

# Contents

<b>Acknowledgments</b>	<b>viii</b>
<b>Foreword</b>	<b>xix</b>
<b>Introduction</b>	<b>xxi</b>
<b>Part I: Introduction to Refactoring</b>	<b>1</b>
<b>Chapter 1: Refactoring: What's All the Fuss About?</b>	<b>3</b>
<b>A Quick Refactoring Overview</b>	<b>4</b>
The Refactoring Process	4
A Look at the Software Situation	5
<b>The Refactoring Process: A Closer Look</b>	<b>7</b>
Using Code Smells	7
Transforming the Code	8
The Benefits of Refactoring	9
Debunking Common Misconceptions	12
<b>Visual Basic and Refactoring</b>	<b>14</b>
Visual Basic History and Legacy Issues	14
Visual Basic Evolution	14
Dealing with Legacy Issues Through Refactoring	16
<b>Summary</b>	<b>16</b>
<b>Chapter 2: A First Taste of Refactoring</b>	<b>19</b>
<b>Calories Calculator Sample Application</b>	<b>19</b>
Calories Calculator Application	20
Growing Requirements: Calculating Ideal Weight	22
Growing Requirements: Persisting Patient Data	24
<b>Refactoring in Action</b>	<b>26</b>
Decomposing the BtnCalculate_Click Method	27
Discovering New Classes	29
Narrowing the Patient Class Interface	32
Putting Conditional Logic in the Patient Class	34
Creating the Patient Class Hierarchy	37
<b>Implementing the Persistence Functionality</b>	<b>43</b>
Saving the Data	44
Implementing Patient-History Display Functionality	52
<b>Calories Calculator, Refactored Version</b>	<b>56</b>
<b>Summary</b>	<b>58</b>

## Contents

---

<b>Chapter 3: Assembling a Refactoring Toolkit</b>	<b>59</b>
<b>Using an Automated Refactoring Tool</b>	<b>60</b>
ReSharper from JetBrains	60
Visual Assist X from Whole Tomato	61
Refactor! Pro from Developer Express	61
Getting Started with Refactor!	61
Exploring Refactor! for the VB User Interface	63
Quick Tour: Available Refactorings	67
<b>Unit-Testing Basics: The Testing Harness</b>	<b>69</b>
Why a Unit-Testing Framework?	70
Your First Taste of NUnit	72
Installing NUnit	72
Implementing Your First Test	74
The Test-Driven Approach	82
Other Test Tools to Consider	83
<b>A Few Words about Version Control</b>	<b>84</b>
<b>Summary</b>	<b>85</b>
<b>Chapter 4: Rent-a-Wheels Application Prototype</b>	<b>87</b>
<b>Interviewing the Client</b>	<b>88</b>
Interviewing the Manager	88
Interviewing the Desk Receptionist	89
Interviewing the Parking Lot Attendant	90
Interviewing Maintenance Personnel	90
<b>Taking the Initial Steps in the Rent-a-Wheels Project</b>	<b>91</b>
Actors and Use Cases	91
Vehicle States	93
First Sketch of the Main Application Window	94
Rent-a-Wheels Team Meeting	96
<b>Making the Prototype Work</b>	<b>96</b>
Examining the Database Model	96
Examining the Visual Basic Code	99
<b>Fast and Furious, a VB Approach to Programming</b>	<b>102</b>
Database-Driven Design	102
GUI-Based Application	103
Event-Driven Programming	103
Rapid Application Development (RAD)	104
Copy-Paste as a Code Reuse Mechanism	104
<b>From Prototype to Delivery Through the Refactoring Process</b>	<b>105</b>
<b>Summary</b>	<b>105</b>

