PHP IN ACTION

Objects, Design, Agility

Dagfinn Reiersøl
Marcus Baker
Chris Shiflett

MANNING

contents

preface xvii
acknowledgments xix
about this book xxi
about the title xxv
about the cover illustration xxvi

Part 1 Tools and concepts 1

1 PHP and modern software development 3

1.1 How PHP can help you 4
Why PHP is so popular 4 ✦ Overcoming PHP's limitations 8

1.2 Languages, principles, and patterns 10
Agile methodologies: from hacking to happiness 10 ✦ PHP 5 and software trends 12 ✦ The evolving discipline of object-oriented programming 12 ✦ Design patterns 13 ✦ Refactoring 14
Unit testing and test-driven development 15

1.3 Summary 17

2 Objects in PHP 18

2.1 Object fundamentals 19
Why we're comparing PHP to Java 19 ✦ Objects and classes 20
Hello world 20 ✦ Constructors: creating and initializing objects 21 ✦ Inheritance and the extends keyword 23
Inheriting constructors 24

2.2 Exception handling 25
How exceptions work 25 ✦ Exceptions versus return codes—when to use which 27 ✦ Creating your own exception classes 29
Replacing built-in PHP fatal errors with exceptions 30
Don't overdo exceptions 30

2.3 Object references in PHP 4 and PHP 5 31
How object references work 32 ✦ The advantages of object references 33 ✦ When references are not so useful 33
2.4 Intercepting method calls and class instantiation 34
What is "method overloading"? 34 ✦ Java-style method overloading in PHP 35 ✦ A near aspect-oriented experience: logging method calls 36 ✦ Autoloading classes 38
2.5 Summary 39

3 Using PHP classes effectively 40

3.1 Visibility: private and protected methods and variables 41
How visible do we want our methods to be? 42 ✦ When to use private methods 43 ✦ When to use protected methods 44
Keeping your instance variables private or protected 44
Accessors for private and protected variables 45 ✦ The best of both worlds? Using interception to control variables 46
Final classes and methods 48
3.2 The class without objects: class methods, variables, and constants 49
Class (static) methods 50 ✦ When to use class methods 51
Class variables 52 ✦ Class constants 53
The limitations of constants in PHP 54
3.3 Abstract classes and methods (functions) 56
What are abstract classes and methods? 56
Using abstract classes 56
3.4 Class type hints 57
How type hints work 58 ✦ When to use type hints 58
3.5 Interfaces 60
What is an interface? 60 ✦ Do we need interfaces in PHP? 61
Using interfaces to make design clearer 61 ✦ Using interfaces to improve class type hints 62 ✦ Interfaces in PHP 5 versus Java 64
3.6 Summary 64

4 Understanding objects and classes 65

4.1 Why objects and classes are a good idea 66
Classes help you organize 67 ✦ You can tell objects to do things 67 ✦ Polymorphism 67 ✦ Objects make code easier to read 68 ✦ Classes help eliminate duplication 73 ✦ You can reuse objects and classes 74 ✦ Change things without affecting everything 75 ✦ Objects provide type safety 75
4.2 Criteria for good design 76
Don't confuse the end with the means 78 ✦ Transparency 78
Simple design 79 ✦ Once and only once 80

4.3 What are objects, anyway? 82
Objects come from the unreal world 82 ✦ Domain object basics 84
4.4 Summary 85

5 Understanding class relationships 87

5.1 Inheritance 88
Inheritance as a thinking tool 88 ✦ Refactoring to inheritance 89
5.2 Object composition 94
5.3 Interfaces 96
The interface as a thinking tool 97 ✦ Single and multiple inheritance 98
5.4 Favoring composition over inheritance 99
Avoiding vaguely named parent classes 99
Avoiding deep inheritance hierarchies 100
5.5 Summary 101

6 Object-oriented principles 102

6.1 Principles and patterns 103
Architectural principles or patterns 104 ✦ Learning OO principles 104
6.2 The open-closed principle (OCP) 105
OCP for beginners 105 ✦ Replacing cases with classes 106
How relevant is the OCP in PHP? 108
6.3 The single-responsibility principle (SRP) 109
Mixed responsibilities: the template engine 110 ✦ An experiment: separating the responsibilities 112 ✦ Was the experiment successful? 114
6.4 The dependency-inversion principle (DIP) 115
What is a dependency? 116 ✦ Inserting an interface 118
6.5 Layered designs 119
The "three-tier" model and its siblings 119
Can a web application have a Domain layer? 120
6.6 Summary 122

7 Design patterns 123

7.1 Strategy 125
"Hello world" using Strategy 125 ✦ How Strategy is useful 127
7.2 Adapter 128
Adapter for beginners 128 ✦ Making one template engine look like another 129 ✦ Adapters with multiple classes 131
Adapting to a generic interface 134
7.3 Decorator 135
Resource Decorator 135 ✦ Decorating and redecorating 136

