

## C++ DESIGN PATTERNS AND DERIVATIVES PRICING

2nd edition

Design patterns are the cutting-edge paradigm for programming in object-oriented languages. Here they are discussed in the context of implementing financial models in C++. Assuming only a basic knowledge of C++ and mathematical finance, the reader is taught how to produce well-designed, structured, reusable code via concrete examples.

This new edition includes several new chapters describing how to increase robustness in the presence of exceptions, how to design a generic factory, how to interface C++ with EXCEL, and how to improve code design using the idea of decoupling. Complete ANSI/ISO compatible C++ source code is hosted on an accompanying website for the reader to study in detail, and reuse as they see fit.

A good understanding of C++ design is a necessity for working financial mathematicians; this book provides a thorough introduction to the topic.

Cambridge University Press  
978-0-521-72162-2 - C++ Design Patterns and Derivatives Pricing  
M. S. Joshi  
Frontmatter  
[More information](#)

---

**Mathematics, Finance and Risk**

**Editorial Board**

Mark Broadie, *Graduate School of Business, Columbia University*  
Sam Howison, *Mathematical Institute, University of Oxford*  
Neil Johnson, *Centre for Computational Finance, University of Oxford*  
George Papanicolaou, *Department of Mathematics, Stanford University*

Cambridge University Press  
978-0-521-72162-2 - C++ Design Patterns and Derivatives Pricing  
M. S. Joshi  
Frontmatter  
[More information](#)

---

# C++ DESIGN PATTERNS AND DERIVATIVES PRICING

M. S. JOSHI  
*University of Melbourne*



Cambridge University Press  
978-0-521-72162-2 - C++ Design Patterns and Derivatives Pricing  
M. S. Joshi  
Frontmatter  
[More information](#)

CAMBRIDGE  
UNIVERSITY PRESS

University Printing House, Cambridge CB2 8BS, United Kingdom

Published in the United States of America by Cambridge University Press, New York

Cambridge University Press is a part of the University of Cambridge.  
It furthers the University's mission by disseminating knowledge in the pursuit of  
education, learning and research at the highest international levels of excellence.

[www.cambridge.org](http://www.cambridge.org)  
Information on this title: [www.cambridge.org/9780521721622](http://www.cambridge.org/9780521721622)

© M. S. Joshi 2008

This publication is in copyright. Subject to statutory exception  
and to the provisions of relevant collective licensing agreements,  
no reproduction of any part may take place without  
the written permission of Cambridge University Press.

First published 2008  
8th printing 2015

Printed in the United States of America by Sheridan Books, Inc.

*A catalogue record for this publication is available from the British Library*

ISBN 978-0-521-72162-2 Paperback

---

Cambridge University Press has no responsibility for the persistence or accuracy of URLs for external or  
third-party internet websites referred to in this publication, and does not guarantee that any content on  
such websites is, or will remain, accurate or appropriate.

---

Cambridge University Press  
978-0-521-72162-2 - C++ Design Patterns and Derivatives Pricing  
M. S. Joshi  
Frontmatter  
[More information](#)

To Jane

Cambridge University Press  
978-0-521-72162-2 - C++ Design Patterns and Derivatives Pricing  
M. S. Joshi  
Frontmatter  
[More information](#)

---

---

Contents

<i>Preface</i>	<i>page</i> xiii
<i>Acknowledgements</i>	xvi
1 A simple Monte Carlo model	1
1.1 Introduction	1
1.2 The theory	1
1.3 A simple implementation of a Monte Carlo call option pricer	2
1.4 Critiquing the simple Monte Carlo routine	7
1.5 Identifying the classes	9
1.6 What will the classes buy us?	10
1.7 Why object-oriented programming?	11
1.8 Key points	11
1.9 Exercises	12
2 Encapsulation	13
2.1 Implementing the pay-off class	13
2.2 Privacy	15
2.3 Using the pay-off class	16
2.4 Further extensibility defects	19
2.5 The open–closed principle	20
2.6 Key points	21
2.7 Exercises	22
3 Inheritance and virtual functions	23
3.1 ‘is a’	23
3.2 Coding inheritance	24
3.3 Virtual functions	24
3.4 Why we must pass the inherited object by reference	29
3.5 Not knowing the type and virtual destruction	30
3.6 Adding extra pay-offs without changing files	34

viii	<i>Contents</i>	
	3.7 Key points	37
	3.8 Exercises	37
4	Bridging with a virtual constructor	38
	4.1 The problem	38
	4.2 A first solution	39
	4.3 Virtual construction	43
	4.4 The rule of three	51
	4.5 The bridge	53
	4.6 Beware of new	57
	4.7 A parameters class	58
	4.8 Key points	65
	4.9 Exercises	65
5	Strategies, decoration, and statistics	66
	5.1 Differing outputs	66
	5.2 Designing a statistics gatherer	66
	5.3 Using the statistics gatherer	69
	5.4 Templates and wrappers	73
	5.5 A convergence table	77
	5.6 Decoration	80
	5.7 Key points	81
	5.8 Exercises	81
6	A random numbers class	83
	6.1 Why?	83
	6.2 Design considerations	84
	6.3 The base class	86
	6.4 A linear congruential generator and the adapter pattern	88
	6.5 Anti-thetic sampling via decoration	93
	6.6 Using the random number generator class	97
	6.7 Key points	102
	6.8 Exercises	102
7	An exotics engine and the template pattern	103
	7.1 Introduction	103
	7.2 Identifying components	104
	7.3 Communication between the components	105
	7.4 The base classes	106
	7.5 A Black–Scholes path generation engine	111
	7.6 An arithmetic Asian option	115
	7.7 Putting it all together	117
	7.8 Key points	120
	7.9 Exercises	120



