



FUNCTIONAL AND CONCURRENT PROGRAMMING

Core Concepts and Features

MICHEL
CHARPENTIER

Foreword by
CAY HORSTMANN



Functional and Concurrent Programming

Functional and Concurrent Programming: Core Concepts and Features

Table of Contents

Cover

Half Title

Title Page

Copyright Page

Contents

Foreword by Cay Horstmann

Preface

Acknowledgments

About the Author

Part I: Functional Programming

Chapter 1 Concepts of Functional Programming

1.1 What Is Functional Programming?

1.2 Functions

1.3 From Functions to Functional Programming Concepts

1.4 Summary

Chapter 2 Functions in Programming Languages

2.1 Defining Functions

2.2 Composing Functions

2.3 Functions Defined as Methods

2.4 Operators Defined as Methods

2.5 Extension Methods

2.6 Local Functions

Table of Contents

2.7 Repeated Arguments

2.8 Optional Arguments

2.9 Named Arguments

2.10 Type Parameters

2.11 Summary

Chapter 3 Immutability

3.1 Pure and Impure Functions

3.2 Actions

3.3 Expressions Versus Statements

3.4 Functional Variables

3.5 Immutable Objects

3.6 Implementation of Mutable State

3.7 Functional Lists

3.8 Hybrid Designs

3.9 Updating Collections of Mutable/Immutable Objects

3.10 Summary

Chapter 4 Case Study: Active{Passive Sets

4.1 Object-Oriented Design

4.2 Functional Values

4.3 Functional Objects

4.4 Summary

Chapter 5 Pattern Matching and Algebraic Data Types

5.1 Functional Switch

5.2 Tuples

5.3 Options

5.4 Revisiting Functional Lists

5.5 Trees

5.6 Illustration: List Zipper

5.7 Extractors

Table of Contents

5.8 Summary

Chapter 6 Recursive Programming

6.1 The Need for Recursion

6.2 Recursive Algorithms

6.3 Key Principles of Recursive Algorithms

6.4 Recursive Structures

6.5 Tail Recursion

6.6 Examples of Tail Recursive Functions

6.7 Summary

Chapter 7 Recursion on Lists

7.1 Recursive Algorithms as Equalities

7.2 Traversing Lists

7.3 Returning Lists

7.4 Building Lists from the Execution Stack

7.5 Recursion on Multiple/Nested Lists

7.6 Recursion on Sublists Other Than the Tail

7.7 Building Lists in Reverse Order

7.8 Illustration: Sorting

7.9 Building Lists Efficiently

7.10 Summary

Chapter 8 Case Study: Binary Search Trees

8.1 Binary Search Trees

8.2 Sets of Integers as Binary Search Trees

8.3 Implementation Without Rebalancing

8.4 Self-Balancing Trees

8.5 Summary

Chapter 9 Higher-Order Functions

9.1 Functions as Values

9.2 Currying

Table of Contents

- 9.3 Function Literals
- 9.4 Functions Versus Methods
- 9.5 Single-Abstract-Method Interfaces
- 9.6 Partial Application
- 9.7 Closures
- 9.8 Inversion of Control
- 9.9 Summary

Chapter 10 Standard Higher-Order Functions

- 10.1 Functions with Predicate Arguments
- 10.2 map and foreach
- 10.3 atMap
- 10.4 fold and reduce
- 10.5 iterate, tabulate, and unfold
- 10.6 sortWith, sortBy, maxBy, and minBy
- 10.7 groupBy and groupMap
- 10.8 Implementing Standard Higher-Order Functions
- 10.9 foreach, map, atMap, and for-Comprehensions
- 10.10 Summary

Chapter 11 Case Study: File Systems as Trees

- 11.1 Design Overview
- 11.2 A Node-Searching Helper Function
- 11.3 String Representation
- 11.4 Building Trees
- 11.5 Querying
- 11.6 Navigation
- 11.7 Tree Zipper
- 11.8 Summary

Chapter 12 Lazy Evaluation

- 12.1 Delayed Evaluation of Arguments

Table of Contents

- 12.2 By-Name Arguments
- 12.3 Control Abstraction
- 12.4 Internal Domain-Specific Languages
- 12.5 Streams as Lazily Evaluated Lists
- 12.6 Streams as Pipelines
- 12.7 Streams as Infinite Data Structures
- 12.8 Iterators
- 12.9 Lists, Streams, Iterators, and Views
- 12.10 Delayed Evaluation of Fields and Local Variables
- 12.11 Illustration: Subset-Sum
- 12.12 Summary

Chapter 13 Handling Failures

- 13.1 Exceptions and Special Values
- 13.2 Using Option
- 13.3 Using Try
- 13.4 Using Either
- 13.5 Higher-Order Functions and Pipelines
- 13.6 Summary

Chapter 14 Case Study: Trampolines

- 14.1 Tail-Call Optimization
- 14.2 Trampolines for Tail-Calls
- 14.3 Tail-Call Optimization in Java
- 14.4 Dealing with Non-Tail-Calls
- 14.5 Summary

A Brief Interlude

Chapter 15 Types (and Related Concepts)

