Swing Implementation Issues

- Strange as it may seem, Swing has implementation issues that sound a lot like operating system concepts:
  - Memory management — specifically, memory leaks (what?? in Java???)
  - Process management — a.k.a. some implications of Swing's single-threaded nature
- Fortunately, there are ways to address these issues, though they may change the way you have written code so far

Implementation Testing Aids

<table>
<thead>
<tr>
<th>Good</th>
<th>Operating system tools: memory or process monitors can show coarse-grained memory or threading issues</th>
</tr>
</thead>
<tbody>
<tr>
<td>Better (sort of)</td>
<td>In-software instrumentation: code can explicitly detect or report the status of threads, object existence</td>
</tr>
<tr>
<td>Best</td>
<td>Java profilers — they allow instrumentation without needing explicit code, and have pretty good user interfaces, too: JProfiler, Optimizelt</td>
</tr>
</tbody>
</table>
Caveat Listener

• Yes, improperly written Swing programs *can* and *do* leak memory

• The main cause comes from orphaned listeners (“loiterers”) for the Java/Swing event model:

Don’t Lose Track of Your Listeners

• Listeners are primarily for “non-deterministic” and “asynchronous” notification:
  ◇ User activity
  ◇ Network notifications
  ◇ Updates from unknown sources

• Treat listeners like pointers in other languages — when you don’t need them anymore, disconnect them
  ◇ Use `addNotify()` / `removeNotify()`
  ◇ Hold onto listeners as instance variables
Swing is Single-Threaded

• While you won’t find this in the tutorials, you will find a good write-up (3, in fact) on threads and Swing in the Swing Connection Web site. Consider these articles as your primary reference for this subject:


• The main points, in a nutshell:

  1. Swing is single-threaded: the “event dispatch thread”
  2. Swing components must be touched only from within this thread — and absolutely nowhere else

Threading Ground Rules

• If you’re going to change a Swing component (set, add, remove, anything), do it on the Swing thread — the article’s official wording is “all code that might affect or depend on the state of that component”

• There are a few exceptions: repaint(), revalidate(), invalidate(); add/remove listener methods

• Put any lengthy, unpredictable, or otherwise non-UI activity on a different thread

• …but how about user interface feedback?
SwingUtilities to the Rescue

• SwingUtilities provides a number of static methods that assist in threading:
  ◆ public static boolean isEventDispatchThread() tells you if you are on the Swing thread
  ◆ public static void invokeLater(Runnable) posts some code (as a Runnable) for execution in the Swing thread at the next possible opportunity

• For threaded operations that follow certain operational flows, some wrapper classes are available...

Common Threading Models

• SwingWorker (available through Swing Connection threading articles) — for long, non-UI tasks followed by Swing feedback

• ProgressMonitor, ProgressMonitorInputStream — for long tasks that require on-going progress feedback
  ◆ My personal opinion — not recommended; not enough control of user interface
  ◆ But adequate for quickie threading/feedback