

Ada for Software Engineers (Second Edition with Ada 2005)

Mordechai Ben-Ari

Ada for Software Engineers (Second Edition with Ada 2005)

 Springer

Mordechai Ben-Ari, BSc, MSc, PhD
Weizmann Institute of Science
Rehovot 76100
Israel

ISBN: 978-1-84882-313-6 e-ISBN: 978-1-84882-314-3

British Library Cataloguing in Publication Data
A catalogue record for this book is available from the British Library

Library of Congress Control Number: 2009921268

First published 1998

© John Wiley & Sons 1998

© Springer-Verlag London Limited 2009

Apart from any fair dealing for the purposes of research or private study, or criticism or review, as permitted under the Copyright, Designs and Patents Act 1988, this publication may only be reproduced, stored or transmitted, in any form or by any means, with the prior permission in writing of the publishers, or in the case of reprographic reproduction in accordance with the terms of licenses issued by the Copyright Licensing Agencies. Enquiries concerning reproduction outside those terms should be sent to the publishers.

The use of registered names, trademarks, etc., in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant laws and regulations and therefore free for general use. The publisher makes no representation, express or implied, with regard to the accuracy of the information contained in this book and cannot accept any legal responsibility or liability for any errors or omissions that may be made.

Printed on acid-free paper

Springer Science+Business Media
springer.com

Preface

Albert Einstein once said everything should be made as simple as possible, but not simpler. Einstein could have been talking about programming languages, as the landscape is strewn with “simple” languages that, several versions later, have 500-page reference manuals!

The truth is that we expect a lot of our programming languages. We demand support for encapsulation and abstraction, type checking and exception handling, polymorphism and more. Ada, unlike other languages which grew by the gradual addition of features, was designed as a coherent programming language for complex software systems. *Ada for Software Engineers (ASE)* is written to equip you with the knowledge necessary to use the Ada programming language to develop software systems.

Although Ada never achieved the popularity of Java and the C-something languages, it remains the programming language of choice for reliable software. Every time you step on an airplane or a fast train, you are quite literally trusting your life to software written in Ada. Given the low level of reliability of much of the software we use daily, I can only regret that the use of Ada is not more widespread, and I hope that this book will encourage more software engineers to give Ada a chance.

Intended audience

The book is intended for software engineers making the transition to Ada, and for upper-level undergraduate and graduate students, including those who have had the good fortune to study Ada as their first programming language. No specific knowledge of Ada is assumed; the prerequisites are a basic knowledge of computer science and computer systems, and at least two years of programming experience. As the title implies, if you are a software engineer or training to become one, this book is for you.

Structure of the book

The complexity of modern programming languages leaves textbook writers with a painful dilemma: either gloss over the gory details, or write books that are heavy enough to be classified as lethal weapons. Is there another way? A concise description of Ada is freely available: the *Ada Reference Manual (ARM)* [7], which is the document defining the language standard. The ARM has a reputation for being ponderous reading meant for “language lawyers.” Nevertheless, I believe that—with a bit of guidance—software engineers can learn to read most of the ARM. *ASE* is based upon two premises: (a) it is best to learn the individual language constructs within the context of actual programs, and (b) if the learning is based upon the terminology and concepts used in the language standard, you will be able to use the ARM as your primary reference manual.

The Ada language will be taught using a few relatively large case studies, rather than a large number of small examples each crafted to demonstrate a particular construct or rule. Experienced programmers know that the key to mastering a programming language is not to memorize the syntax and semantics of individual constructs, but to learn how to integrate the constructs into language-specific paradigms. We will need to gloss over details when explaining a case study; rest assured that everything will eventually be explained. Certain sections are marked with an asterisk and can be omitted during your initial study of Ada. This material is not necessarily more difficult, but you can’t learn everything at once, and these are topics that can be left for your second reading of the book.

The chapters in the book can be divided into five parts and three appendices. The first two parts contain the core concepts of Ada and should probably be read first and in sequence (skipping the starred sections, of course). The others chapters can be read as necessary.

1. After an introductory chapter, Chapters 2–5 quickly cover elementary language constructs such as statements, subprograms, arrays, records and pointers that should be familiar from your previous programming experience.
2. The second part is based upon the progressive development of a case study demonstrating the Ada constructs for programming-in-the-large (including object-oriented programming): packages and private types (Chapter 6), type extension, inheritance, class-wide types and dynamic polymorphism (Chapter 7–8), and generics (Chapter 9).
3. Chapters 10–14 cover other topics and are more or less independent of the previous ones: exceptions, the type system in depth, access types (pointers), numerics and input–output.
4. The fourth part contains material on program structure (Chapter 15), the container library (Chapter 16), and interfaces (Chapter 17).
5. Chapters 18–22 include topics broadly covered by the term *systems programming*: multitasking, hardware interfaces, and real-time and distributed systems.

A special feature of the book is the comprehensive *Glossary* (Appendix A), which explains the ARM terminology with examples. In addition, discussions in the text always contain references to the relevant paragraphs in the ARM. The *Index of ARM Sections* will be invaluable when you begin to use the ARM as a reference manual.

