



Memory Leaks

Learning objectives

At the end of this module you will be able to:

- Describe the causes of memory leaks in Java
- Distinguish between a memory leak in the Java heap and the C heap
- Recognize causes of memory leaks in Java
- Recommend tools for finding such leaks

Memory Leak - Definition

Error in the program's dynamic storage allocation logic that causes:

1. Failure to reclaim memory no longer in use
2. Eventual memory exhaustion

Components in a Java program

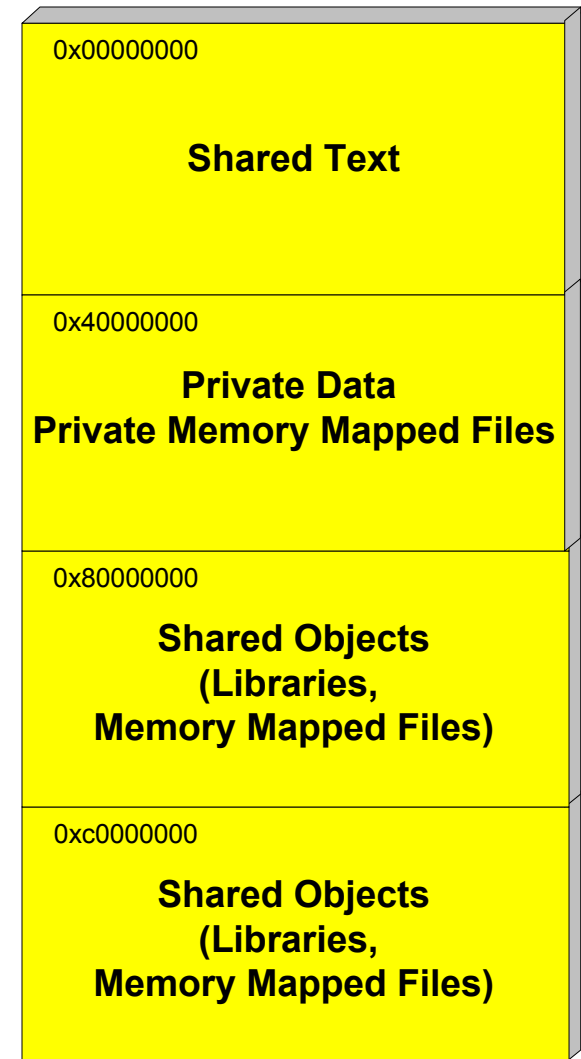
- Virtual Machine is written in C/C++
- Java code
- Java code calling native methods
- Native code calling into Java code

Memory Allocation

- Java Heap
 - Objects created with the 'new' keyword in Java
- C Heap
 - Memory allocated with
 - 'malloc' in C
 - 'new' in C++

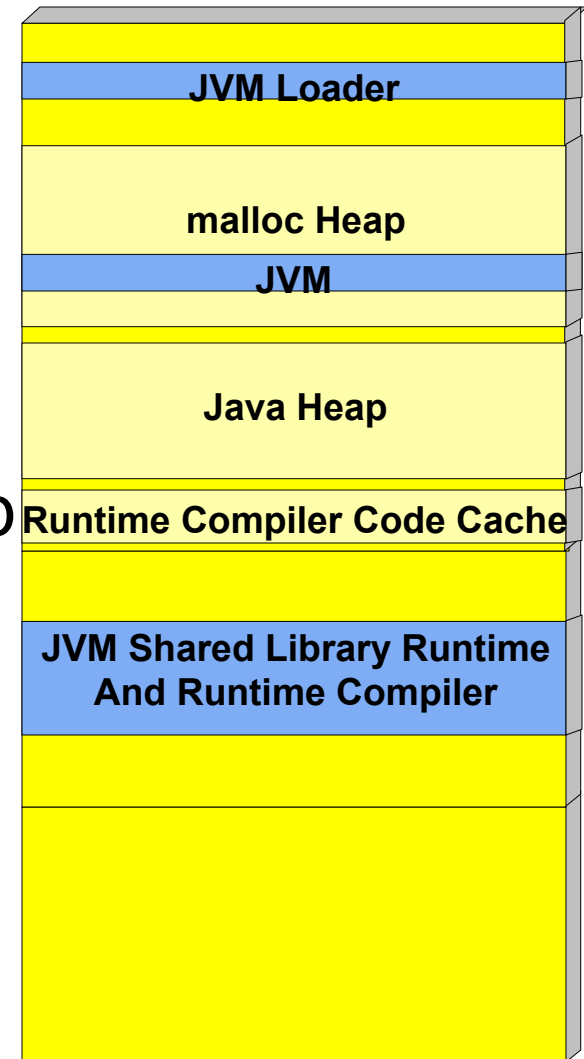
Virtual Memory Layout HP-UX 32 Bit Process