	<i>Contents</i>	ix
8	Trees	121
8.1	Introduction	121
8.2	The design	123
8.3	The TreeProduct class	125
8.4	A tree class	129
8.5	Pricing on the tree	135
8.6	Key points	139
8.7	Exercises	139
9	Solvers, templates, and implied volatilities	141
9.1	The problem	141
9.2	Function objects	142
9.3	Bisecting with a template	145
9.4	Newton–Raphson and function template arguments	149
9.5	Using Newton–Raphson to do implied volatilities	151
9.6	The pros and cons of templization	154
9.7	Key points	156
9.8	Exercises	156
10	The factory	157
10.1	The problem	157
10.2	The basic idea	157
10.3	The singleton pattern	158
10.4	Coding the factory	159
10.5	Automatic registration	162
10.6	Using the factory	165
10.7	Key points	166
10.8	Exercises	167
11	Design patterns revisited	168
11.1	Introduction	168
11.2	Creational patterns	168
11.4	Behavioural patterns	170
11.5	Why design patterns?	171
11.6	Further reading	172
11.7	Key points	172
11.8	Exercise	173
12	The situation in 2007	174
12.1	Introduction	174
12.2	Compilers and the standard library	174
12.3	Boost	176

x	<i>Contents</i>	
	12.4 QuantLib	177
	12.5 xlw	177
	12.6 Key points	178
	12.7 Exercises	178
13	Exceptions	179
	13.1 Introduction	179
	13.2 Safety guarantees	180
	13.3 The use of smart pointers	180
	13.4 The rule of almost zero	183
	13.5 Commands to never use	184
	13.6 Making the wrapper class exception safe	185
	13.7 Throwing in special functions	186
	13.8 Floating point exceptions	187
	13.9 Key points	192
	13A The new wrapper class	192
14	Templatizing the factory	197
	14.1 Introduction	197
	14.2 Using inheritance to add structure	197
	14.3 The curiously recurring template pattern	199
	14.4 Using argument lists	200
	14.5 The private part of the <code>ArgumentList</code> class	207
	14.6 The implementation of the <code>ArgumentList</code>	208
	14.7 Cell matrices	220
	14.8 Cells and the <code>ArgumentLists</code>	224
	14.9 The template factory	233
	14.10 Using the templated factory	237
	14.11 Key points	242
	14.12 Exercises	243
15	Interfacing with EXCEL	244
	15.1 Introduction	244
	15.2 Usage	245
	15.3 Basic data types	247
	15.4 Extended data types	248
	15.5 xlw commands	250
	15.6 The interface file	250
	15.7 The interface generator	253
	15.8 Troubleshooting	254
	15.9 Debugging with xlls	254
	15.10 Key points	255
	15.11 Exercises	255

	<i>Contents</i>	xi
16	Decoupling	256
16.1	Introduction	256
16.2	Header files	256
16.3	Splitting files	259
16.4	Direction of information flow and levelization	260
16.5	Classes as insulators	262
16.6	inlining	262
16.7	Template code	263
16.8	Functional interfaces	264
16.9	Pimpls	264
16.10	Key points	265
16.11	Exercises	265
	Appendix A Black–Scholes formulas	266
	Appendix B Distribution functions	270
	Appendix C A simple array class	274
	C.1 Choosing an array class	274
	C.2 The header file	275
	C.3 The source code	278
	Appendix D The code	285
	D.1 Using the code	285
	D.2 Compilers	285
	D.3 License	285
	Appendix E Glossary	286
	<i>Bibliography</i>	287
	<i>Index</i>	289

Cambridge University Press  
978-0-521-72162-2 - C++ Design Patterns and Derivatives Pricing  
M. S. Joshi  
Frontmatter  
[More information](#)

---

---

# Preface

This book is aimed at a reader who has studied an introductory book on mathematical finance and an introductory book on C++ but does not know how to put the two together. My objective is to teach the reader not just how to implement models in C++ but more importantly how to think in an object-oriented way. There are already many books on object-oriented programming; however, the examples tend not to feel real to the financial mathematician so in this book we work exclusively with examples from derivatives pricing.

We do not attempt to cover all sorts of financial models but instead examine a few in depth with the objective at all times of using them to illustrate certain OO ideas. We proceed largely by example, rewriting, our designs as new concepts are introduced, instead of working out a great design at the start. Whilst this approach is not optimal from a design standpoint, it is more pedagogically accessible. An aspect of this is that our examples are designed to emphasize design principles rather than to illustrate other features of coding, such as numerical efficiency or exception safety.

