Design patterns using Spring and Guice

Dependency Injection

Dhanji R. Prasanna

MANNING

contents

preface xv
acknowledgments xvii
about this book xix
about the cover illustration xxii

1 *Dependency injection: what's all the hype?* 1

1.1 Every solution needs a problem 2
Seeing objects as services 2

1.2 Pre-DI solutions 4
Construction by hand 5 ▪ The Factory pattern 7 ▪ The Service Locator pattern 12

1.3 Embracing dependency injection 13
The Hollywood Principle 13 ▪ Inversion of Control vs. dependency injection 15

1.4 Dependency injection in the real world 17
Java 17 ▪ DI in other languages and libraries 19

1.5 Summary 19

2 *Time for injection* 21

2.1 Bootstrapping the injector 22

2.2 Constructing objects with dependency injection 23

2.3 Metadata and injector configuration 26

XML injection in Spring 27 ▪ From XML to in-code configuration 30
Injection in PicoContainer 31 ▪ Revisiting Spring and autowiring 34

2.4 Identifying dependencies for injection 36

Identifying by string keys 37 ▪ Limitations of string keys 42
Identifying by type 44 ▪ Limitations of identifying by type 46
Combinatorial keys: a comprehensive solution 47

2.5 Separating infrastructure and application logic 51

2.6 Summary 52

3 Investigating DI 54

3.1 Injection idioms 55

Constructor injection 55 ▪ Setter injection 56 ▪ Interface injection 60
Method decoration (or AOP injection) 62

3.2 Choosing an injection idiom 65

Constructor vs. setter injection 66 ▪ The constructor pyramid problem 69
The circular reference problem 71 ▪ The in-construction problem 75
Constructor injection and object validity 78

3.3 Not all at once: partial injection 81

The reinjection problem 81 ▪ Reinjection with the Provider pattern 82
The contextual injection problem 84 ▪ Contextual injection with the Assisted Injection pattern 86 ▪ Flexible partial injection with the Builder pattern 88

3.4 Injecting objects in sealed code 92

Injecting with externalized metadata 93 ▪ Using the Adapter pattern 95

3.5 Summary 96

4 Building modular applications 99

4.1 Understanding the role of an object 100

4.2 Separation of concerns (my pants are too tight!) 101

Perils of tight coupling 102 ▪ Refactoring impacts of tight coupling 105 ▪ Programming to contract 108 ▪ Loose coupling with dependency injection 111

4.3 Testing components 112

Out-of-container (unit) testing 113 ▪ I really need my dependencies! 114
More on mocking dependencies 115 ▪ Integration testing 116

4.4 Different deployment profiles 118

Rebinding dependencies 118 ▪ Mutability with the Adapter pattern 119

4.5 Summary 121

5 *Scope: a fresh breath of state 123*

5.1 What is scope? 124

5.2 The no scope (or default scope) 125

5.3 The singleton scope 128

Singletons in practice 131 ▪ The singleton anti-pattern 135

5.4 Domain-specific scopes: the web 139

HTTP request scope 141 ▪ HTTP session scope 149

5.5 Summary 154

6 *More use cases in scoping 156*

6.1 Defining a custom scope 157

A quick primer on transactions 157 ▪ Creating a custom transaction scope 158 ▪ A custom scope in Guice 160 ▪ A custom scope in Spring 164

6.2 Pitfalls and corner cases in scoping 166

Singletons must be thread-safe 167 ▪ Perils of scope-widening injection 169

6.3 Leveraging the power of scopes 180

Cache scope 181 ▪ Grid scope 181 ▪ Transparent grid computing with DI 183

6.4 Summary 184

7 *From birth to death: object lifecycle 186*

7.1 Significant events in the life of objects 187

Object creation 187 ▪ Object destruction (or finalization) 189

7.2 One size doesn't fit all (domain-specific lifecycle) 191

Contrasting lifecycle scenarios: servlets vs. database connections 191 ▪ The Destructor anti-pattern 196 ▪ Using Java's Closeable interface 197

7.3 A real-world lifecycle scenario: stateful EJBs 198

7.4 Lifecycle and lazy instantiation 201

7.5 Customizing lifecycle with postprocessing 202

7.6 Customizing lifecycle with multicasting 205

7.7 Summary 207

8 *Managing an object's behavior 210*

8.1 Intercepting methods and AOP 211

A tracing interceptor with Guice 212 ▪ A tracing interceptor with Spring 214 How proxying works 216 ▪ Too much advice can be dangerous! 219

8.2 Enterprise use cases for interception 221

Transactional methods with warp-persist 222 ▪ Securing methods with Spring Security 224

8.3 Pitfalls and assumptions about interception and proxying 228

Sameness tests are unreliable 228 ▪ Static methods cannot be intercepted 230 ▪ Neither can private methods 231 ▪ And certainly not final methods! 233 ▪ Fields are off limits 234 ▪ Unit tests and interception 236

8.4 Summary 238

9 *Best practices in code design 240*

9.1 Objects and visibility 241

Safe publication 244 ▪ Safe wiring 245

9.2 Objects and design 247

On data and services 247 ▪ On better encapsulation 252

9.3 Objects and concurrency 257

More on mutability 258 ▪ Synchronization vs. concurrency 261

9.4 Summary 264

10 *Integrating with third-party frameworks 266*

10.1 Fragmentation of DI solutions 267

10.2 Lessons for framework designers 270

Rigid configuration anti-patterns 271 ▪ Black box anti-patterns 276

10.3 Programmatic configuration to the rescue 280

Case study: JSR-303 280

10.4 Summary 286

11 *Dependency injection in action! 289*

11.1 Crosstalk: a Twitter clone! 290

Crosstalk's requirements 290

11.2 Setting up the application 290

11.3 Configuring Google Sitebricks 294

11.4 Crosstalk's modularity and service coupling 295

11.5 The presentation layer 296

The HomePage template 298 ▪ The Tweet domain object 301
Users and sessions 302 ▪ Logging in and out 304

11.6 The persistence layer 308
Configuring the persistence layer 310
11.7 The security layer 311
11.8 Tying up to the web lifecycle 312
11.9 Finally: up and running! 313
11.10 Summary 314

appendix A The Butterfly Container 315
appendix B SmartyPants for Adobe Flex 320
index 323