MultithreadedTC
Deterministic and repeatable unit tests for concurrent abstractions

FEATURES
Simple Clock MultithreadedTC provides a clock which test designers can use to regulate the activities in multiple threads. But the clock is not a timer. It advances to the next requested tick when all threads are blocked.

Just Java Each test case is a Java class. Threads are specified using thread methods which return void, have no arguments and have names prefixed with the word “thread”. Java 1.4 compatible version available.

Integrates with JUnit Run the test framework from a JUnit test. Use JUnit assertions to verify the current clock tick. Exceptions thrown in threads will cause entire test to fail.

Concise Code MultithreadedTC eliminates much of the scaffolding code needed when writing concurrent tests, e.g. setting up and tearing down threads, joining threads, using Thread.sleep()

EXAMPLE
Let us validate some properties of a bounded blocking buffer with a capacity of 1 element. We want to ensure that:
(a) The assertion \texttt{take = 42} occurs after the call to \texttt{put 17}
(b) The call to \texttt{put 17} blocks thread 1

Solution 1: Use \texttt{Thread.sleep()} to delay the first statement in thread 2. This introduces unnecessary timing dependence (test does not work well in a debugger or with an ill-timed garbage collector).

Solution 2: Use a latch to coordinate activities in both threads. This will not work because the call to \texttt{put 17} blocks thread 1 before the latch can be released.

Solution 3: Use MultithreadedTC!

class BoundedBufferTest extends MultithreadedTestCase { 
    BoundedBuffer buf;
    ...
}

USEFUL METHODS
- \texttt{waitForTick(tick)}: cause the host thread to block until the clock reaches tick
- \texttt{assertTick(tick)}: compare tick with the current clock and throw an assertion error if they do not match
- \texttt{getThread(threadID)}: returns a reference the the Thread object corresponding to threadID
- \texttt{freezeClock()}: prevent the clock from advancing until \texttt{unfreezeClock()} is called. Useful when doing tests that involve timed waiting and you don’t want to advance the clock during the wait.

RELATED WORK

ConTest is a Java testing framework that uses a deterministic replay algorithm to record and replay specific interleavings that lead to faults.

JUnit, TestNG, GroboUtils, and ConTest all provide facilities for running concurrent tests many times to generate a representative set of interleavings.

Lines of Code

<table>
<thead>
<tr>
<th>Constructs removed from TCK Tests</th>
</tr>
</thead>
<tbody>
<tr>
<td>Anonymous inner classes</td>
</tr>
<tr>
<td>Thread’s join() method</td>
</tr>
<tr>
<td>Try-catch blocks</td>
</tr>
<tr>
<td>Thread’s sleep() method</td>
</tr>
</tbody>
</table>

ConTest is a Java testing framework that uses a deterministic replay algorithm to record and replay specific interleavings that lead to faults.

JUnit, TestNG, GroboUtils, and ConTest all provide facilities for running concurrent tests many times to generate a representative set of interleavings.