<b>Part II: Preliminary VB Refactorings</b>	<b>107</b>
<b>Chapter 5: Chameleon Language: From Weak Static Typing to Strong Dynamic Typing</b>	<b>109</b>
<b>Option Explicit and Option Strict, the .NET Effect</b>	<b>110</b>
<b>Setting Option Explicit On in Relaxed Code</b>	<b>111</b>
Understanding the Set Option Explicit On Refactoring	112
Refactoring the Rent-a-Wheels Code to Explicit Form	114
<b>Setting Option Strict On in Relaxed Code</b>	<b>115</b>
A Slightly Artificial Example of Permissive VB code	116
Convoluted Use of Variables Resolved by the Definition of New Variables	119
Inferring Variable Type	122
Putting It All Together with Type-Conversion Functions	125
Dealing with Methods, Fields, Properties, and Other Members	127
Applying Set Option Strict On Refactoring to the Rent-a-Wheels Application	132
<b>Static Versus Dynamic Typing and Visual Basic</b>	<b>135</b>
Late Binding in Visual Basic 6 and Prior	136
Duck Typing	136
Resetting Dynamic or Static Behavior at the File Level	138
Providing a Statically Typed Wrapper for Dynamic Code	138
<b>Activating Explicit and Strict Compiler Options</b>	<b>141</b>
Setting Options in the Project Properties	141
Changing the Default Behavior of the Visual Basic Compiler	142
Setting Options in Source Files	143
Using Item Templates to Set Options	143
<b>Summary</b>	<b>145</b>
<b>Chapter 6: Error Handling: From Legacy to Structured in a Few Easy Steps</b>	<b>147</b>
<b>Legacy Error Handling Versus Structured Error Handling</b>	<b>148</b>
Legacy (Unstructured) Error Handling	148
Structured Error Handling	150
<b>The Benefits of Structured Error Handling</b>	<b>153</b>
Structured Versus Unstructured Code	153
Exceptions as Types, Not Numbers	154
Error Filtering	154
The Finally Block	155
.NET Interoperability	155
<b>Replacing the On Error Construct with Try-Catch-Finally</b>	<b>156</b>
Understanding the When Keyword	158
Refactoring Steps for Replacing On Error with Try-Catch-Finally	158
Replacing the On Error Goto Label with the Try-Catch-Finally Construct	159
Replacing On Error Resume Next with the Try-Catch-Finally Construct	162

## Contents

---

<b>Replacing Error Code with Exception Type</b>	<b>164</b>
Replacing System Error Codes with Exception Types	166
Replacing Custom Error Codes with Exception Types	168
<b>Error Handling in the Rent-a-Wheels Application</b>	<b>169</b>
Application-Level Events in VB 2009	169
<b>Summary</b>	<b>171</b>
<b>Chapter 7: Basic Hygiene: Eliminating Dead Code, Reducing Scope, Using Explicit Imports, and Removing Unused References</b>	<b>173</b>
<hr/>	
<b>Eliminating Dead Code</b>	<b>174</b>
Types of Dead Code	175
Common Sources of Dead Code	176
<b>Reducing the Scope and Access Level of Unduly Exposed Elements</b>	<b>179</b>
Scope and Access Level	181
Common Sources of Overexposure	182
Dealing with Overexposure	186
<b>Using Explicit Imports</b>	<b>187</b>
Imports Section Depicts Dependencies in Your System	188
<b>Removing Unused Assembly References</b>	<b>191</b>
<b>Basic Hygiene in the Rent-a-Wheels Application</b>	<b>192</b>
<b>Summary</b>	<b>193</b>
<b>Part III: Getting Started with Standard Refactoring Transformations</b>	<b>195</b>
<b>Chapter 8: From Problem Domain to Code: Closing the Gap</b>	<b>197</b>
<hr/>	
<b>Understanding the Problem Domain</b>	<b>198</b>
Step One: Gathering the Information	198
Step Two: Agreeing on the Vocabulary	199
Step Three: Describing the Interactions	200
Step Four: Building the Prototype	201
<b>Naming Guidelines</b>	<b>201</b>
Capitalization Styles	202
Simple Naming Guidelines	203
Good Communication: Choosing the Right Words	204
Rename Refactoring	206
<b>Published and Public Interfaces</b>	<b>208</b>
Self-Contained Applications Versus Reusable Modules	209
Modifying the Public Interfaces	212
Safe Rename Refactoring in Refactor!	214
<b>Rename and Safe Rename Refactoring in the Rent-a-Wheels Application</b>	<b>217</b>
<b>Summary</b>	<b>218</b>

