Undergraduate Topics in Computer Science

Series Editor
Ian Mackie

Advisory Board
Samson Abramsky, University of Oxford, Oxford, UK
Karin Breitman, Pontifical Catholic University of Rio de Janeiro, Rio de Janeiro, Brazil
Chris Hankin, Imperial College London, London, UK
Dexter C. Kozen, Cornell University, Ithaca, USA
Andrew Pitts, University of Cambridge, Cambridge, UK
Hanne Riis Nielson, Technical University of Denmark, Kongens Lyngby, Denmark
Steven S. Skiena, Stony Brook University, Stony Brook, USA
Iain Stewart, University of Durham, Durham, UK

Undergraduate Topics in Computer Science (UTiCS) delivers high-quality instructional content for undergraduates studying in all areas of computing and information science. From core foundational and theoretical material to final-year topics and applications, UTiCS books take a fresh, concise, and modern approach and are ideal for self-study or for a one- or two-semester course. The texts are all authored by established experts in their fields, reviewed by an international advisory board, and contain numerous examples and problems. Many include fully worked solutions.

More information about this series at http://www.springer.com/series/7592

Bhim P. Upadhyaya

Programming with Scala

Language Exploration

Springer

Bhim P. Upadhyaya
Carnegie Mellon University – Silicon Valley
NASA Ames Research Park
Bldg. 23, Moffett Field
CA 94035
USA

ISSN 1863-7310 ISSN 2197-1781 (electronic)
Undergraduate Topics in Computer Science
ISBN 978-3-319-69367-5 ISBN 978-3-319-69368-2 (eBook)
https://doi.org/10.1007/978-3-319-69368-2

Library of Congress Control Number: 2017955281

© Springer International Publishing AG 2017
This work is subject to copyright. All rights are reserved by the Publisher, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilms or in any other physical way, and transmission or information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed.
The use of general descriptive names, registered names, trademarks, service marks, etc. in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant protective laws and regulations and therefore free for general use.
The publisher, the authors and the editors are safe to assume that the advice and information in this book are believed to be true and accurate at the date of publication. Neither the publisher nor the authors or the editors give a warranty, express or implied, with respect to the material contained herein or for any errors or omissions that may have been made. The publisher remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Printed on acid-free paper

This Springer imprint is published by Springer Nature
The registered company is Springer International Publishing AG
The registered company address is: Gewerbestrasse 11, 6330 Cham, Switzerland

To the United Nations
-Bhim

Preface

We are living in a very interesting time, in terms of technological advancements among other things. In the last 60 years, humanity has made tremendous progress in the field of computing. High level programming languages appeared in the late 1950s, but there were no microprocessors available. One can imagine how tedious it could be to program. With the invention of microprocessors in the late 1960s, this field got a great boost. Along with hardware developments, new high level programming languages started emerging. Among those are Pascal, Smalltalk, C, etc. Some of these languages are still widely used. One of the interesting facts about these high level languages is that they had niche application areas, even though they were widely projected as general purpose languages, including in the textbooks. For example, Pascal was leaning toward education, C was primarily developed to ease operating systems developments, Fortran was for scientific computing, etc. Whatever the domains, each of these languages has shaped the history of computing.

Today, shapes, sizes, and capacities of computing devices are much different than 60 years ago. In the last 20 years, computing has shifted from localized client-server environments to world wide client-server environments. There are several catalysts for this change, including the invention of the world wide web, the advancements in integrated circuits, and the innovations in device manufacturing. One of the catalysts, which doesn't get enough credit, is the advancements of programming languages. Developers are truly enabled by language innovations. The innovation that JVM brought, has greatly impacted software engineering in general, and Internet based applications development in particular. Being able to program, without worrying about incompatibilities with hundreds of vendor specific architectures, is a great relief for software engineers.