We commence by introducing a simple Monte Carlo model which does not use OO techniques but rather is the simplest procedural model for pricing a call option one could write. We examine its shortcomings and discuss how classes naturally arise from the concepts involved in its construction.

In Chapter 2, we move on to the concept of encapsulation – the idea that a class allows to express a real-world analogue and its behaviours precisely. In order to illustrate encapsulation, we look at how a class can be defined for the pay-off of a vanilla option. We also see that the class we have defined has certain defects, and this naturally leads on to the open–closed principle.

In Chapter 3, we see how a better pay-off class can be defined by using inheritance and virtual functions. This raises technical issues involving destruction and passing arguments, which we address. We also see how this approach is compatible with the open–closed principle.

Using virtual functions causes problems regarding the copying of objects of unknown type, and in Chapter 4 we address these problems. We do so by introducing virtual constructors and the bridge pattern. We digress to discuss the ‘rule of three’ and the slowness of `new`. The ideas are illustrated via a vanilla options class and a parameters class.

With these new techniques at our disposal, we move on to looking at more complicated design patterns in Chapter 5. We first introduce the strategy pattern that expresses the idea that decisions on part of an algorithm can be deferred by delegating responsibilities to an auxiliary class. We then look at how templates can be used to write a wrapper class that removes a lot of our difficulties with memory handling. As an application of these techniques, we develop a convergence table using the decorator pattern.

In Chapter 6, we look at how to develop a random numbers class. We first examine why we need a class and then develop a simple implementation which provides a reusable interface and an adequate random number generator. We use the implementation to introduce and illustrate the adapter pattern, and to examine further the decorator pattern.

We move on to our first non-trivial application in Chapter 7, where we use the classes developed so far in the implementation of a Monte Carlo pricer for path-dependent exotic derivatives. As part of this design, we introduce and use the template pattern. We finish with the pricing of Asian options.

We shift from Monte Carlo to trees in Chapter 8. We see the similarities and differences between the two techniques, and implement a reusable design. As part of the design, we reuse some of the classes developed earlier for Monte Carlo.

We return to the topic of templates in Chapter 9. We illustrate their use by designing reusable solver classes. These classes are then used to define implied volatility functions. En route, we look at function objects and pointers to member functions. We finish with a discussion of the pros and cons of templization.

In Chapter 10, we look at our most advanced topic: the factory pattern. This patterns allows the addition of new functionality to a program without changing any existing files. As part of the design, we introduce the singleton pattern.

We pause in Chapter 11 to classify, summarize, and discuss the design patterns we have introduced. In particular, we see how they can be divided into creational, structural, and behavioural patterns. We also review the literature on design patterns to give the reader a guide for further study.

The final four chapters are new for the second edition. In these our focus is different: rather than focussing exclusively on design patterns, we look at some other important aspects of good coding that neophytes to C++ tend to be unaware of.

*Preface*

xv

In Chapter 12, we take a historical look at the situation in 2007 and at what has changed in recent years both in C++ and the financial engineering community's use of it.

The study of exception safety is the topic of Chapter 13. We see how making the requirement that code functions well in the presence of exceptions places a large number of constraints on style. We introduce some easy techniques to deal with these constraints.

In Chapter 14, we return to the factory pattern. The original factory pattern required us to write similar code every time we introduced a new class hierarchy; we now see how, by using argument lists and templates, a fully general factory class can be coded and reused forever.

In Chapter 15, we look at something rather different that is very important in day-to-day work for a quant: interfacing with EXCEL. In particular, we examine the `xlw` package for building xlls. This package contains all the code necessary to expose a C++ function to EXCEL, and even contains a parser to write the new code required for each function.

The concept of physical design is introduced in Chapter 16. We see how the objective of reducing compile times can affect our code organization and design.

The code for the examples in the first 11 chapters of this book can be freely downloaded from [www.markjoshi.com/design](http://www.markjoshi.com/design), and any bugfixes will be posted there. The code for the remaining chapters is taken from the `xlw` project and can be downloaded from [xlw.sourceforge.net](http://xlw.sourceforge.net). All example code is taken from release 2.1.

---

## Acknowledgements

I am grateful to the Royal Bank of Scotland for providing a stimulating environment in which to learn, study and do mathematical finance. Most of my views on coding C++ and financial modelling have been developed during my time working there. My understanding of the topic has been formed through daily discussions with current and former colleagues including Chris Hunter, Peter Jäckel, Dherminder Kainth, Sukhdeep Mahal, Robin Nicholson and Jochen Theis. I am also grateful to a host of people for their many comments on the manuscript, including Alex Barnard, Dherminder Kainth, Rob Kitching, Sukhdeep Mahal, Nadim Mahassen, Hugh McBride, Alan Stacey and Patrik Sundberg. I would also like to thank David Tranah and the rest of the team at Cambridge University Press for their careful work and attention to detail. Finally my wife has been very supportive.

I am grateful to a number of people for their comments on the second edition, with particular thanks to Chris Beveridge, Narinder Claire, Nick Denson and Lorenzo Liesch.