
     //8B for long
     private long long_ = 258L;
}
//total = 16 B
```

```
//8B for object header
public class ClassWithInt {
          //4B for int
          private int int_ = 258;
}
//total = 12B + 4B of alignment = 16B
```

```
//8B for object header
public class ClassWithBoolean {

//1B for boolean
private boolean bool = false;

}

//total = 9B + 7B of alignment = 16B
```

```
//8B for object header
//8B for superClass fields
public class ClassExtends16B
  extends ClassWithLong {

//8B for long
private long long2 = 258L;

}

//total = 24 B
```

```
//8B for object header
//1B? or 8B? for superClass fields
public class ClassExtends9B
  extends ClassWithBoolean {

//4B for int
private int int_ = 258;

}
```

JVM alignment – what's that?

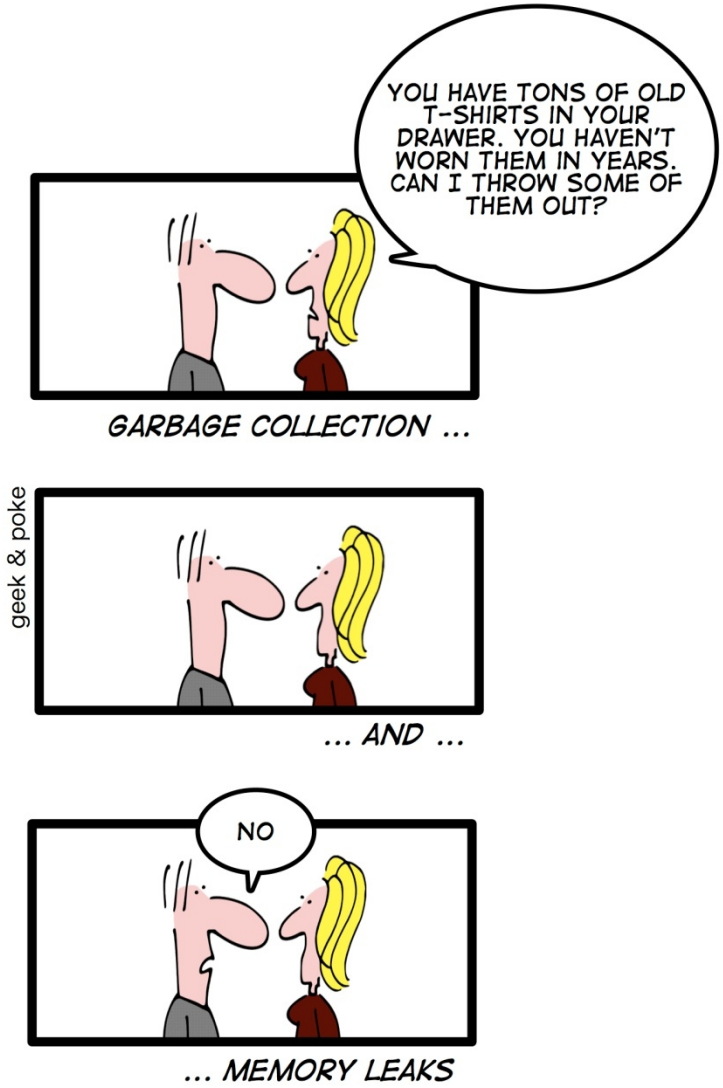
```
//8B for object header
public class ClassWithLong {
     //8B for long
     private long long_ = 258L;
}
//total = 16 B
```

```
//8B for object header
public class ClassWithInt {
          //4B for int
          private int int_ = 258;
}
//total = 12B + 4B of alignment = 16B
```

```
//8B for object header
public class ClassWithBoolean {

//1B for boolean
private boolean bool = false;

}

//total = 9B + 7B of alignment = 16B
```

```
//8B for object header
//8B for superClass fields
public class ClassExtends16B
  extends ClassWithLong {

//8B for long
private long long2 = 258L;

}

//total = 24 B
```

```
//8B for object header
//1B for superClass fields
public class ClassExtends9B
  extends ClassWithBoolean {

//4B for int
private int int_ = 258;