One of the important aspects of advancements, not discussed enough, in industrial settings is knowledge evolution. I have had the privilege of working for the world's largest (non-profit) organization, the United Nations, which gave me an opportunity to work with people from almost every major cultural background. Also I have worked for some of the largest and finest for-profit organizations in the world. In addition to this, I was fortunate enough to be part of some of the finest universities in the world, either through academic programs, or by working directly with academicians in unique settings. This experience allows me to make some inferences on knowledge evolution, specially in the field of software engineering. The first inference is that changing a programming language is not pleasant, specially for professional programmers. The second inference is that adapting a new language to the level that one can sell the skills has strong limitations. The third inference is that the popularity of a particular language is strongly affected by the industry–academic loop, directly or indirectly. To be precise, how a language is taught, where it is taught, what kind of learning materials are available, etc., determine how popular a language will become.

Dr. Martin Odersky, the creator of Scala, has done a great job of a language innovation. Scala not only provides an opportunity to program in multiple paradigms, but also makes developers more productive with today's computing infrastructure. Most of the earlier languages were not designed to program in distributed environments. Also most of them were not designed to evolve. One of the fascinating aspects of Scala is that it can be grown, based on developers' needs. Remember, for every genius, there is a limit to how many languages one can become expert in. It is pretty much like the case of natural languages. We can recall ourselves, how good we are in our mother tongue and how good we are in other languages. Do we struggle? The Scala creator has gifted a beautiful solution to the world of computer programmers.

With this book, I have tried to enrich the learning aspect of Scala. I strongly believe that Scala can be a great first programming language. There is no need to lean on any other programming language in order to learn it, except the environment, which is JVM for a good reason. The approach that I have taken in this book was primarily inspired by my first hand observations of how professional programmers master a new technology. Also it is partly influenced by my experience of teaching undergraduate computer science and engineering students, over half a decade. I have tried to present complete and runnable programs, whenever possible. In the last 12 years, I noticed professional programmers learning faster by tweaking existing programs. This was true for my undergraduate students as well. Tweaking allowed learners to build self-confidence, which I found was the basis for the majority of learners for their long term pursuits. This book can be used, both at an undergraduate level and at a graduate level, to teach a first programming course. Also it is a great companion for professional programmers planning to switch to Scala.

Each chapter has review questions to reinforce your learning, which makes you ready for problem solving. You will find yourself adequate and self-confident, thereby keeping you away from the path of frustration. I have seen many great professional programmers being frustrated while learning a new language. Also each chapter has problems to solve. These problems are much closer to what you will need when you embark on your career as a professional Scala programmer. The process of solving problems expands your knowledge boundary; you are moving from reinforcing to developing salable skills.

Sunnyvale, California
June 2017

Bhim P. Upadhyaya

Contents

1 Introduction to Computing 1

1.1 Introduction to Computers 3

1.1.1 Basic Components 3

1.1.2 Operation 4

1.2 Operating Systems 5

1.3 Programming Languages 6

1.4 Introduction to Scala 9

1.5 Program Attributes 12

1.6 Conclusion 13

1.7 Review Questions 14

1.8 Problems 14

1.9 Answers to Review Questions 15

1.10 Solutions to Problems 16

2 Scala Fundamentals 17

2.1 Literals 17

2.1.1 Character Literals 17

2.1.2 String Literals 18

2.1.3 Integer Literals 19

2.1.4 Floating Point Literals 19

2.1.5 Boolean Literals 20

2.1.6 Escape Sequences 20

2.1.7 Symbol Literals 21

2.1.8 Other Lexical Elements 21

2.2 Identifiers and Reserved Words 23

2.3 Types 23

2.4 Declarations and Definitions 24

2.5 Expressions 26

2.6 Conclusion 28

2.7 Review Questions 28

2.8 Problems 28

2.9 Answers to Review Questions 29
2.10 Solutions to Problems 29

3 Classes and Objects 31
3.1 Class Members 31
3.2 Class Definitions 33
3.3 Object Definitions 33
3.4 Conclusion 34
3.5 Review Questions 34
3.6 Problems 34
3.7 Answers to Review Questions 34
3.8 Solutions to Problems 35

