

# Modern C++ Design

---

*Generic Programming  
and Design Patterns Applied*

**Andrei Alexandrescu**

▼▼Addison-Wesley

Boston • San Francisco • New York • Toronto • Montreal  
London • Munich • Paris • Madrid  
Capetown • Sydney • Tokyo • Singapore • Mexico City

---

---

# Contents

Foreword by Scott Meyers		xi
Foreword by John Vlissides		xv
Preface		xvii
Acknowledgments		xxi
<b>Part I</b>	<b>Techniques</b>	<b>1</b>
<b>Chapter 1</b>	<b>Policy-Based Class Design</b>	<b>3</b>
1.1	The Multiplicity of Software Design	3
1.2	The Failure of the Do-It-All Interface	4
1.3	Multiple Inheritance to the Rescue?	5
1.4	The Benefit of Templates	6
1.5	Policies and Policy Classes	7
1.6	Enriched Policies	12
1.7	Destructors of Policy Classes	12
1.8	Optional Functionality Through Incomplete Instantiation	13
1.9	Combining Policy Classes	14
1.10	Customizing Structure with Policy Classes	16
1.11	Compatible and Incompatible Policies	17
1.12	Decomposing a Class into Policies	19
1.13	Summary	20

<b>Chapter 2</b>	<b>Techniques</b>	<b>23</b>
2.1	Compile-Time Assertions	23
2.2	Partial Template Specialization	26
2.3	Local Classes	28
2.4	Mapping Integral Constants to Types	29
2.5	Type-to-Type Mapping	31
2.6	Type Selection	33
2.7	Detecting Convertibility and Inheritance at Compile Time	34
2.8	A Wrapper Around <code>type_info</code>	37
2.9	<code>NullType</code> and <code>EmptyType</code>	39
2.10	Type Traits	40
2.11	Summary	46
<b>Chapter 3</b>	<b>Typelists</b>	<b>49</b>
3.1	The Need for Typelists	49
3.2	Defining Typelists	51
3.3	Linearizing Typelist Creation	52
3.4	Calculating Length	53
3.5	Intermezzo	54
3.6	Indexed Access	55
3.7	Searching Typelists	56
3.8	Appending to Typelists	57
3.9	Erasing a Type from a Typelist	58
3.10	Erasing Duplicates	59
3.11	Replacing an Element in a Typelist	60
3.12	Partially Ordering Typelists	61
3.13	Class Generation with Typelists	64
3.14	Summary	74
3.15	Typelist Quick Facts	75
<b>Chapter 4</b>	<b>Small-Object Allocation</b>	<b>77</b>
4.1	The Default Free Store Allocator	78
4.2	The Workings of a Memory Allocator	78
4.3	A Small-Object Allocator	80
4.4	Chunks	81
4.5	The Fixed-Size Allocator	84
4.6	The <code>SmallObjAllocator</code> Class	87
4.7	A Hat Trick	89
4.8	Simple, Complicated, Yet Simple in the End	92
4.9	Administrivia	93
4.10	Summary	94
4.11	Small-Object Allocator Quick Facts	94

<b>Part II</b>	<b>Components</b>	<b>97</b>
<b>Chapter 5</b>	<b>Generalized Functors</b>	<b>99</b>
5.1	The Command Design Pattern	100
5.2	Command in the Real World	102
5.3	C++ Callable Entities	103
5.4	The Functor Class Template Skeleton	104
5.5	Implementing the Forwarding Functor: <code>operator()</code>	108
5.6	Handling Functors	110
5.7	Build One, Get One Free	112
5.8	Argument and Return Type Conversions	114
5.9	Handling Pointers to Member Functions	115
5.10	Binding	119
5.11	Chaining Requests	122
5.12	Real-World Issues I: The Cost of Forwarding Functions	122
5.13	Real-World Issues II: Heap Allocation	124
5.14	Implementing Undo and Redo with Functor	125
5.15	Summary	126
5.16	Functor Quick Facts	126
<b>Chapter 6</b>	<b>Implementing Singletons</b>	<b>129</b>
6.1	Static Data + Static Functions != Singleton	130
6.2	The Basic C++ Idioms Supporting Singleton	131
6.3	Enforcing the Singleton's Uniqueness	132
6.4	Destroying the Singleton	133
6.5	The Dead Reference Problem	135
6.6	Addressing the Dead Reference Problem (I): The Phoenix Singleton	137
6.7	Addressing the Dead Reference Problem (II): Singletons with Longevity	139
6.8	Implementing Singletons with Longevity	142
6.9	Living in a Multithreaded World	145
6.10	Putting It All Together	148
6.11	Working with SingletonHolder	153
6.12	Summary	155
6.13	SingletonHolder Class Template Quick Facts	155
<b>Chapter 7</b>	<b>Smart Pointers</b>	<b>157</b>
7.1	Smart Pointers 101	157
7.2	The Deal	158
7.3	Storage of Smart Pointers	160
7.4	Smart Pointer Member Functions	161