At the end of each chapter are *Projects* and *Quizzes*. Projects are non-trivial programming exercises, many of which ask you to extend the case studies in the text. Quizzes are *not* routine exercises as the term is usually used; they are intended help you understand the finer points of the Ada language, and can be skipped until you have significant experience programming with Ada. Each quiz is a small Ada program; you are to examine it and decide what will happen when the program is compiled, and—if the program contains executable statements and compiles successfully—the result of the execution. Appendix B *Hints* refers you to the relevant clauses of the *ARM* from which you should be able to find the answers to the quizzes *before* looking at Appendix C *Answers*. (To save space, **with** and **use** clauses for `Ada.Text_IO`, `Ada.Integer_IO` and `Ada.Exceptions` are omitted.)

Supplementary material and links

All the case studies, quizzes and programs in the book were compiled and executed. The full source code is available in the software archive at:

- <http://www.springer.com/978-1-84882-313-6>.

The programs were developed using the GNAT compiler for Ada 2005. It is available in commercial, academic and free versions; see:

- <https://libre.adacore.com>.

There are two comprehensive websites on Ada:

- ACM Special Interest Group on Ada (SIGAda), <http://www.sigada.org>,
- Ada Resource Association (ARA), <http://www.adaic.com>.

From these websites you can download documents such as the *ARM*, and find links to compiler vendors, support software, relevant conferences, and other resources.

Ada 2005

This book was original written for the Ada 95 version of the language. That edition is out of print and I have used the occasion of the publication of a new version of Ada, Ada 2005, to expand and improve the book. Constructs of Ada 2005 are freely used, but sections that use them are annotated to indicate which constructs are new. The software archive for the previous edition of the book will remain freely available.

Acknowledgments

First edition: I would like to thank Michael Feldman, Kevlin Henney, Richard Riehle, Reuben Sumner, Tucker Taft and Peter Wegner for reviewing the manuscript, and Simon Pluntree of John Wiley for his support and assistance.

Second edition: I am grateful to Edmond Schonberg for his constant help as I learned Ada 2005. I am indebted to Robert Duff for his comprehensive review that enabled me to correct many bugs in the text. The staff at AdaCore has been extremely helpful answering my queries on the GNAT compiler. I would like to thank Beverley Ford and the staff at Springer for their friendly and efficient handling of the publishing issues that I raised.

Mordechai (Moti) Ben-Ari
Rehovot, 2008

Contents

Preface	v
1 The Language for a Complex World	1
1.1 Programming or software engineering?	1
1.2 Reliable software engineering	1
1.3 Programming languages for software engineering	2
1.4 Ada for software engineering	3
1.5 The development of Ada	3
1.6 The Ada Reference Manual	6
1.7 Case studies	11
2 First Steps in Ada	13
2.1 Case study: country of origin	13
2.2 Library units and context clauses	15
2.3 Input–output	15
2.4 Attributes for string conversion	15
2.5 Statements	16
2.6 Exceptions	18
2.7 Types	19
2.8 Objects	23
2.9 Expressions	23
2.10 Subtypes	25
2.11 Lexical elements	28
3 Subprograms	31
3.1 Parameter modes	32
3.2 Overloading	34
3.3 Parameter associations and default expressions	36
3.4 Operators	37

3.5	Block statement*	38
3.6	Implicit dereferencing*	39
4	Arrays	41
4.1	Case study: fill and justify text	41
4.2	Array types	46
4.3	Constrained array subtypes and objects*	54
4.4	Type conversion for arrays*	55
4.5	Operations on one-dimensional arrays*	55
4.6	The context of array aggregates*	56
5	Elementary Data Structures	59
5.1	Case study: array priority queue	59
5.2	Records	62
5.3	Case study: tree priority queue	63
5.4	Access types	66
6	Packages and Abstract Data Types	73
6.1	Modularization	73
6.2	Case study: priority queue package—version 1	74
6.3	Case study: priority queue package—version 2	79
6.4	Case study: priority queue package—version 3	81
6.5	Case study: priority queue package—version 4	82
6.6	Case study: priority queue package—version 5	86
6.7	Case study: priority queue package—version 6	88
6.8	Nonlimited private types*	89
6.9	Limited types that are not private*	94
6.10	Initialization of limited types*	95
7	Type Extension and Inheritance	99
7.1	Case study: discrete event simulation	100
7.2	Tagged types	101
7.3	Primitive operations	105
7.4	Overriding an operation	106
7.5	The package bodies of the case study	107
7.6	Class-wide types	109
7.7	Dynamic dispatching	113
7.8	Types and packages	117
7.9	Encapsulation and child packages	118

