

Undergraduate Topics in Computer Science

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.

For other volumes:
<http://www.springer.com/series/7592>

Sarnath Ramnath and Brahma Dathan

Object-Oriented Analysis and Design

 Springer

 Universities Press

Sarnath Ramnath
Department of Computer Science
St. Cloud State University
ECC 139
56303 St. Cloud, Minnesota
USA
rsarnath@stcloudstate.edu

Brahma Dathan
Department of Information
and Computer Science
Metropolitan State University
L118 New Main
700 7th Street East
55106 St. Paul, Minnesota
USA
Brahma.Dathan@metrostate.edu

Series editor
Ian Mackie

Advisory board
Samson Abramsky, University of Oxford, UK
Chris Hankin, Imperial College London, UK
Dexter Kozen, Cornell University, USA
Andrew Pitts, University of Cambridge, UK
Hanne Riis Nielson, Technical University of Denmark, Denmark
Steven Skiena, Stony Brook University, USA
Iain Stewart, University of Durham, UK

A co-publication with the Universities Press (India) Pvt. Ltd, licensed for sale in all countries outside of India, Pakistan, Bhutan, Bangladesh, Sri Lanka, Nepal, The Maldives, Middle East, Malaysia, Indonesia and Singapore. Sold and distributed within these territories by the Universities Press (India) Private Ltd.

Undergraduate Topics in Computer Science ISSN 1863-7310
ISBN 978-1-84996-521-7 e-ISBN 978-1-84996-522-4
DOI 10.1007/978-1-84996-522-4
Springer London Dordrecht Heidelberg New York

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

Library of Congress Control Number: 2010934228

© Universities Press (India) Pvt. Ltd

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 licences issued by the Copyright Licensing Agency. 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 is part of Springer Science+Business Media (www.springer.com)

Contents

<i>Preface</i>	<i>xi</i>
Part I Basic Object-Oriented Concepts	1
1 Introduction	3
1.1 What is Object-Oriented Development?	4
1.2 Key Concepts of Object-Oriented Design	5
1.3 Other Related Concepts	7
1.3.1 Modular design and encapsulation	7
1.3.2 Cohesion and coupling	7
1.3.3 Modifiability and testability	8
1.4 Benefits and Drawbacks of the Paradigm	9
1.5 History	10
1.6 Discussion and Further Reading	11
Exercises	11
2 Basics of Object-Oriented Programming	12
2.1 The Basics	12
2.2 Implementing Classes	15
2.2.1 Constructors	19
2.2.2 Printing an object	22
2.2.3 Static members	23
2.3 Programming with Multiple Classes	24
2.4 Interfaces	28
2.4.1 Implementation of StudentLinkedList	30
2.4.2 Array implementation of lists	33
2.5 Abstract Classes	36
2.6 Comparing Objects for Equality	37
2.7 A Notation for Describing Object-Oriented Systems	39
2.7.1 Class diagrams	42
2.7.2 Use cases and use case diagrams	43
2.7.3 Sequence diagrams	44
2.8 Discussion and Further Reading	47
Exercises	50

3 Relationships between Classes	52
3.1 Association	53
3.1.1 Characteristics of associations	54
3.2 Inheritance	56
3.2.1 An example of a hierarchy	57
3.2.2 Inheriting from an interface	62
3.2.3 Polymorphism and dynamic binding	62
3.2.4 Protected fields and methods	69
3.2.5 The object class	71
3.3 Genericity	71
3.4 Discussion and Further Reading	73
3.4.1 A generalised notion of conformance	75
Exercises	78
4 Language Features for Object-Oriented Implementation	80
4.1 Organising the Classes	80
4.1.1 Creating the files	81
4.1.2 Packages	81
4.1.3 Protected access and package access	82
4.2 Collection Classes	83
4.3 Exceptions	84
4.4 Run-Time Type Identification	86
4.4.1 Reflection: Using the Class object	87
4.4.2 Using the instanceof operator	88
4.4.3 Downcasting	89
4.5 Graphical User Interfaces: Programming Support	90
4.5.1 The basics	90
4.5.2 Event handling	93
4.5.3 More on widgets and layouts	95
4.5.4 Drawing shapes	97
4.5.5 Displaying a piece of text	98
4.6 Long-Term Storage of Objects	98
4.6.1 Storing and retrieving objects	100
4.6.2 Issues in storing and retrieving objects	101
4.6.3 The Java serialization mechanism	104
4.7 Discussion and Further Reading	106
Exercises	109

