org.jax.mgi.shr.timing Design Document

Author: jsb

Created: June 10, 2003

Last Modified: May 25, 2004 11:11

1 Purpose of Document

This document outlines the design for the `org.jax.mgi.shr.timing` package.

2 Organizing Principle

All classes contained in `org.jax.mgi.shr.timing` deal with measuring times during the execution of code.

3 Configuration

All classes in `org.jax.mgi.shr.timing` are stand-alone; no configuration is necessary.

4 Classes

Rather than present a detailed API for each class, we only present an overview here. The details are covered in the javadocs for each class, so we do not replicate them here.

Note that both the `Stopwatch` and `Countdown` classes deal in a `double` number of seconds, rather than milliseconds.

4.1 Stopwatch

The `Stopwatch` class represents a traditional stopwatch. For a given `Stopwatch` object, you can start the timer, stop it, reset it, and look at the elapsed time (whether currently running or not).

Example 1: Using a `Stopwatch` to time a section of code

```
Stopwatch timer = new Stopwatch();
timer.start();
...
timer.stop();
System.out.println (timer.time());
```

4.2 Countdown

The `Countdown` class represents a lightweight kitchen timer -- you can tell it how much time you want it to count down, but you must periodically check to see if it has elapsed (there is no notion of a bell sounding to let you know). The main operations are resetting the `Countdown` timer and asking whether the time has elapsed or not.

Example 2: Using a `Countdown` to rebuild an object every 30 seconds

```
Object foo = new Object();
Countdown timer = new Countdown (30.0);
while (true)
{
    if (timer.elapsed())
    {
        foo = new Object();
        timer.reset();
    }
Thread.sleep(100);
}
```

4.3 TimeStampedMessage

A `TimeStampedMessage` is a message with an associated date/time `String` and a `double` number of seconds noting how far into a program’s runtime this message was recorded. The three attributes are accessed using `getMessage()`, `getDateTime()`, and `getSeconds()`. See Example 3 for usage.

4.4 TimeStamper

A `TimeStamper` records messages and adds a time-stamp to them, for use in simplistic profiling of code. “Simplistic profiling” means that a programmer can record messages at various points in the code, then look at the timestamps to help identify bottlenecks in the program. The main operations are recording messages and getting a `List` of messages recorded so far.

Example 3: using a `TimeStamper`

```
TimeStampedMessage entry = null;

TimeStamper ts = new TimeStamper();
ts.record (“message 1”);
Thread.sleep (5000);
ts.record (“message 2”);

List messages = ts.getMessages();
int messageCount = messages.size();

for (int i = 0; i < messageCount; i++)
{
   entry = messages.get(i);
   System.out.println (entry.getDateTime());
   System.out.println (entry.getMessage());
   System.out.println (entry.getSeconds());
   System.out.println ();
}
```