- Four 1 GB Quadrants
- Java Heap
 - JDK 1.2 maximum of 1.8 GB
 - JDK 1.3.1 maximum of 3.8 GB



Virtual Memory Layout HP-UX 32 Bit Process

- JVM executable
- Process stack/heap
- Garbage collected Java Heap
- Dynamically loaded libraries
 - libjava.sl - Java interpreter
 - other Java native libraries



Memory
Leaks

Tools Available

Process: Glance Memory Regions

GlancePlus - Process Memory Regions (java)

File Reports Admin Configure Help

System: hp46t181 Last Update: 17:01:05 Int: 830 ms T ?

java PID: 2233 PPID: 2232 User: root State: active

Memory Regions: All 218 Selected

Data RSS	(kb): 131028	Data VSS	(kb): 131048
Text RSS	(kb): 16	Text VSS	(kb): 20
Stack RSS	(kb): 1088	Stack VSS	(kb): 2112
Shared Mem RSS	(kb): 0	Shared Mem VSS	(kb): 0
Other RSS	(kb): 1714756	Other VSS	(kb): 1850016
Private RSS	(kb): 1838776		
Shared RSS	(kb): 8112		

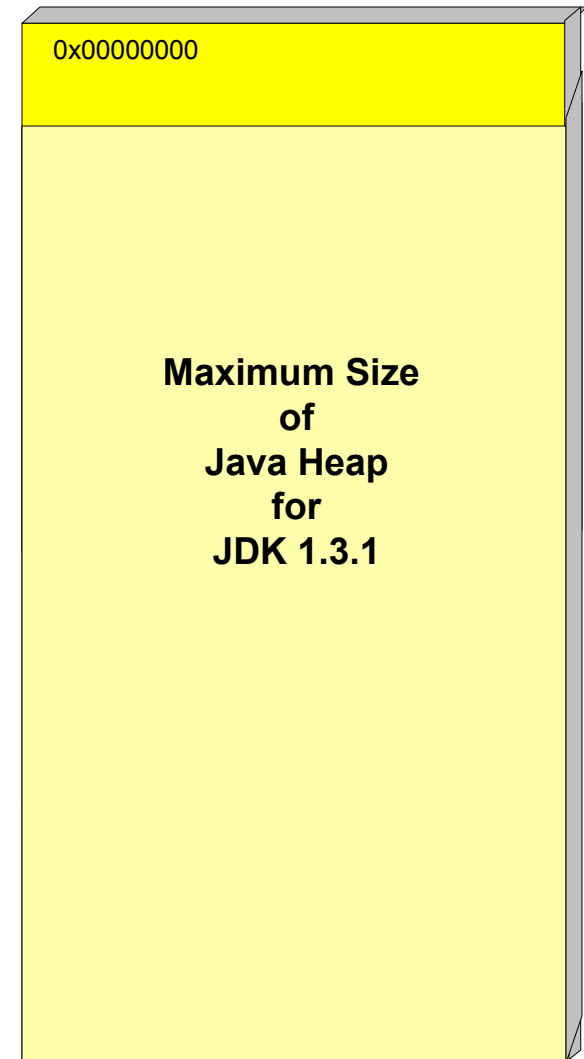
C Heap

Type	File Name	P/S	RSS KB
MEMMAP	<mmap>	Priv	1.59gb
DATA	<vxfs,/opt,/dev/vg00/1vol6,inode:2529>	Priv	128.0mb
MEMMAP	<mmap>	Priv	17.2mb
MEMMAP	<mmap>	Priv	388kb
MEMMAP	<vxfs,/opt,/dev/vg00/1vol6,inode:4549>	Shared	6.0mb
MEMMAP	<mmap>	Priv	3.2mb