7.4 Null Object 139
Mixing dark and bright lights 140 ✦ Null Strategy objects 140
7.5 Iterator 142
How iterators work 142 ✦ Good reasons to use iterators 143
Iterators versus plain arrays 143 ✦ SPL iterators 144
How SPL helps us solve the iterator/array conflict 145
7.6 Composite 145
Implementing a menu as a Composite 146 ✦ The basics 148
A fluent interface 149 ✦ Recursive processing 149
Is this inefficient? 150
7.7 Summary 151

8 Design how-to: date and time handling 152

8.1 Why object-oriented date and time handling? 153
Easier, but not simpler 153 ✦ OO advantages 154
8.2 Finding the right abstractions 155
Single time representation: Time Point, Instant, DateAndTime 155 ✦ Different kinds of time spans: Period, Duration, Date Range, Interval 156
8.3 Advanced object construction 158
Using creation methods 158 ✦ Multiple constructors 159
Using factory classes 162
8.4 Large-scale structure 163
The package concept 164 ✦ Namespaces and packages 165
PHP's lack of namespace support 166
Dealing with name conflicts 167
8.5 Using value objects 173
How object references can make trouble 173 ✦ Implementing value objects 174 ✦ Changing an immutable object 175
8.6 Implementing the basic classes 176
DateAndTime 176 ✦ Properties and fields 177
Periods 183 ✦ Intervals 185
8.7 Summary 186

Part 2 Testing and refactoring 187

9 Test-driven development 189

9.1 Building quality into the process 190
Requirements for the example 191 ✦ Reporting test results 192

9.2 Database select 192
A rudimentary test 193 ✦ The first real test 194 ✦ Make it pass 196 ✦ Make it work 198 ✦ Test until you are confident 200
9.3 Database insert and update 201
Making the tests more readable 201 ✦ Red, green, refactor 203
9.4 Real database transactions 205
Testing transactions 205 ✦ Implementing transactions 207
The end of debugging? 208 ✦ Testing is a tool, not a substitute 209
9.5 Summary 209

10 Advanced testing techniques 210

10.1 A contact manager with persistence 211
Running multiple test cases 212 ✦ Testing the contact's persistence 213 ✦ The Contact and ContactFinder classes 215
setUp() and tearDown() 217 ✦ The final version 218
10.2 Sending an email to a contact 219
Designing the Mailer class and its test environment 219 ✦ Manually coding a mock object 220 ✦ A more sophisticated mock object 221 ✦ Top-down testing 222 ✦ Mock limitations 224
10.3 A fake mail server 225
Installing fakemail 225 ✦ A mail test 227
Gateways as adapters 230
10.4 Summary 230

11 Refactoring web applications 232

11.1 Refactoring in the real world 233
Early and late refactoring 234
Refactoring versus reimplementation 235
11.2 Refactoring basics: readability and duplication 236
Improving readability 236 ✦ Eliminating duplication 238
11.3 Separating markup from program code 241
Why the separation is useful 242 ✦ Using CSS appropriately 242 ✦ Cleaning up a function that generates a link 243 ✦ Introducing templates in SimpleTest 248
11.4 Simplifying conditional expressions 253
A simple example 254 ✦ A longer example: authentication code 255 ✦ Handling conditional HTML 261
11.5 Refactoring from procedural to object-oriented 262
Getting procedural code under test 263
Doing the refactorings 264
11.6 Summary 267

12 Taking control with web tests 269

12.1 Revisiting the contact manager 270
The mock-up 271 ✦ Setting up web testing 272
Satisfying the test with fake web page interaction 274
Write once, test everywhere 275

12.2 Getting a working form 277
Trying to save the contact to the database 278 ✦ Setting up the database 279 ✦ Stubbing out the finder 281

12.3 Quality assurance 283
Making the contact manager unit-testable 283
From use case to acceptance test 285

12.4 The horror of legacy code 288

12.5 Summary 292

Part 3 Building the web interface 293

13 Using templates to manage web presentation 295

13.1 Separating presentation and domain logic 296
To separate or not to separate… 296 ✦ Why templates? 297

13.2 Which template engine? 299
Plain PHP 301 ✦ Custom syntax: Smarty 302
Attribute language: PHPTAL 304

13.3 Transformation: XSLT 308
"XMLizing" a web page 309 ✦ Setting up XSLT 309
The XSLT stylesheet 310 ✦ Running XSLT from PHP 312

13.4 Keeping logic out of templates 313
View Helper 314 ✦ Alternating row colors 315 ✦ Handling date and time formats 315 ✦ Generating hierarchical displays 318 ✦ Preventing updates from the template 321

