

Example: Web Server Statistics

- Track number of requests and average processing time
 - ◆ Statistics updated by server thread
 - ◆ Statistics reported via UI driven from the main application thread
- Shared data must be protected via synchronization
 - ◆ `synchronized public` accessor methods
 - ◆ `synchronized` statement within internal server method

```
// WebServer9.java
private long requests = 0;
private double avgTime = 0.0;

public synchronized long getRequests() {
    return requests;
}
public synchronized double getAverageTime() {
    return avgTime;
}
// ...
private void processRequest(Socket sock) throws IOException {
    long start = System.currentTimeMillis();
    // ... Process
    long elapsed = System.currentTimeMillis() - start;
    synchronized(this) {
        avgTime = (avgTime*requests + elapsed)/(++requests);
    }
}
```

© 2003 David Holmes and Doug Lea

47

Example: Web Server Statistics

- Functional requirements:
 - ◆ Track number of requests
 - ◆ Track average processing time per request
 - ◆ Update each statistic at the end of each request
 - ◆ Allow each statistic to be read
- Safety requirements:
 - ◆ Statistics are always valid (no storage conflicts)
 - ◆ All statistics must be updated together
 - ◆ Querying statistics should always yield a valid set of statistics

© 2003 David Holmes and Doug Lea

55

Concurrent Programming in Java

Isolating Dependent Representations

- Class `Webserver9` fails to satisfy all safety requirements
- No protection from interleaving such as:

Main Thread	Server Thread
<code>n = ws.getRequests();</code>	...
...	<code>processRequest(sock);</code>
<code>T = ws.getAverageTime();</code>	...

- ◆ Main thread sees invalid set of statistics
 - Old number of requests and new average processing time
- Avoid by **splitting** out dependent representation into a separate class

WebServer

Statistics getStatistics()

→ stats

Statistics

requests
avgTime

© 2003 David Holmes and Doug Lea 56

Concurrent Programming in Java

Split Representation Example

```

public class WebServer10 {
    public static class Statistics { // immutable
        public final long requests;
        public final double avgTime;

        public Statistics(long requests, double avgTime) {
            this.requests = requests;
            this.avgTime = avgTime;
        }
    }

    private Statistics stats = new Statistics(0, 0.0);
    public synchronized Statistics getStatistics() {
        return stats;
    }
    // ...
    private void processRequest(Socket sock) throws IOException {
        // .... Process
        synchronized(this) {
            double total = stats.avgTime*stats.requests + elapsed;
            stats = new Statistics(stats.requests + 1,
                                  total / (stats.requests+1));
        }
    }
}

```

© 2003 David Holmes and Doug Lea 57

Split Objects

- Clients of principal classes can only view consistent states
- Splitting can be used in other contexts
 - ◆ Reducing lock contention
 - Associate a lock object with an isolable subset of state and functionality
 - ◆ Reducing “condition variable” contention
 - Associate a condition object with an isolable subset of methods with the same wait conditions
 - To be discussed in a later section
 - ◆ Enabling rollback
 - Isolate some or all state in separate objects so you can save and track versions

© 2003 David Holmes and Doug Lea

58

Containment of Unsafe Objects

- Suppose `Statistics` class was written as follows:


```
public static class Statistics { // Mutable!
    public long requests;
    public double avgTime;
    public Statistics(long requests, double avgTime) {
        this.requests = requests; this.avgTime = avgTime;
    }
}
```

 - ◆ Fields are **public** and **mutable!**
 - Therefore instances **can not** be shared
- Can be safely contained within a `WebServer` instance


```
private final Statistics stats = new Statistics(0,0.0);
public synchronized Statistics getStatistics() {
    return new Statistics(stats.requests, stats.avgTime);
}
private void processRequest(Socket sock) throws IOException {
    // ....
    synchronized(this) {
        double total = stats.avgTime*stats.requests + elapsed;
        stats.avgTime = total / (++stats.requests);
    }
}
```

 - Can't expose mutable state so we make **copies** of it

© 2003 David Holmes and Doug Lea

59

This document was created with Win2PDF available at <http://www.daneprairie.com>.
The unregistered version of Win2PDF is for evaluation or non-commercial use only.