<b>Chapter 9: The Method Extraction Remedy for Duplicated Code</b>	<b>219</b>
<b>Why Keep Code Encapsulated and Details Hidden?</b>	<b>219</b>
<b>Information and Implementation Hiding</b>	<b>220</b>
<b>Decomposing Methods</b>	<b>223</b>
Circumference Calculation — Long Method Example	223
Extracting Circumference Length Calculation Code	226
Extracting the Radius Calculation Code	229
Extracting the “Wait for User to Close” Code	229
Extracting the Read Coordinates Code	229
Extract Method Refactoring in Refactor!	233
<b>The Duplicated Code Smell</b>	<b>234</b>
Sources of Duplicated Code	235
Copy-Paste Programming	236
<b>Magic Literals</b>	<b>237</b>
Introduce Constant Refactoring in Refactor!	239
<b>Extract Method and Replace Magic Literal Refactoring in the Rent-a-Wheels Application</b>	<b>240</b>
<b>Summary</b>	<b>240</b>
<b>Chapter 10: Method Consolidation and Extraction Techniques</b>	<b>243</b>
<b>Dealing with Temporary Variables</b>	<b>243</b>
Move Declaration Near Reference Refactoring	244
Move Initialization to Declaration Refactoring	248
Split Temporary Variable Refactoring	249
Inline Temp Refactoring	253
Replace Temp with Query Refactoring	256
<b>Method Reorganization Heuristics</b>	<b>258</b>
<b>Method Reorganization and Rent-a-Wheels</b>	<b>259</b>
Removing Duplication in Rent-a-Wheels	261
Magic Literals, Comments, and Event-Handling Blindness in Rent-a-Wheels	263
<b>Summary</b>	<b>267</b>
<b>Part IV: Advanced Refactorings</b>	<b>269</b>
<b>Chapter 11: Discovering Objects</b>	<b>271</b>
<b>A Quick Object-Oriented Programming Overview</b>	<b>272</b>
What Are Objects Anyway?	272
Encapsulation and Objects	272
Encapsulate Field Refactoring in Refactor!	274
Object State Retention	276

## Contents

---

Classes	276
Object Identity	277
Objects as Basic Building Blocks	278
Root Object	278
Object Lifetime and Garbage Collection	279
Messages	280
<b>Designing Classes</b>	<b>281</b>
Classes Are Nouns, Operations Are Verbs	284
Classes, Responsibilities, and Collaborators	288
Entities and Relationships	297
<b>Discovering Hidden Classes</b>	<b>298</b>
Dealing with Database-Driven Design	299
Moving From Procedural to Object-Oriented Design	302
Keeping Domain, Presentation, and Persistence Apart	308
Discovering Objects and the Rent-a-Wheels Application	313
<b>Summary</b>	<b>320</b>
<b>Chapter 12: Advanced Object-Oriented Concepts and Related Refactorings</b>	<b>323</b>
<hr/>	
<b>Inheritance, Polymorphism, and Genericity</b>	<b>324</b>
Inheritance	324
Polymorphism	329
Genericity	332
<b>Inheritance Abuse and Refactoring Solutions</b>	<b>334</b>
Composition Mistaken for Inheritance and Other Misuses	337
Refactoring for Inheritance — Print-System Illustration	342
<b>Making Use of Generics</b>	<b>360</b>
<b>Inheritance and Generic Types in the Rent-a-Wheels Application</b>	<b>364</b>
Extracting Super	364
Employing Generics	364
Extract Data Objects Provider Class	365
<b>Summary</b>	<b>369</b>
<b>Chapter 13: Code Organization on a Large Scale</b>	<b>371</b>
<hr/>	
<b>Namespaces</b>	<b>371</b>
Naming Guidelines and Namespace Organization	372
Nested Namespaces	372
Changing the Root Namespace Name	372
Using Import Statements	373

