Contents

Listings xii

Preface xvii

1 Introduction 1
1.1 A (very) brief history of concurrency 1
1.2 Benefits of threads 3
1.3 Risks of threads 5
1.4 Threads are everywhere 9

I Fundamentals 13

2 Thread Safety 15
2.1 What is thread safety? 17
2.2 Atomicity 19
2.3 Locking 23
2.4 Guarding state with locks 27
2.5 Liveness and performance 29

3 Sharing Objects 33
3.1 Visibility 33
3.2 Publication and escape 39
3.3 Thread confinement 42
3.4 Immutability 46
3.5 Safe publication 49

4 Composing Objects 55
4.1 Designing a thread-safe class 55
4.2 Instance confinement 58
4.3 Delegating thread safety 62
4.4 Adding functionality to existing thread-safe classes 71
4.5 Documenting synchronization policies 74

5 **Building Blocks** 79
5.1 Synchronized collections . . . 79
5.2 Concurrent collections . . . 84
5.3 Blocking queues and the producer-consumer pattern . . . 87
5.4 Blocking and interruptible methods . . . 92
5.5 Synchronizers . . . 94
5.6 Building an efficient, scalable result cache . . . 101

II Structuring Concurrent Applications 111

6 **Task Execution** 113
6.1 Executing tasks in threads . . . 113
6.2 The `Executor` framework . . . 117
6.3 Finding exploitable parallelism . . . 123

7 **Cancellation and Shutdown** 135
7.1 Task cancellation . . . 135
7.2 Stopping a thread-based service . . . 150
7.3 Handling abnormal thread termination . . . 161
7.4 JVM shutdown . . . 164

8 **Applying Thread Pools** 167
8.1 Implicit couplings between tasks and execution policies . . . 167
8.2 Sizing thread pools . . . 170
8.3 Configuring `ThreadPoolExecutor` . . . 171
8.4 Extending `ThreadPoolExecutor` . . . 179
8.5 Parallelizing recursive algorithms . . . 181

9 **GUI Applications** 189
9.1 Why are GUIs single-threaded? . . . 189
9.2 Short-running GUI tasks . . . 192
9.3 Long-running GUI tasks . . . 195
9.4 Shared data models . . . 198
9.5 Other forms of single-threaded subsystems . . . 202

III Liveness, Performance, and Testing 203

10 **Avoiding Liveness Hazards** 205
10.1 Deadlock . . . 205
10.2 Avoiding and diagnosing deadlocks . . . 215
10.3 Other liveness hazards . . . 218

11 **Performance and Scalability** 221
11.1 Thinking about performance . . . 221
11.2 Amdahl's law . . . 225
11.3 Costs introduced by threads . . . 229
11.4 Reducing lock contention . . . 232

11.5 Example: Comparing Map performance 242
11.6 Reducing context switch overhead 243

12 Testing Concurrent Programs 247
12.1 Testing for correctness 248
12.2 Testing for performance 260
12.3 Avoiding performance testing pitfalls 266
12.4 Complementary testing approaches 270

IV Advanced Topics 275

13 Explicit Locks 277
13.1 Lock and ReentrantLock 277
13.2 Performance considerations 282
13.3 Fairness 283
13.4 Choosing between synchronized and ReentrantLock 285
13.5 Read-write locks 286

14 Building Custom Synchronizers 291
14.1 Managing state dependence 291
14.2 Using condition queues 298
14.3 Explicit condition objects 306
14.4 Anatomy of a synchronizer 308
14.5 AbstractQueuedSynchronizer 311
14.6 AQS in java.util.concurrent synchronizer classes 314

15 Atomic Variables and Nonblocking Synchronization 319
15.1 Disadvantages of locking 319
15.2 Hardware support for concurrency 321
15.3 Atomic variable classes 324
15.4 Nonblocking algorithms 329

16 The Java Memory Model 337
16.1 What is a memory model, and why would I want one? 337
16.2 Publication 344
16.3 Initialization safety 349

A Annotations for Concurrency 353
A.1 Class annotations 353
A.2 Field and method annotations 353

Bibliography 355

Index 359