13.5 Templates and security 322
PHPTAL 322 ✦ Smarty 323 ✦ XSLT 323

13.6 Summary 323

14 Constructing complex web pages 325

14.1 Combining templates (Composite View) 325
Composite View: one or several design patterns? 326
Composite data and composite templates 326

14.2 Implementing a straightforward composite view 326
What we need to achieve 327 ✦ Using Smarty 328
Using PHPTAL 330 ✦ Using page macros with PHPTAL 331
14.3 Composite View examples 332
Making print-friendly versions of pages 333
Integrating existing applications into a Composite View 335
Multi-appearance sites and Fowler's Two Step View 336
14.4 Summary 337

15 User interaction 338

15.1 The Model-View-Controller architecture 340
Clearing the MVC fog 341 ✦ Defining the basic concepts 342
Command or action? 344 ✦ Web MVC is not rich-client MVC 345
15.2 The Web Command pattern 346
How it works 347 ✦ Command identifier 347
Web handler 348 ✦ Command executor 349
15.3 Keeping the implementation simple 349
Example: a "naive" web application 349
Introducing command functions 351
15.4 Summary 355

16 Controllers 356

16.1 Controllers and request objects 357
A basic request object 357 ✦ Security issues 358
16.2 Using Page Controllers 361
A simple example 361 ✦ Choosing Views from a Page Controller 363 ✦ Making commands unit-testable 364
Avoiding HTML output 365 ✦ Using templates 365
The redirect problem 366
16.3 Building a Front Controller 369
Web Handler with single-command classes 370 ✦ What more does the command need? 371 ✦ Using command groups 371
Forms with multiple submit buttons 373 ✦ Generating commands with JavaScript 374 ✦ Controllers for Composite Views 374
16.4 Summary 376

17 Input validation 377

17.1 Input validation in application design 378
Validation and application architecture 378 ✦ Strategies for validation 379 ✦ Naming the components of a form 380

17.2 Server-side validation and its problems 381
The duplication problem 381 ✦ The styling problem 382
Testing and page navigation problems 383
How many problems can we solve? 383

17.3 Client-side validation 384
Ordinary, boring client-side validation 384 ✦ Validating field-by-field 386 ✦ You can't do that! 388 ✦ The form 391

17.4 Object-oriented server-side validation 393
Rules and validators 393 ✦ A secure request object architecture 394 ✦ Now validation is simple 399 ✦ A class to make it simple 400 ✦ Using Specification objects 403 ✦ Knowledge-rich design 407 ✦ Adding validations to the facade 407

17.5 Synchronizing server-side and client-side validation 409
Form generator 410 ✦ Configuration file 410
Generating server-side validation from client-side validation 410

17.6 Summary 412

18 Form handling 413

18.1 Designing a solution using HTML_QuickForm 414
Minimalistic requirements and design 414 ✦ Putting generated elements into the HTML form 415 ✦ Finding abstractions 416
More specific requirements 417 ✦ The select problem 418

18.2 Implementing the solution 419
Wrapping the HTML_QuickForm elements 420 ✦ Input controls 421 ✦ Which class creates the form controls? 425
Validation 426 ✦ Using the form object in a template 427
What next? 430

18.3 Summary 431

19 Database connection, abstraction, and configuration 432

19.1 Database abstraction 433
Prepared statements 434
Object-oriented database querying 437

19.2 Decorating and adapting database resource objects 438
A simple configured database connection 438
Making an SPL-compatible iterator from a result set 440

19.3 Making the database connection available 442
Singleton and similar patterns 443
Service Locator and Registry 445

19.4 Summary 448

Part 4 Databases and infrastructure 449

20 Objects and SQL 451

20.1 The object-relational impedance mismatch 452

20.2 Encapsulating and hiding SQL 453
A basic example 454 ✦ Substituting strings in SQL statements 455

20.3 Generalizing SQL 459
Column lists and table names 460 ✦ Using SQL aliases 463
Generating INSERT, UPDATE and DELETE statements 463
Query objects 468 ✦ Applicable design patterns 468

20.4 Summary 469

21 Data class design 470

21.1 The simplest approaches 471
Retrieving data with Finder classes 471
Mostly procedural: Table Data Gateway 474

21.2 Letting objects persist themselves 479
Finders for self-persistent objects 480
Letting objects store themselves 485

21.3 The Data Mapper pattern 486
Data Mappers and DAOs 487 ✦ These patterns are all the same 488 ✦ Pattern summary 490

21.4 Facing the real world 490
How the patterns work in a typical web application 490
Optimizing queries 492

21.5 Summary 492

appendix A Tools and tips for testing 493

appendix B Security 503

resources 511

index 513