Part II	Introduction to Object-Oriented Analysis, Design, Implementation and Refactoring	111
5	Elementary Design Patterns	113
5.1	Iterator	114
5.1.1	Iterator implementation	118
5.2	Singleton	121
5.2.1	Subclassing singletons	122
5.3	Adapter	125
5.4	Discussion and Further Reading	130
	Exercises	132
6	Analysing a System	134
6.1	Overview of the Analysis Phase	135
6.2	Stage 1: Gathering the Requirements	136
6.2.1	Case study introduction	137
6.3	Functional Requirements Specification	139
6.3.1	Use case analysis	139
6.4	Defining Conceptual Classes and Relationships	150
6.5	Using the Knowledge of the Domain	158
6.6	Discussion and Further Reading	160
	Exercises	163
7	Design and Implementation	167
7.1	Design	167
7.1.1	Major subsystems	168
7.1.2	Creating the software classes	169
7.1.3	Assigning responsibilities to the classes	171
7.1.4	Class diagrams	182
7.1.5	User interface	188
7.1.6	Data storage	188
7.2	Implementing Our Design	189
7.2.1	Setting up the interface	189
7.2.2	Adding new books	190
7.2.3	Issuing books	191
7.2.4	Printing transactions	194
7.2.5	Placing and processing holds	195
7.2.6	Storing and retrieving the library object	198

7.3 Discussion and Further Reading	202
7.3.1 Conceptual, software and implementation classes	203
7.3.2 Building a commercially acceptable system	204
7.3.3 The facade pattern	205
7.3.4 Implementing singletons	207
7.3.5 Further reading	207
Exercises	208
8 How 'Object-Oriented' is Our Design?	210
8.1 Introduction	210
8.2 A First Example of Refactoring	211
8.2.1 A library that charges fines: Initial solution	211
8.2.2 Refactoring the solution	215
8.3 A Second Look at Remove Books	219
8.4 Using Generics to Refactor Duplicated Code	222
8.4.1 A closer look at the collection classes	222
8.4.2 Instantiating Catalog and MemberList	227
8.5 Discussion and Further Reading	229
Exercises	229
Part III Advanced Concepts in Object-Oriented Design	231
9 Exploring Inheritance	233
9.1 Introduction	233
9.2 Applications of Inheritance	234
9.2.1 Restricting behaviours and properties	234
9.2.2 Abstract superclass	234
9.2.3 Adding features	235
9.2.4 Hiding features of the superclass	236
9.2.5 Combining structural and type inheritance	237
9.3 Inheritance: Some Limitations and Caveats	237
9.3.1 Deep hierarchies	238
9.3.2 Lack of multiple inheritance	238
9.3.3 Changes in the superclass	238
9.3.4 Typing issues: The Liskov substitution principle	239
9.3.5 Addressing the limitations	242
9.4 Type Inheritance	243
9.4.1 A simple example	243
9.4.2 The cloneable interface	244

9.4.3 The runnable interface	248
9.5 Making Enhancements to the Library Class	250
9.5.1 A first attempt	250
9.5.2 Drawbacks of the above approach	254
9.6 Improving the Design	255
9.6.1 Designing the hierarchy	256
9.6.2 Invoking the constructors	258
9.6.3 Distributing the responsibilities	262
9.6.4 Factoring responsibilities across the hierarchy	264
9.7 Consequences of Introducing Inheritance	266
9.7.1 Exception handling	268
9.7.2 Adding new functionality to a hierarchy	269
9.8 Multiple Inheritance	273
9.8.1 Mechanisms for resolving conflicts	276
9.8.2 Repeated inheritance	277
9.8.3 Multiple inheritance in Java	281
9.9 Discussion and Further Reading	282
9.9.1 Design patterns that facilitate inheritance	283
9.9.2 Performance of object-oriented systems	284
Exercises	285
10 Modelling with Finite State Machines	287
10.1 Introduction	287
10.2 A Simple Example	287
10.3 Finite State Modelling	289
10.4 A First Solution to the Microwave Problem	291
10.4.1 Completing the analysis	291
10.4.2 Designing the system	293
10.4.3 The implementation classes	295
10.4.4 A critique of the above design	299
10.5 Using the State Pattern	301
10.5.1 Creating the state hierarchy	302
10.5.2 Implementation	307
10.6 Improving Communication between Objects	310
10.6.1 Loosely coupled communication	310
10.7 Redesign Using the Observer Pattern	312
10.7.1 Communication with the user	313
10.7.2 The improved design	315
10.8 Eliminating the Conditionals	315

