

Contents

Chapter 1 The Architecture of Threads

The Problems with Threads

- All Nontrivial Java Programs Are Multithreaded

- Java's Thread Support Is Not Platform Independent

Threads and Processes

Thread Safety and Synchronization

- Synchronization Is Expensive

- Avoiding Synchronization

Concurrency, or How Can You Be Two Place at Once (When You're Really Nowhere at All)

Get Your Priorities Straight

Cooperate!

- The Cooperative Multithreading Model

- The Preemptive Multithreading Model

- Mapping Kernel Threads to User Processes

Wrapping Up

Chapter 2 The Perils of Multithreaded Programming

Monitors and Exclusion Semaphores (Mutex)

Race Conditions and Spin Locks

- The Spin lock Class

Threads Are Not Objects

Deadlock

Get out the Magnifying Glass

Nested-Monitor Lockout

- Synchronization Wrappers

Time Out!

- A Digression on Style

Why Is **suspend()** Deprecated?

Deadlock on a Blocking I/O Operation

Stopping Threads

Starvation and Synchronizing **run()**

The **volatile** Keyword

Exceptions and Threads

Conclusion

Chapter 3 The Mutex and Lock Management

When **synchronized** Isn't Good Enough
Handling Granularity with **synchronized**
Roll Your Own Semaphores: The **Semaphore** Interface
Managing Semaphores and Deadlock-Resistant Locking
 A Digression: Booch Utilities and Strategy
Implementing a Manageable Mutex Class

Chapter 4 Condition Variables and Counting Semaphores

Condition Variables
 Waiting for the Electrician (or Somebody Like Him):
 Condition Variables vs. *wait ()*
 Send in the Cavalry: Using a Condition Variable
 Implementing a Condition Variable
Condition Sets: Waiting for Multiple Conditions
Counting Semaphores for Managing Resource Pools
Wrapping Up

Chapter 5 Timers, Alarms, and Swing Thread Safety

Why a Timer?
Swingin' Threads: Swing Isn't Thread Safe
 The *invokeLater()* and *invokeAndWait()* Methods
Using the Swing **Timer**
 So, How Does It Work?
 Why Use a Swing Timer (or Not)
Roll Your Own Timer: Implementing the **Alarm** Class
 The Static Structure of an *Alarm*
 Dissecting a Notification
 Restarting an *Alarm* (Stopping a Thread)
 Suspending the Clock
 Notifier Problems
 Unit Tests
Summing Up

Chapter 6 Observers and Multicasters

Implementing Observer in a Multithreaded World
Observer-side Problems: Inner-class Synchronization
Notifier-side Problems:
 Notifications in a Multithreaded World
Mysteries of the **AWTEventMulticaster**
 Immutable Objects and Blank Finals
 Using the Multicaster
Building a Multicaster

Chapter 7 Singletons, Critical Sections, and Reader/Writer Locks

Critical Sections, Singletons, and the “Class Object”..

Static Members

Singletons

Critical Sections, Doubled-checked Locking, and Cache-related Problems in Multiple-CPU Machines

The *Std* Class: An Example of Singleton

Closing Singletons

Reader/Writer Locks

It’s a Wrap

Chapter 8 Threads in an Object-Oriented World..

Modeling Threads in Object-Oriented Systems

Synchronous vs. Asynchronous Messages

Implementing Asynchronous Messages

Using Thread-per-Method

An Exceptional Problem

Thread Pools and Blocking Queues

Blocking Queues

Pooling Threads

Passing Arguments to the Operation

Using Introspection for Runnable Objects that Pass Arguments

Implementing the *Thread_pool*

Putting the Pool to Work

Sockets and Thread Pools

Conclusion

Chapter 9 Object-Oriented Threading Architectures

Reactors and Active Objects

Synchronous Dispatching and Round-Robin Scheduling:

Reactors and Proactors

Asynchronous Dispatching: Active Objects

A General Solution

Detangling Console Output

That’s It

Chapter 10 If I Were King: Fixing Java's Threading Problems

The Task

Improvements to **synchronized**

Improvements to **wait()** and **notify()**

Fixing the **Thread** Class

Inter-Thread Coordination

Internal Support for Reader/Writer Locks

Access to Partially Constructed Objects Should Be Illegal

Volatile Should Always Work as Expected

Access Issues

- Immutability

- Instance-Level Access of Class-Level Fields

Singleton Destruction

Abrupt Shut Down of Daemon Threads

Bring Back the **stop()**, **suspend()**, and **resume()** Methods

Blocking I/O Should Work Correctly

The **ThreadGroup** Class

Wrapping Up

Index



<http://www.springer.com/978-1-893115-10-1>

Taming Java Threads

Holub, A.

2000, X, 300 p. 104 illus., Softcover

ISBN: 978-1-893115-10-1

A product of Apress