- 15.1 Typing Strategies
- 15.2 Types as Sets
- 15.3 Types as Services
- 15.4 Abstract Data Types

Table of Contents

- 15.5 Type Inference
- 15.6 Subtypes
- 15.7 Polymorphism
- 15.8 Type Variance
- 15.9 Type Bounds
- 15.10 Type Classes
- 15.11 Summary

Part II: Concurrent Programming

Chapter 16 Concepts of Concurrent Programming

- 16.1 Non-sequential Programs
- 16.2 Concurrent Programming Concepts
- 16.3 Summary

Chapter 17 Threads and Nondeterminism

- 17.1 Threads of Execution
- 17.2 Creating Threads Using Lambda Expressions
- 17.3 Nondeterminism of Multithreaded Programs
- 17.4 Thread Termination
- 17.5 Testing and Debugging Multithreaded Programs
- 17.6 Summary

Chapter 18 Atomicity and Locking

- 18.1 Atomicity
- 18.2 Non-atomic Operations
- 18.3 Atomic Operations and Non-atomic Composition
- 18.4 Locking
- 18.5 Intrinsic Locks
- 18.6 Choosing Locking Targets
- 18.7 Summary

Chapter 19 Thread-Safe Objects

- 19.1 Immutable Objects
- 19.2 Encapsulating Synchronization Policies

Table of Contents

- 19.3 Avoiding Reference Escape
- 19.4 Public and Private Locks
- 19.5 Leveraging Immutable Types
- 19.6 Thread-Safety
- 19.7 Summary

Chapter 20 Case Study: Thread-Safe Queue

- 20.1 Queues as Pairs of Lists
- 20.2 Single Public Lock Implementation
- 20.3 Single Private Lock Implementation
- 20.4 Applying Lock Splitting
- 20.5 Summary

Chapter 21 Thread Pools

- 21.1 Fire-and-Forget Asynchronous Execution
- 21.2 Illustration: Parallel Server
- 21.3 Different Types of Thread Pools
- 21.4 Parallel Collections
- 21.5 Summary

Chapter 22 Synchronization

- 22.1 Illustration of the Need for Synchronization
- 22.2 Synchronizers
- 22.3 Deadlocks
- 22.4 Debugging Deadlocks with Thread Dumps
- 22.5 The Java Memory Model
- 22.6 Summary

Chapter 23 Common Synchronizers

- 23.1 Locks
- 23.2 Latches and Barriers
- 23.3 Semaphores
- 23.4 Conditions

Table of Contents

23.5 Blocking Queues

23.6 Summary

Chapter 24 Case Study: Parallel Execution

24.1 Sequential Reference Implementation

24.2 One New Thread per Task

24.3 Bounded Number of Threads

24.4 Dedicated Thread Pool

24.5 Shared Thread Pool

24.6 Bounded Thread Pool

24.7 Parallel Collections

24.8 Asynchronous Task Submission Using Conditions

24.9 Two-Semaphore Implementation

24.10 Summary

Chapter 25 Futures and Promises

25.1 Functional Tasks

25.2 Futures as Synchronizers

25.3 Timeouts, Failures, and Cancellation

25.4 Future Variants

25.5 Promises

25.6 Illustration: Thread-Safe Caching

25.7 Summary

Chapter 26 Functional-Concurrent Programming

26.1 Correctness and Performance Issues with Blocking

26.2 Callbacks

26.3 Higher-Order Functions on Futures

26.4 Function atMap on Futures

26.5 Illustration: Parallel Server Revisited

26.6 Functional-Concurrent Programming Patterns

26.7 Summary

Table of Contents

Chapter 27 Minimizing Thread Blocking

- 27.1 Atomic Operations
- 27.2 Lock-Free Data Structures
- 27.3 Fork/Join Pools
- 27.4 Asynchronous Programming
- 27.5 Actors
- 27.6 Reactive Streams
- 27.7 Non-blocking Synchronization
- 27.8 Summary

Chapter 28 Case Study: Parallel Strategies

- 28.1 Problem Definition
- 28.2 Sequential Implementation with Timeout
- 28.3 Parallel Implementation Using `invokeAny`
- 28.4 Parallel Implementation Using `CompletionService`
- 28.5 Asynchronous Implementation with Scala Futures
- 28.6 Asynchronous Implementation with `CompletableFuture`
- 28.7 Caching Results from Strategies
- 28.8 Summary

Appendix A Features of Java and Kotlin

- A.1 Functions in Java and Kotlin
- A.2 Immutability
- A.3 Pattern Matching and Algebraic Data Types
- A.4 Recursive Programming
- A.5 Higher-Order Functions
- A.6 Lazy Evaluation
- A.7 Handling Failures
- A.8 Types
- A.9 Threads
- A.10 Atomicity and Locking

Table of Contents

A.11 Thread-Safe Objects

A.12 Thread Pools

A.13 Synchronization

A.14 Futures and Functional-Concurrent Programming

A.15 Minimizing Thread Blocking

Glossary

A

C

D

E

F

G

H

I

J

L

M

N

O

P

Q

R

S

T

V

Index