10.8.1 Using the Java event mechanism	317
10.8.2 Using the context as a 'switchboard'	320
10.8.3 Implementation	322
10.9 Designing GUI Programs Using the State Pattern	326
10.9.1 Design of a GUI system for the library	326
10.9.2 The context	330
10.10 Discussion and Further Reading	330
10.10.1 Implementing the state pattern	330
10.10.2 Features of the state pattern	331
10.10.3 Consequences of observer	332
10.10.4 Recognising and processing external events	333
10.10.5 Handling the events	334
Exercises	337
11 Interactive Systems and the MVC Architecture	339
11.1 Introduction	339
11.2 The MVC Architectural Pattern	340
11.2.1 Examples	342
11.2.2 Implementation	342
11.2.3 Benefits of the MVC pattern	344
11.3 Analysing a Simple Drawing Program	344
11.3.1 Specifying the requirements	345
11.3.2 Defining the use cases	345
11.4 Designing the System	348
11.4.1 Defining the model	348
11.4.2 Defining the controller	349
11.4.3 Selection and deletion	355
11.4.4 Saving and retrieving the drawing	355
11.5 Design of the Subsystems	356
11.5.1 Design of the model subsystem	356
11.5.2 Design of item and its subclasses	358
11.5.3 Design of the controller subsystem	365
11.5.4 Design of the view subsystem	367
11.6 Getting into the Implementation	370
11.6.1 Item and its subclasses	370
11.6.2 Implementation of the model class	372
11.6.3 Implementation of the controller class	373
11.6.4 Implementation of the view class	375
11.6.5 The driver program	378

11.6.6 A critique of our design	378
11.7 Implementing the Undo Operation	379
11.7.1 Employing the command pattern	383
11.7.2 Implementation	388
11.8 Drawing Incomplete Items	391
11.9 Adding a New Feature	394
11.10 Pattern-Based Solutions	399
11.10.1 Examples of architectural patterns	400
11.11 Discussion and Further Reading	402
11.11.1 Separating the view and the controller	402
11.11.2 The space overhead for the command pattern	403
11.11.3 How to store the items	403
11.11.4 Exercising caution when allowing undo	403
11.11.5 Synchronising updates	404
Exercises	405
12 Designing with Distributed Objects	408
12.1 Client/Server Systems	409
12.1.1 Basic architecture of client/server systems	409
12.2 Java Remote Method Invocation	411
12.2.1 Remote interfaces	413
12.2.2 Implementing a remote interface	413
12.2.3 Creating the server	415
12.2.4 The client	416
12.2.5 Setting up the system	417
12.3 Implementing an Object-Oriented System on the Web	418
12.3.1 HTML and Java servlets	418
12.3.2 Deploying the library system on the world-wide web	424
12.4 Discussion and Further Reading	446
Exercises	448
Appendix A: Java Essentials	449
A.1 Language Basics	449
A.2 A Simple Java Program	449
A.3 Primitive Data Types	452
A.4 Relational Operators	453
A.5 A Note on Input and Output	454
A.6 Selection Statements	455
A.7 Loops	457

A.8 Methods	460
A.9 Arrays	460
Bibliography	463
Index	466

Preface

At least some people reading the title of this book may wonder why there should be one more book on the topic of Object Oriented Analysis and Design (OOAD). The short answer to this question is that in our teaching of the subject for over a decade, we have not been able to find a suitable textbook on this topic at our respective universities.