8	Type Extension and Inheritance (Continued)	125
8.1	Designated receiver syntax	125
8.2	Type conversion	126
8.3	Extension aggregates	127
8.4	Abstract types	129
8.5	Null procedures	130
8.6	Overriding indicators	132
8.7	Objects of class-wide type	133
8.8	View conversion and redispaching*	134
8.9	Multiple controlling operands*	135
8.10	Dispatching on the function result*	136
8.11	Indirect derivation*	138
8.12	Freezing*	138
8.13	Implementation of dispatching*	139
9	Generics	143
9.1	Generic declaration and instantiation	144
9.2	The contract model	146
9.3	Generic formal subprogram parameters	148
9.4	Generic formal array types	150
9.5	General access types*	152
9.6	Generic formal objects*	153
9.7	Indefinite type parameters*	154
9.8	Formal package parameters*	155
9.9	Generic children*	162
9.10	The fine print in the contract model*	163
10	Exceptions and Run-Time Checks	171
10.1	Declaring and raising exceptions	171
10.2	Handling exceptions	173
10.3	Propagating exceptions	174
10.4	Package Exceptions*	176
10.5	Re-raising exceptions*	178
10.6	Saving exceptions*	181
10.7	Suppressing checks*	184
10.8	Assertions*	186
11	Composite Types	189
11.1	Characters and strings	189
11.2	Multibyte characters and strings*	190
11.3	Case study: dot2dot	193
11.4	Discriminants	202

11.5	Variant records	204
11.6	Unconstrained types	207
11.7	Discriminants of private types*	208
11.8	Inheriting discriminants*	209
11.9	Untagged derived types*	211
11.10	Untagged derived types and discriminants*	212
12	Access Types	217
12.1	General access types	217
12.2	Access-to-subprogram types	219
12.3	Null exclusions	221
12.4	Accessibility rules	223
12.5	Anonymous access types*	225
12.6	Access parameters	226
12.7	Access discriminants*	230
12.8	Storage pools*	232
12.9	Controlled types*	232
12.10	Mutually dependent types*	236
13	Numeric Types	245
13.1	Basic concepts	245
13.2	Signed integer types	248
13.3	Types versus subtypes	249
13.4	Modular types	250
13.5	Real types	253
13.6	Floating point types	254
13.7	Ordinary fixed point types	257
13.8	Decimal fixed point types*	258
13.9	Fixed point multiplication and division*	262
13.10	Complex numbers*	263
13.11	Advanced concepts*	267
14	Input–Output	277
14.1	Libraries for input–output	277
14.2	Interface with the operation system	279
14.3	Streams*	279
14.4	Generic dispatching constructors*	283
15	Program Structure	287
15.1	Compilation and execution	287
15.2	Compilation and the environment of compilation*	288
15.3	Subunits*	289
15.4	Pragmas	292

15.5	Elaboration*	292
15.6	Renamings	295
15.7	Use type clause	297
15.8	Visibility rules*	298
15.9	Overloading	301
16	Containers	307
16.1	Concepts	308
16.2	Vectors	312
16.3	Doubly-linked lists	313
16.4	Maps	314
16.5	Sets	317
16.6	Indefinite containers	318
17	Interfaces and Multiple Inheritance	323
17.1	Interfaces	324
17.2	Case study: displayable events	326
17.3	Case study: storable interface*	329
17.4	Synchronized interfaces	336
17.5	Generic formal tagged private types*	339
17.6	Generic formal derived types*	341
18	Concurrency	345
18.1	Tasks and protected objects	346
18.2	Rendezvous	352
18.3	Implementation of entry calls	357
18.4	Case study: synchronization with rendezvous and protected objects	358
18.5	Entry families	363
18.6	Protected subprograms	364
18.7	The requeue statement	364
18.8	Additional rules for protected types*	367
19	Concurrency (Continued)	375
19.1	Activation and termination	375
19.2	Exceptions	379
19.3	Time	380
19.4	Time formatting and time zones*	382
19.5	Representation of Time and Duration*	383
19.6	Timed and conditional entry calls*	384
19.7	Asynchronous transfer of control*	387
19.8	Alternatives for selective accept	389
19.9	Case study: concurrent simulation	389
19.10	Tasks as access discriminants*	393

20	Systems Programming	401
20.1	Implementation dependences	401
20.2	Representation items	402
20.3	Interfaces to other languages	404
20.4	Annex C Systems Programming	409
20.5	Machine code*	410
20.6	Interrupts*	411
20.7	Shared variables*	413
20.8	Task identification and attributes*	414
20.9	Detecting task termination*	417
21	Real-Time Systems	423
21.1	Annex D Real-Time Systems	423
21.2	Scheduling and priorities	423
21.3	Task dispatching policies	425
21.4	Base and active priorities	428
21.5	Entry queuing policies	430
21.6	Dynamic priorities*	431
21.7	Priority ceiling locking	431
21.8	Monotonic time	433
21.9	Execution time*	435
21.10	Preemptive abort*	437
21.11	Synchronous task control*	438
21.12	Asynchronous task control*	438
21.13	Tasking restrictions	438
21.14	The Ravenscar profile	439
22	Distributed and High Integrity Systems	451
22.1	Distributed systems	451
22.2	High integrity systems	456
A	Glossary of ARM Terms	459
B	Hints	477
C	Answers	479
	References	487
	Index of ARM Sections	489
	Subject Index	495