Java Heap

Virtual Memory Layout HP-UX 32 Bit Process

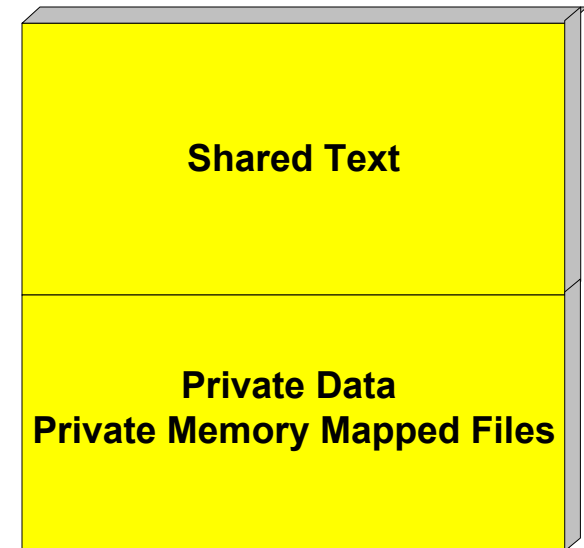
- Four 1 GB Quadrants
- Java Heap
 - JDK 1.2 maximum of 1.8 GB
 - EXEC_MAGIC
 - JDK 1.3.1 maximum of 3.8 GB



Virtual Memory Layout

HP-UX 64 Bit Process (PA + IA-64)

- Limited only by word size
 - No quadrants
- Java Heap
 - Nearly unlimited in size
 - Limited by time required to collect



⋮



Garbage Collector

A memory manager that reclaims objects that are not reachable from a root-set

- Root set definition:
 - All local reference variables in each frame of every thread's stack
 - All static reference fields in all loaded classes
 - JNI references to Java objects stored in the C Heap

C heap

- There is no garbage collector for the C heap
- A leak in the C heap originate in:
 - JVM code
 - Native code in the APIs
 - Native code in the application accessed using JNI

Java Heap – Object Retention

- Potential causes for object retention
 - JNI references not cleaned up properly
 - Can prevent the collection of other Java objects
 - Local versus Global
 - All reachable objects may not be actively in use
 - An object or objects referenced by a long-lived objects but no longer required by the application

Java Objects Allocated, Reachable and Live

- **Allocated objects**

- The set of **all** objects in the Java Heap at any given time during program execution

- **Reachable objects**

- If we can **reach** the object **from the root** set through any number of intermediate references

