JAVA Hints and Tips, Part 1
Finalizers

Kenneth Irwin

Session 8379

March 2, 2005

irwink@us.ibm.com

Objectives

- An introduction to Garbage Collection and Finalization
- Problems associated with Finalization
- Hints and tips for avoiding these problems

Agenda

- What are Finalizers
  - Something about Java and Garbage Collection
  - From the Application perspective
  - From the Implementation (JVM) perspective
- Problems associated with Finalizers
  - Increased Java Heap occupancy
  - Resource exhaustion
- Hints and tips for avoiding these problems
  - Implementing a synchronous close method
  - ‘Forcing’ Finalization
  - The Weak reference alternative

Something about java and Garbage Collection

- Objects are created explicitly

  classA objectA = new classA();

and are allocated in a contiguous storage area called the java heap.

- There is no command in java to free this storage, such as:

  free objectA;

- When the java heap is full the jvm invokes a process called ‘Garbage Collection’ which marks storage occupied by unreferenced objects as available for reuse.

Finalizers from the application perspective

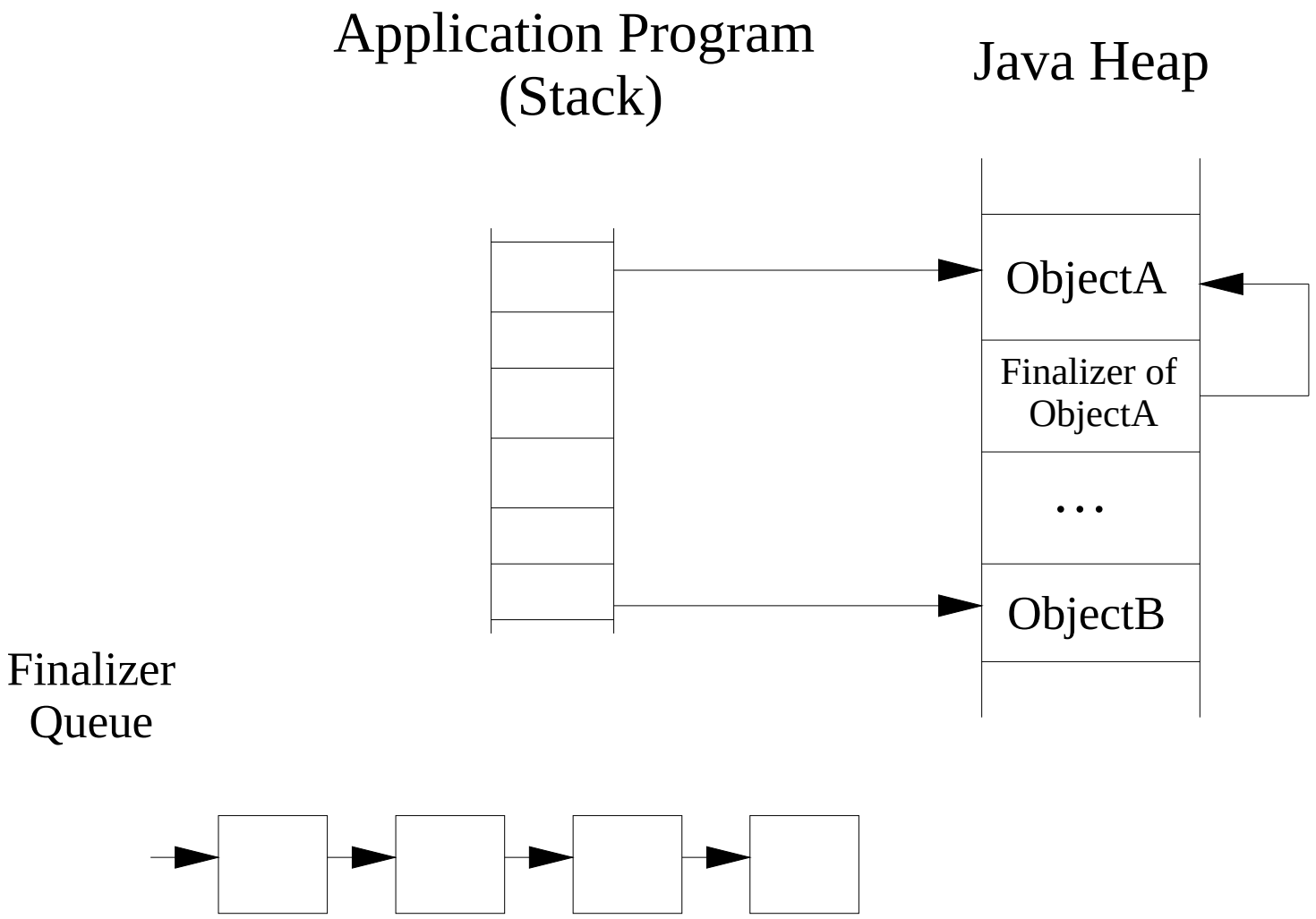
```
public class ClassA
{
   …

   protected void finalize ()
   {
       //some statements
    }
}
```

- The finalize() method is run when there are no more references to the object.