We wrote up a long answer to the above question in a paper published in the 2008 SIGCSE conference. (So, if you are not satisfied with this preface, we hope you will consider reading our paper.) To summarise some of the observations and experiences in that paper, we note that our approach has always been to find ways to give a comprehensive introduction to the field of OOAD. Over the years the field has become quite vast, comprising diverse topics such as design process and principles, documentation tools (Unified Modelling Language), refactoring and, design and architectural patterns. In our experience, for most students the experience is incomplete without implementation, so, that is one more addition to the laundry list of topics to be covered in the course.

It was impossible to find a single book that gave a balanced coverage of all these topics in a manner that is understandable to the average college student. There are, of course, a number of books, some of them profound, that cover one or more of the above topics quite well. Besides their specialised nature, these books are primarily not meant to be textbooks. Expecting our students to read parts of these books and assimilate the material was not a realistic option for us.

This text is the result of our efforts over several years and provides the following:

1. A sound footing on object-oriented concepts such as classes, objects, interfaces, inheritance, polymorphism, dynamic linking, etc.
2. A good introduction to the stage of requirements analysis.
3. Use of UML to document user requirements and design.
4. An extensive treatment of the design process. The design step is, arguably, the most demanding activity (from an intellectual perspective) in the OOAD process. It is thus imperative that the student go through the design of complete systems. For pedagogical reasons we have kept the systems simple, yet sufficiently interesting to offer design choices. Going through these design exercises should help the student gain confidence to undertake reasonably complex designs.
5. Coverage of implementation issues. The reader will find critical excerpts from the implementation in Java. But he/she would be well advised to remember that this is not a book on Java. (More on this later.)
6. Appropriate use of design and architectural patterns.
7. Introduction to the art and craft of refactoring.

8. Pointers to resources that further the reader's knowledge.

It is important to remember what this book is *not* about.

1. It is not a book on Java. While the appendix has a short tutorial on the language and most of the code in the book is in Java, we do not cover constructs for the sake of teaching the language. Coverage is limited to the extent needed for understanding the implementation and for highlighting object-oriented concepts.
2. It does not cover software engineering concepts such as project management, agile technology, etc.
3. It does not treat UML extensively. Although we mention the various types of UML diagrams, many of them are not expanded because an occasion does not arise for such an undertaking.
4. It is not a catalog of design patterns or refactoring techniques. We cover only those patterns that arise naturally in our case studies. It has been our experience that design pattern discussions without a meaningful context are not well received by students.

Who will find this book useful?

Although the material in this text has primarily evolved out of a course taught for computer science senior undergraduates, others without a formal computer science background may also find this handy. In our program, students taking this are expected to have completed a course in data structures, but the material in this text does not require an intimate knowledge of the intricacies of any of these. A programmer who has used and is familiar with the APIs for some of the data structures could easily handle the material in the text. However, a certain amount of maturity with the programming process is needed, and for a typical undergraduate student this is usually obtained through a data structures course.

All the main case studies used for this book have been implemented by the authors using Java. The text is liberally peppered with snippets of code wherever we felt that a more 'concrete' feel for the design would be helpful. Most of these snippets are short and should be fairly self-explanatory and easy to read. Familiarity with a Java-like syntax and a broad understanding of the structure of Java would certainly be extremely helpful. The reader not familiar with Java but having significant software experience, need not, however, be deterred by this and can get a good feel of the entire OOAD process even without examining the code.

How to use this as computer science text

There clearly are several ways of structuring a computer science program, and the way in which this text could be used would depend on that structure.

The text is divided into three parts:

- **Part I** provides a thorough coverage of object-oriented ideas.
- **Part II** introduces the concepts of object-oriented analysis, design, implementation and, refactoring.
- **Part III** deals with more advanced design issues and approaches.

Part I, which comprises Chapters 1 through 4, gives a broad and solid foundation in concepts that are central to OOAD. The amount of time spent on covering these materials would vary considerably, depending on the program structure.

Part II begins in Chapter 5 with three useful design patterns. This part also includes Chapters 6 through 8, which introduces the first case study involving the analysis, design, and implementation of a simple library system. This is a critical choice since the entire process of design is being introduced through this case study. We chose this application because it met the following three major goals we had in selecting the case study: (i) the system should be simple so that it can be covered from analysis to implementation in a reasonable amount of time; (ii) students have an intuitive understanding of the application; (iii) several areas can be ‘naturally’ touched upon within the scope of the case study.