4 Control Structures 37
4.1 Conditional Expressions 37
4.2 For Expressions 39
4.3 While Loops 42
4.4 Exception Handling 43
4.5 Conclusion 45
4.6 Review Questions 45
4.7 Problems 46
4.8 Answers to Review Questions 46
4.9 Solutions to Problems 47

5 Operators 51
5.1 Operators as Methods 51
5.2 Arithmetic Operators 52
5.3 Relational and Logical Operators 54
5.4 Bitwise Operators 55
5.5 Operator Precedence and Associativity 56
5.6 Conclusion 57
5.7 Review Questions 58
5.8 Problems 58
5.9 Answers to Review Questions 58
5.10 Solutions to Problems 59

6 Data Input and Output 63
6.1 Regular Expressions 63
6.2 Single Character Input 66
6.3 Single Character Output 67
6.4 Reading From a File 68
6.5 Writing to a File 69
6.6 Navigating Directories 70
6.7 Conclusion 72
6.8 Review Questions 72
6.9 Problems 72

6.10 Answers to Review Questions 73
6.11 Solutions to Problems 73

7 Inheritance and Composition 77
7.1 Extending Classes 77
7.2 Overriding Methods and Fields 79
7.3 Abstract Classes 79
7.4 Invoking Super Class Constructors 80
7.5 Polymorphism and Dynamic Binding 81
7.6 Composition 82
7.7 Conclusion 83
7.8 Review Questions 83
7.9 Problems 84
7.10 Answers to Review Questions 84
7.11 Solutions to Problems 84

8 Traits 87
8.1 Traits as Interfaces 87
8.2 Construction Order and Linearizing 88
8.3 Trait Members 90
8.4 Multiple Inheritance 91
8.5 Traits with Implementations 92
8.6 Conclusion 93
8.7 Review Questions 94
8.8 Problems 95
8.9 Answers to Review Questions 96
8.10 Solutions to Problems 96

9 Functions 99
9.1 Functions as Methods 100
9.2 Anonymous Functions 101
9.3 Functions as Values 102
9.4 Function Parameters 102
9.5 Higher Order Functions 103
9.6 Closures 104
9.7 Currying 104
9.8 Writing New Control Structures 106
9.9 Conclusion 107
9.10 Review Questions 107
9.11 Problems 108
9.12 Answers to Review Questions 108
9.13 Solutions to Problems 109

10 Pattern Matching 111
10.1 Case Classes 111
10.2 Sealed Classes 112
10.3 Variable Patterns 113
10.4 Type Patterns 114
10.5 Literal Patterns 115
10.6 Constructor Patterns 116
10.7 Tuple Patterns 117
10.8 Extractor Patterns 118
10.9 Sequence Patterns 119
10.10XML Patterns 120
10.11Conclusion 121
10.12Review Questions 121
10.13Problems 122
10.14Answers to Review Questions 122
10.15Solutions to Problems 123

11 List Processing 125
11.1 List Construction 125
11.2 Operations 126
11.3 Patterns 129
11.4 List Class 130
11.5 List Object 132
11.6 Conclusion 133
11.7 Review Questions 134
11.8 Problems 134
11.9 Answers to Review Questions 135
11.10Solutions to Problems 135

12 The Scala Collections Framework 137
12.1 Mutable versus Immutable Collections 137
12.2 Sets 138
12.3 Maps 139
12.4 Sequences 140
12.5 Tuples 143
12.6 Conclusion 144
12.7 Review Questions 144
12.8 Problems 146
12.9 Answers to Review Questions 146
12.10Solutions to Problems 147

13 Actors 151
13.1 The Components of Actors 151
13.2 Creating Actors 152
13.3 Sending and Receiving Messages 153

13.4 Life Cycle 155
13.5 Child Actors 156
13.6 Monitoring 159
13.7 Conclusion 160
13.8 Review Questions 161
13.9 Problems 161
13.10 Answers to Review Questions 162
13.11 Solutions to Problems 162