- Finalizers provide a chance to free up resources (such as file descriptors or operating system graphics contexts) that cannot be freed automatically by an automatic storage manager.
  http://java.sun.com/docs/books/jls/first_edition/html/12.doc.html

Finalizers from the JVM perspective



Verbosegc output

<AF[7]: Allocation Failure. need 20016 bytes, 5 ms since last AF>
<AF[7]: managing allocation failure, action=1 (143984/4025056)
(167712/167712)>
<GC(7): GC cycle started Sun Jul 04 01:04:53 2004
<GC(7): freed 2824952 bytes, 74% free (3136648/4192768), in 3 ms>
<GC(7): mark: 3 ms, sweep: 0 ms, compact: 0 ms>
<GC(7): refs: soft 0 (age >= 32), weak 0, final 141, phantom 0>
<AF[7]: completed in 36 ms>

Problems with Finalizers

- Increased Java Heap occupancy
  - Finalizers delay object collection for an undefined period.
  - Causing degradation on GCs key performance measures:
    - Pause time (increased mark phase, mark stack overflow)
    - Time between GCs reduced (less free space)

Problems with Finalizers

- Resource exhaustion
  - One of the uses of Finalizers is to ‘free’ system resources such as native storage or file handlers.
  - … but there is no guarantee that finalizers will run.

Hints and Tips for avoiding problems with Finalizers

- if you must use them:
  - Keep their content to a minimum. Finalizers are run on a single thread and sequentially so a long running finalizer will ‘block’ other finalizers from being run
  - Consider the alternatives …

Hints and Tips for avoiding problems with Finalizers

- Use of a synchronous close() method
  - If the application is able to determine that an object will no longer be used free resources using a synchronous ‘close()’ method.
  - In the finalizer check that resources have been freed
  - This approach will alleviate the resource exhaustion problem.

Hints and Tips for avoiding problems with Finalizers

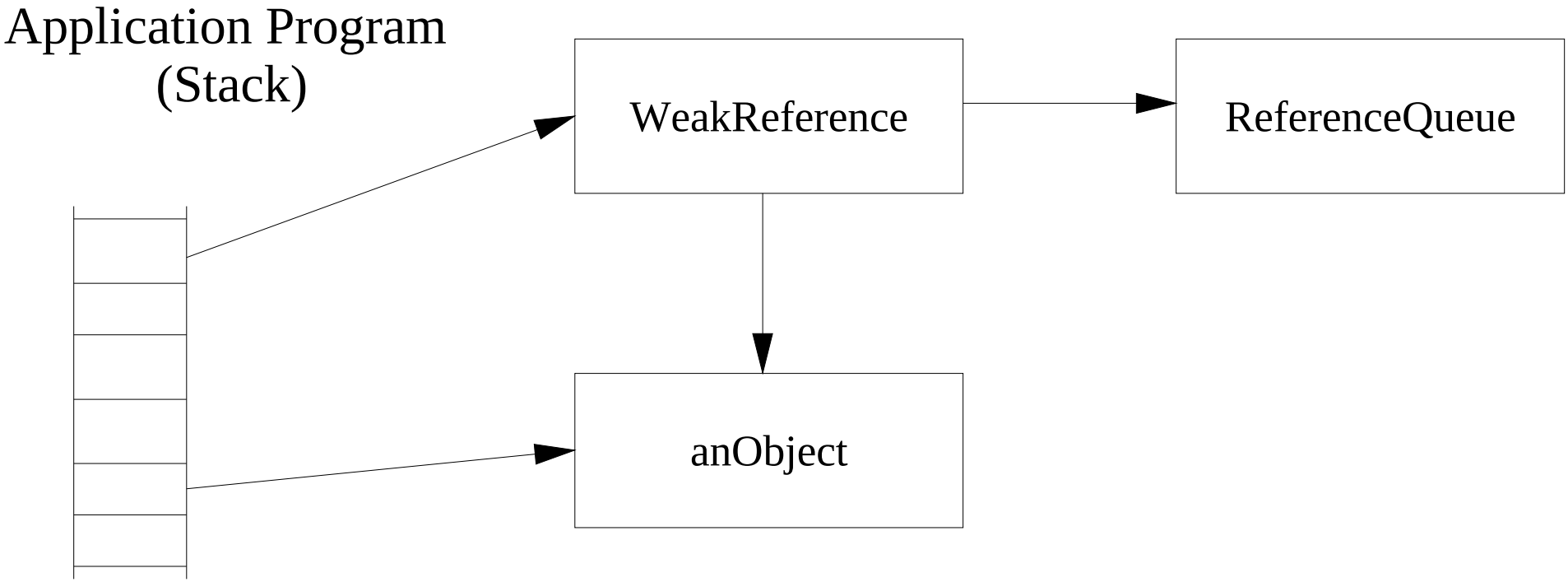
- ‘Forcing’ finalization
  - Even if there are no references to an object, GC is required to recognise this and enqueue the object’s finalizer before it can be run.
  - It is possible for resources to be exhausted before a GC is (ever) run.
  - Finalizers can be enqueued for finalization and the finalizers run synchronously from the application using:
  -

```
System.gc();     // calls a GC
System.runFinalization();  // this thread acts as an additional finalizer thread
```

  - This has performance implications

Hints and Tips for avoiding problems with Finalizers

- The Weak Reference Alternative
  http://java.sun.com/developer/technicalArticles/ALT/RefObj/
-



Summary

- Garbage Collection enqueues an object's finalizer when there are no longer (strong) references to the object.
- A finalizer frees up resources which can't be freed up automatically
- Reading the verbosegc output from an IBM JVM
- Problems with finalizers: resource exhaustion and GC performance
- Tip: Keep finalizer content to a minimum
- Tip: Consider using a synchronous close() method for freeing resources
- Tip: Finalization can be 'forced'
- Tip: WeakReferences can be used for notification of object collection