An Example: Timer vs ScheduledThreadPool

Contrasting Timer and ScheduledThreadPool

As an example, we'll compare and contrast using a timer and a pool to schedule periodic or delayed threads.

Timer

The achilles' heel of the `Timer` class is its use of a single thread to execute submitted tasks. Timer has a single worker thread that attempts to execute all user submitted tasks. Issues with this approach are detailed below:

- If a task misbehaves and never terminates, all other tasks would not be executed
- If a task takes too long to execute, it can block timely execution of other tasks. Say two tasks are submitted and the first is scheduled to execute after 100ms and the second is scheduled to execute after 500ms. Now if the first task takes 5 minutes to execute then the second task would get delayed by 5 minutes rather than the intended 500ms.
- In the above example, if the second task is scheduled to run periodically after every 500ms, then when it finally gets a chance to run after 5 minutes, it'll run for all the times it missed its turns, one after the other, without any delay between consecutive runs.

```
import java.util.Timer;
import java.util.TimerTask;

class Demonstration {
    public static void main( String args[] ) throws Exception {
        Timer timer = new Timer();
        TimerTask badTask = new TimerTask() {

            @Override
            public void run() {

                // run forever
                while (true)
                    ;

            }
        };

        TimerTask goodTask = new TimerTask() {

            @Override
            public void run() {

                System.out.println("Hello I am a well-behaved task");

            }
        };

        timer.schedule(badTask, 100);
        timer.schedule(goodTask, 500);

        // By three seconds, both tasks are expected to have launched
        Thread.sleep(3000);
    }
}
```

Bad Use of Timer

Below is another example of `Timer`'s shortcoming. We schedule a task which throws a runtime exception and ends up killing the lone worker thread Timer possess. The subsequent submission of a task reports the *timer is canceled* when in fact the previously submitted task crashed the Timer.

```
import java.util.Timer;
import java.util.TimerTask;
```

```
class Demonstration {
    public static void main( String args[] ) throws Exception{


        Timer timer = new Timer();
        TimerTask badTask = new TimerTask() {

            @Override
            public void run() {
                throw new RuntimeException("Something Bad Happened");
            }
        };

        TimerTask goodTask = new TimerTask() {

            @Override
            public void run() {
                System.out.println("Hello I am a well-behaved task");
            }
        };

        timer.schedule(badTask, 10);
        Thread.sleep(500);
        timer.schedule(goodTask, 10);
    }
}
```