14 XML Processing 165
14.1 XML Literals 165
14.2 Data Extraction 167
14.3 Pattern Matching 170
14.4 Serialization and Deserialization 171
14.5 Loading and Saving 173
14.6 Conclusion 175
14.7 Review Questions 175
14.8 Problems 176
14.9 Answers to Review Questions 176
14.10 Solutions to Problems 177

15 Parsing 179
15.1 Lexical Analysis and Parsing 179
15.2 Creating and Running a Parser 180
15.3 Regular Expression Parser 181
15.4 JSON Parser 182
15.5 Error Handling 184
15.6 Conclusion 185
15.7 Review Questions 185
15.8 Problems 186
15.9 Answers to Review Questions 186
15.10 Solutions to Problems 187

References 189

Index 191

List of Figures

1.1 Sample Assembly Language Code Snippet 6
1.2 Sample Pascal Program 7
1.3 Sample C Program 7
1.4 Sample Java Program 8
1.5 Sample Scala Program 8
1.6 Program to Add Two Pre-defined Numbers 10
1.7 Program to Add Two Numbers 10
1.8 Program for Biological Taxonomy Information Base 11

2.1 Symbol literals 21
2.2 Types Demonstration 24
2.3 Value Declaration and Definition 25
2.4 Variable Declaration and Definition 25
2.5 Type Definition 26
2.6 Expressions 27

3.1 Circle Application 32

4.1 If Expression 38
4.2 If Expression - Positive Difference 39
4.3 Prime Number Detection 40
4.4 Prime Number Detection using While 42
4.5 Exception Handling 43
4.6 Exception Handling with Custom Throw 45

5.1 Operators as Methods 52
5.2 Custom Operators as Methods 52

6.1 Regular Expressions 65
6.2 Character Level Input and Output 66
6.3 Character Level Output 67
6.4 Reading from a Text File 69

6.5 Writing to a Text File 69
6.6 Directory Navigation 70
6.7 System Commands 71

7.1 Extending a Class 78
7.2 Overriding a Method 79
7.3 Abstract Class 80
7.4 Polymorphism and Dynamic Binding 82
7.5 Composition 83

8.1 Trait as an Interface 88
8.2 Construction Order and Linearizing 89
8.3 Multiple Inheritance 91
8.4 Traits with Implementation 93

9.1 Functions as Methods 100
9.2 Writing a New Control Structure 107

10.1 Case Classes 112
10.2 Sealed Class 113
10.3 Variable Pattern 114
10.4 Type Pattern 114
10.5 Literal Pattern 115
10.6 Constructor Pattern 116
10.7 Tuple Pattern 117
10.8 Extractor Pattern 118
10.9 Sequence Pattern 119
10.10XML Pattern 121

11.1 List Extraction 130

13.1 Simple Actor 152
13.2 Sending and Receiving Messages 154
13.3 Actor Life Cycle 156
13.4 Child Actors 158
13.5 Actor Monitoring 160

14.1 XML with Scala Code 167
14.2 XML Pattern Matching – Any Sequence 170
14.3 XML Serialization 171
14.4 XML Deserialization 172
14.5 Loading XML 173
14.6 Saving XML 174

15.1 Simple Product Expression Parser 180
15.2 Regular Expression Parser 182

15.3 JSON Parser 183
15.4 Parser Error Handling 185

List of Tables

1.1 Sample Biological Taxonomy Data 2

2.1 Integer Literals and Ranges 19
2.2 Reserved Words 23

5.1 Truth Table for Logical Operators 55
5.2 Truth Table for Bitwise Operators 55
5.3 Operator Precedence 57

6.1 Regular Expression Notations 64

15.1 Sample Tokenization 179
15.2 Parser Combinators 184