7.5	Ownership-Handling Strategies	163
7.6	The Address-of Operator	170
7.7	Implicit Conversion to Raw Pointer Types	171
7.8	Equality and Inequality	173
7.9	Ordering Comparisons	178
7.10	Checking and Error Reporting	181
7.11	Smart Pointers to const and const Smart Pointers	182
7.12	Arrays	183
7.13	Smart Pointers and Multithreading	184
7.14	Putting It All Together	187
7.15	Summary	194
7.16	SmartPtr Quick Facts	194
<b>Chapter 8</b>	<b>Object Factories</b>	<b>197</b>
8.1	The Need for Object Factories	198
8.2	Object Factories in C++: Classes and Objects	200
8.3	Implementing an Object Factory	201
8.4	Type Identifiers	206
8.5	Generalization	207
8.6	Minutiae	210
8.7	Clone Factories	211
8.8	Using Object Factories with Other Generic Components	215
8.9	Summary	216
8.10	Factory Class Template Quick Facts	216
8.11	CloneFactory Class Template Quick Facts	217
<b>Chapter 9</b>	<b>Abstract Factory</b>	<b>219</b>
9.1	The Architectural Role of Abstract Factory	219
9.2	A Generic Abstract Factory Interface	223
9.3	Implementing AbstractFactory	226
9.4	A Prototype-Based Abstract Factory Implementation	228
9.5	Summary	233
9.6	AbstractFactory and ConcreteFactory Quick Facts	233
<b>Chapter 10</b>	<b>Visitor</b>	<b>235</b>
10.1	Visitor Basics	235
10.2	Overloading and the Catch-All Function	242
10.3	An Implementation Refinement: The Acyclic Visitor	243
10.4	A Generic Implementation of Visitor	248
10.5	Back to the "Cyclic" Visitor	255
10.6	Hooking Variations	258
10.7	Summary	260
10.8	Visitor Generic Component Quick Facts	261

<b>Chapter 11</b>	<b>Multimethods</b>	<b>263</b>
11.1	What Are Multimethods?	264
11.2	When Are Multimethods Needed?	264
11.3	Double Switch-on-Type: Brute Force	265
11.4	The Brute-Force Approach Automated	268
11.5	Symmetry with the Brute-Force Dispatcher	273
11.6	The Logarithmic Double Dispatcher	276
11.7	FnDispatcher and Symmetry	282
11.8	Double Dispatch to Functors	282
11.9	Converting Arguments: <code>static_cast</code> or <code>dynamic_cast</code> ?	285
11.10	Constant-Time Multimethods: Raw Speed	290
11.11	BasicDispatcher and BasicFastDispatcher as Policies	293
11.12	Looking Forward	294
11.13	Summary	296
11.14	Double Dispatcher Quick Facts	297
<b>Appendix</b>	<b>A Minimalist Multithreading Library</b>	<b>301</b>
A.1	A Critique of Multithreading	302
A.2	Loki's Approach	303
A.3	Atomic Operations on Integral Types	303
A.4	Mutexes	305
A.5	Locking Semantics in Object-Oriented Programming	306
A.6	Optional <code>volatile</code> Modifier	308
A.7	Semaphores, Events, and Other Good Things	309
A.8	Summary	309
<b>Bibliography</b>		<b>311</b>
<b>Index</b>		<b>313</b>