---

## Contents

<b>Assemblies</b>	<b>375</b>
Binary Reuse	375
Namespace Organization Guidelines	377
Dependency Considerations	381
<b>Visual Basic Project File Structure Organization</b>	<b>387</b>
Move Type to File Refactoring in Refactor!	388
Partial Classes	390
Inherited Form	390
Abstract Form Inheritance	392
Delegating Abstract Form Work to a Form Helper Class	392
<b>Namespace Organization and Windows Forms Inheritance in Rent-a-Wheels</b>	<b>394</b>
Extracting Parent Administration Form through Abstract Form Helper Pattern Application	394
Namespace and Assembly Reorganization	402
<b>Summary</b>	<b>403</b>
<b>Part V: Refactoring Applied</b>	<b>405</b>
<b>Chapter 14: Refactoring to Patterns</b>	<b>407</b>
<b>Design Patterns: What's All the Fuss About?</b>	<b>408</b>
Defining Design Patterns	408
Classifying Patterns	409
Pattern Elements	409
Weighing the Benefits of Design Patterns	410
Using Patterns	410
<b>Example Design Pattern: Abstract Factory</b>	<b>411</b>
Name	411
Problem	411
Solution	420
Consequences	424
<b>Dependency Injection Pattern</b>	<b>426</b>
Problem	426
Solution	428
Constructor-Based vs. Property-Based Injection	429
What Service Implementation to Inject	430
Consequences	431
Refactoring to DI	434
<b>Refactoring to Patterns and Rent-a-Wheels Application</b>	<b>434</b>
Eliminating Code That Duplicates Functionality Available in .NET Framework	434
Injecting Data Classes to GUI Classes via Dependency Injection	435
CRUD Persistence Pattern	437
<b>Summary</b>	<b>438</b>

## Contents

---

<b>Chapter 15: LINQ and Other VB 2008 Enhancements</b>	<b>439</b>
<b>Type Inference for Local Variables</b>	<b>439</b>
<b>XML Productivity Enhancements</b>	<b>440</b>
XML Literals	440
Navigating XML with XML Axis Properties	444
Extract XML Literal to Resource in Refactor!	444
<b>Querying the Objects with LINQ</b>	<b>445</b>
Old Example in New Robes	448
Object-Relational Mapping with LINQ to SQL	451
LINQ and the Rent-a-Wheels Application	454
<b>Summary</b>	<b>464</b>
<b>Chapter 16: The Future of Legacy VB Code</b>	<b>465</b>
<b>To Migrate or Not To Migrate</b>	<b>466</b>
Migration Cannot Be 100 Percent Automated	467
VB 6 and VB .NET Code Can Interoperate	467
Migration Tools and Libraries	469
<b>Preliminary VB 6 Refactorings</b>	<b>470</b>
Breaking the Monolith	470
Dealing with Conditional Compilation	472
<b>Putting Your Migrated Code under a Testing Harness</b>	<b>472</b>
Introducing a Functional Testing Harness	472
Implementing a Functional Testing Harness	473
<b>Upgrading Your Legacy Code</b>	<b>477</b>
Strict Static Typing	477
Moving Design from Procedural toward an Object-Oriented Paradigm	477
Introducing Inheritance	478
Making Use of Parameterized Constructor	479
Using Generic Containers for Additional Type Safety	479
Upgrading Exception Handling	481
Implementing XML Comments	481
Releasing Resources in .NET	482
<b>Summary</b>	<b>482</b>
<b>Appendix A: Unleash Refactor!</b>	<b>483</b>
<b>Appendix B: Rent-a-Wheels Prototype Internals and Intricacies</b>	<b>487</b>
<b>Hand Over Button Click Event-Handling Code</b>	<b>487</b>
<b>Receive Button Click Event-Handling Code</b>	<b>488</b>

---

## Contents

<b>Charge Button Click Event-Handling Code</b>	<b>488</b>
<b>Change Branch Button Click Event-Handling Code</b>	<b>489</b>
<b>To Maintenance and From Maintenance Button Click Event Code</b>	<b>493</b>
<b>Administer Fleet Form</b>	<b>494</b>
Delete Button Click Event-Handling Routine	495
New Button Click Event-Handling Routine	495
Reload Button Click Event-Handling Routine	495
Form Load Event-Handling Routine	495
Administer Fleet Form Class Code: Fields	498
Left Button Click Event-Handling Routine	499
Save Button Click Event-Handling Routine	500
<b>Display Button Click Event-Handling Routine</b>	<b>502</b>
<b>Summary</b>	<b>506</b>
<b>Index</b>	<b>507</b>