Several areas are touched upon in this case study and it would be pedagogically useful to emphasise these in the classroom.

- The importance of (and the quirks associated with) precisely specifying requirements and creating use case model.
- The design process. We naturally progress from the use case model to the the process of identifying classes and assigning responsibilities and coming up with sequence diagrams to implement use cases. The case study explores options in the design, which can result in lively discussions and contribute to student learning.
- The data is stored on stable storage so as to give students a sense of completeness. In this process, the student can see how the language quirks are affecting the implementation.
- The case study incorporates several design patterns in the code: Facade, Iterator, Adapter, Singleton, and Factory.
- Chapter 8 introduces refactoring and applies it to the completed design. This is done to underscore the fact that an awareness of refactoring is integral to the design process.

Covering this case study and assigning a similar project for students would be, in our opinion, essential. The amount of time spent on discussing these materials would depend on the background of the students.

Part III covers more advanced topics and spans Chapters 9 through 12. Chapter 9 introduces the use of inheritance in design, and also extends the case study. The

use of inheritance was deliberately avoided in the main case study, not only to keep the case study simple, but also to ensure that the issues associated with the use of inheritance can be dealt with in context. The extension involves some inheritance hierarchies that allow us to illustrate sound object-oriented principles including the *Liskov Substitution Principle* and the *Open–Closed Principle*. A natural extension to the library system case study leads to a discussion of the Visitor pattern.

Chapter 10 deals with the second case study, which is from the domain of electronic devices that are controlled by software. Our example concerns a microwave oven that allows the user to perform the most common functions. To keep the case study manageable we have restricted the microwave functionality, but the model is enough for our purpose. Here we introduce the concept of states, finite state machines and state transition diagrams and compare and contrast it with the use case model. In this context, we introduce the State and Observer patterns.

The third case study, in Chapter 11, is an interactive program that can be used for creating figures. The objective here is to also examine the creation of larger systems that may require decomposition into subsystems. Before presenting the case study, the student is familiarised with the Model–View–Controller architecture. During the course of the case study, the student learns the Bridge, Command, and Composite patterns.

Chapter 12 shows how to design an object-oriented system for a distributed environment. As more and more applications become available remotely, we believe it is important for students to learn how to design and implement a distributed, object-oriented system. We have focused on Java Remote Method Invocation and the implementation of web-based systems using Java Servlets. To keep the discussion within reasonable size, we have left out other technologies such as ASP.NET and some important topics such as CORBA and distributed garbage collection.

Normally, while each case study is being discussed, we expect students to work on similar projects. This may be adapted as necessary to suit each situation. Presenting the topics in this integrated manner using case studies has been very helpful in giving students a complete picture of the OOAD process. We hope that by writing this textbook we have, in some small way, contribute to the advancement of the discipline.

Acknowledgments

The following individuals at Universities Press and Springer deserve special thanks: Madhu Reddy, Manoj Karthikeyan and Beverley Ford for help with the negotiations and the contract, and Sreelatha Menon for her efficient editorial work.

Brahma Dathan would like to thank his wife, Asha, and children, Anupama and Alok, for their support during the several years it took to complete this project.

Sarnath would like to thank his family, friends and colleagues for their encouragement and support during the years he worked on the project.

The authors would like to thank Dr Bina Ramamurthy for her helpful suggestions on an early draft of the book.

As we mentioned earlier, the book was shaped by our experience in teaching the subject over a fairly long period of time. Although the courses have stabilised now, the current form does not resemble much the original version taught a decade, or even four years ago. We experimented with the topics (adding, deleting, emphasising, de-emphasising and rearranging) and changed the pedagogical approach, moving from a theory-first-practice-later approach to a more case-study-based approach. Needless to say, we did all this at the expense of our students, but they took it all in good spirit. Many of our students also provided valuable, creative criticisms on different versions of the manuscript of the book. We cannot thank our students, past and present, enough!

Brahma Dathan
Sarnath Ramnath