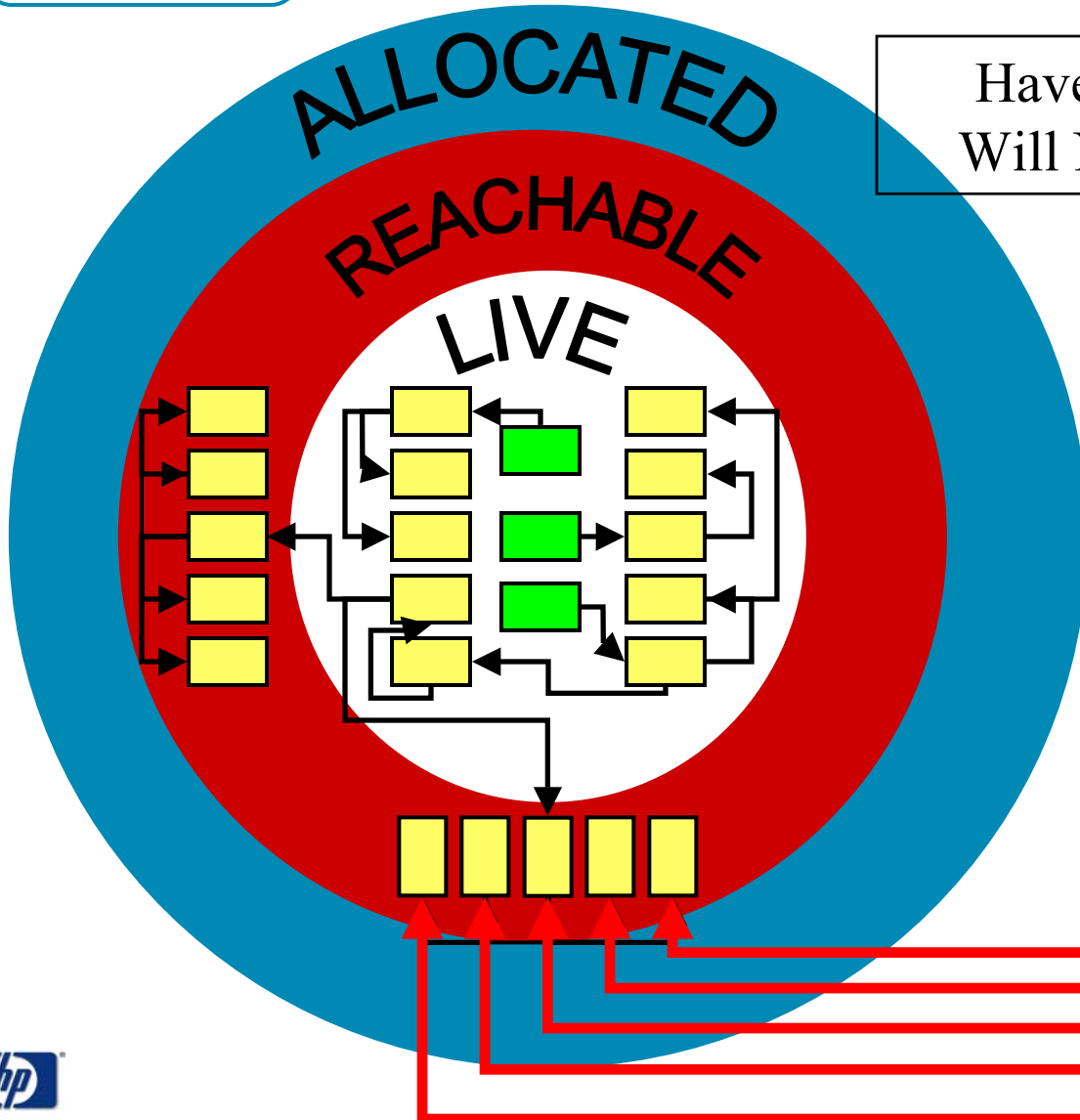
- **Live objects**

- Reachable objects in **active use** by the program

Memory
Leaks

What is a Memory Leak in Java?

Have references from Root Set
Will NOT be Garbage Collected



Root Set Reference
Object in Java Heap

Symptoms of a Memory Leak

- Either Java or C Heaps:
 - Java “OutOfMemoryException”
- Java Heap Leak:
 - Unaccountable growth of the Java Heap
- C Heap Leak:
 - Constantly increasing DATA RSS and VSS
 - System running out of swap space
 - Programs failing with out of memory (ENOMEM) errors

Java OutOfMemoryException

- Insufficient Java Heap space
 - Objects holding space in the Java heap
 - Too many objects with finalizers
- Low values for Operating System kernel parameters
 - maxdsiz Data region size
 - max_thread_proc Number of threads per process
 - nkthread Total number of threads
 - maxssiz Stack size
 - maxtsiz Text region size
 - nfiles Total number of open files
 - maxfiles Number of open files per process

Leak Example 1

Object No Longer Required

```
static public void run() {  
    File f1 = readXML("X1"); // Create large objects  
    // Create f2 ... to fn-1 here  
    File fn = readXML("Xn");  
    Code c1 = f1.parse();    // Process the information for each  
    // parse() f2 ... to fn-1 here  
    Code cn = fn.parse();  
    // f1 to fn are not needed any more, so could be set to null  
    while (1) {                // This method will run a long time  
        ... do stuff with c1 ... cn  
    }  
}
```

Leak Example 2

Reference from Long-lived Object

```
// Class A 's objects are long lived
```

```
Class B
```

```
{
```

```
    public B( ) {
```

```
        A.addListener(this);
```

```
    }
```

```
    public static void main(String[ ] args) {
```

```
        B b = new B( );
```

```
        // do stuff with b
```

```
        b = null; // leak if no removeEventListener
```

```
    }
```

```
}
```

Leak Example 3

Inadvertent Reference Held

```
public class SimpleExampleFrame extends JFrame
    implements ActionListener {
    protected void finalize() throws Throwable{
        System.out.println("FIN....");
        metalButton.removeActionListener(this);
        motifButton.removeActionListener(this);
        windowsButton.removeActionListener(this);
        // No call to super class's super.finalize()
        // Omitting this call causes a reference to the
        // object to be held indefinitely
    }
}
```

And more leaks ...

- Object reuse
 - Some fields of the object are updated with newer values
 - Some fields still hold references from previous uses
 - Prevents the object referenced in the previous usage from being garbage collected

Tools for Memory Leak Detection

- -Xverbosegc
 - Plotting of data over time
- Glance/gpm
 - Process Memory Regions page's VSS values
- gdb – gnu debugger
 - See tutorial on-line information at www.hp.com
- HPjmeter
 - Xrunhprof heap analysis

Memory Leaks: Summary

You should now be able to:

- Explain the use of the Java and C Heaps in Java programs
- Use the appropriate tools to check whether there is memory growth in either of the Heaps
- Use tools to track down memory leaks