}

//total = 13B + 3B of alignment = 16B
```



Graphics source: [5]

Out of Memory Error

Graphics source: [4]

Out of Memory Error

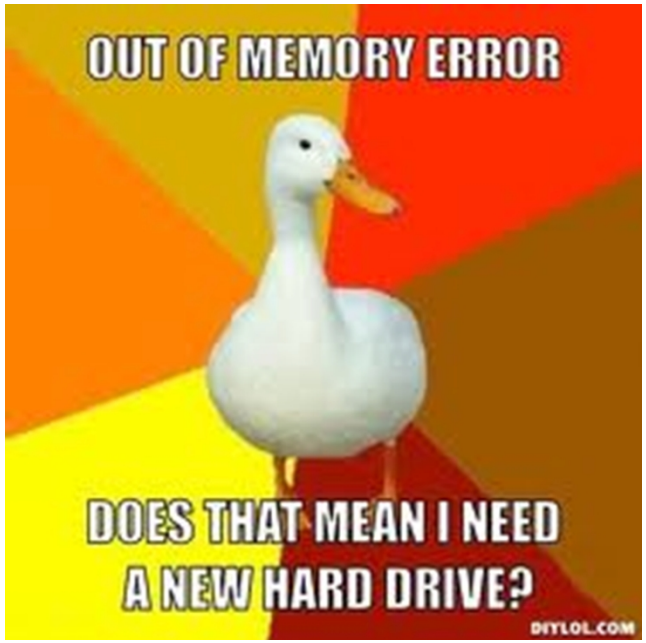
-XX:+HeapDumpOnOutOfMemoryError

-XX:HeapDumpPath=<path>

Graphics source: [4]

Should we always enlarge heap?

- The maximum theoretical heap limit for the 32-bit JVM is 4GB.
- In practice the limit can be much lower (~1,5GB on 32-bit Windows , 2GB on 32-bit Solaris).
- 64-bit operating system can allocate up to 4GB of heap memory on 32-bit JVM
- 64-bit JVM has larger (12B) object header than 32-bit JVM (8B)
- Object references can by either 4 bytes or 8 bytes, depending on JVM flags and the size of the heap.
- Using -XX:+UseCompressedOops flag cause compressing ordinary object pointers to 4B
- Compressed Oops option limits your heap size up to 32GB



Graphics source: [5]

64-bit overhead

„The performance difference comparing an application running on a 64-bit platform versus a 32-bit platform on SPARC is on the order of 10-20% degradation when you move to a 64-bit VM. On AMD64 and EM64T platforms this difference ranges from 0-15% depending on the amount of pointer accessing your application performs.”

[http://www.oracle.com/technetwork/java/hotspotfaq-138619.html]

„If you have a larger room to clean then it tends to take more time for the janitor to clean the room. The very same applies to cleaning unused objects from memory. When running applications on small heaps (below 4GB) you often do not need to think about GC internals. But when increasing heap sizes to tens of gigabytes you should definitely be aware of the potential stop-the-world pauses induced by the full GC. The very same pauses did also exist with small heap sizes, but their length was significantly shorter – your pauses that now last for more than a minute might have originally spanned only a few hundred milliseconds.”

[http://plumbr.eu/blog/increasing-heap-size-beware-of-the-cobra-effect]

Usefull tools

- Low-level Java object layout dumpers https://github.com/shipilev/java-object-layout
- Jmap & Jps:
  jmap -dump:live,format=b,file=<filename> <PID>
- Eclipse Memory Analyzer (MAT)
  http://www.eclipse.org/mat/
- Memory Leak Detector (part of Oracle JRockit Mission Control)

  www.oracle.com/technetwork/middleware/jrockit/overview/ind
- Visual JVM
- jstat

Java Collections

- java.util collections
- GS Collections:

  https://github.com/goldmansachs/gs-collections
- Google collections:

  http://code.google.com/p/guava-libraries/
- Trove primitive collections

  http://trove.starlight-systems.com/

Retained heap size of collections (10 000 elements)

Collection	Retained size
int[]	40016
Integer[]	197968
ArrayList<Integer>	197992
Vector<Integer>	198952
LinkedList<Integer>	398000
FastList<Integer>	204464
TIntArrayList	41000
TIntLinkedList	240024

Retained heap size of collections (10 000 elements)

Collection	Retained size
HashSet<Integer>	463576
TreeSet<Integer>	478016
UnifiedSet<Integer>	223528
TIntSet	128672
RangeSet<Integer>	168-1118024

Retained heap size of collections (10 000 elements)

Collection	Retained size
HashMap<Integer, Integer>	621512
TreeMap<Integer, Integer>	635968
LinkedHashMap<Integer, Integer>	701552
UnifiedMap<Integer, Integer>	447032
TIntIntHashMap	231560
TIntObjectHashMap<Integer>	389544
TObjectIntHashMap<Integer>	363784

Graphics sources

[1] http://blog.codecentric.de/

[2] http://plumbr.eu/blog/

[3] http://geekandpoke.typepad.com/

[4] http://pune.emagzin.com/

[5] http://diylol.com

Presentation was inspired by:

- http://plumbr.eu/blog
- http://eclipsesource.com/blogs/2013/01/21/10-tips-for-using-the-eclipse-memory-analyzer