# 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.

**Mathematics, Finance and Risk**

# C++ DESIGN PATTERNS AND DERIVATIVES PRICING

M. S. JOSHI
*University of Melbourne*

CAMBRIDGE
UNIVERSITY PRESS

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 Universitys mission by disseminating knowledge in the pursuit of
education, learning and research at the highest international levels of excellence.

www.cambridge.org
Information on this title: www.cambridge.org/9780521721622

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

To Jane

# Contents

# 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 templatization.

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.

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`, 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`. 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.