PART 1  OVERVIEW OF POJOs AND LIGHTWEIGHT FRAMEWORKS .............1

Developing with POJOs: faster and easier  3

1.1 The disillusionment with EJBs  5
  A brief history of EJBs  5  •  A typical EJB 2 application architecture  6
  The problems with EJBs  7  •  EJB 3 is a step in the right direction  11

1.2 Developing with POJOs  12
  Using an object-oriented design  14  •  Using POJOs  15  •  Persisting
  POJOs  16  •  Eliminating DTOs  18  •  Making POJOs
  transactional  19  •  Configuring applications with Spring  25
  Deploying a POJO application  27  •  POJO design summary  28

1.3 Summary  30
3.3 Implementing a domain model: an example 80
   Implementing a domain service method 80
   Implementing a domain entity method 87
   Summary of the design 92

3.4 Summary 93

4 Overview of persisting a domain model 95

4.1 Mapping an object model to a database 96
   Mapping classes 97
   Mapping object relationships 99
   Mapping inheritance 103
   Managing object lifecycles 107
   Persistent object identity 107

4.2 Overview of ORM frameworks 108
   Why you don't want to persist objects yourself 109
   The key features of an ORM framework 109
   Benefits and drawbacks of using an ORM framework 114

4.3 Overview of JDO and Hibernate 117
   Declarative mapping between the object model and the schema 117
   API for creating, reading, updating, and deleting objects 118
   Query language 119
   Support for transactions 120
   Lazy and eager loading 121
   Object caching 121
   Detached objects 124
   Hibernate vs. JDO 124

4.4 Designing repositories with Spring 125
   Implementing JDO and Hibernate repositories 125
   Using the Spring ORM classes 126
   Making repositories easier to test 129

4.5 Testing a persistent domain model 132
   Object/relational testing strategies 133
   Testing against the database 135
   Testing without the database 138
   Overview of ORMUnit 140

4.6 Performance tuning JDO and Hibernate 141
   Without any tuning 141
   Configuring eager loading 142
   Using a process-level cache 145
   Using the query cache 145

4.7 The example schema 146

4.8 Summary 148

5 Persisting a domain model with JDO 2.0 149

5.1 JDO issues and limitations 150
   Configuring JDO object identity 151
   Persisting interfaces 155
   Using the JDO enhancer 158
5.2 Persisting a domain model class with JDO 159
   Writing JDO persistence tests with ORMUnit 159 • Testing persistent JDO objects 164 • Making a class persistent 170
5.3 Implementing the JDO repositories 173
   Writing a mock object test for findRestaurants() 174 • Implementing JDORestaurantRepositoryImpl 178 • Writing the query that finds the restaurants 180 • Writing tests for a query 180
5.4 JDO performance tuning 183
   Using fetch groups to optimize object loading 184 • Using a PersistenceManagerFactory-level cache 191 • Using a query cache 193
5.5 Summary 193

6 Persisting a domain model with Hibernate 3 195
6.1 Hibernate ORM issues 196
   Fields or properties 196 • Hibernate entities and components 198
   Configuring object identity 200 • Using the cascade attribute 205
   Persisting interfaces 207
6.2 Other Hibernate issues 209
   Exception handling 209
   Lazy loading and inheritance hierarchies 209
6.3 Persisting a domain model class using Hibernate 212
   Writing Hibernate persistence tests with ORMUnit 213 • Testing persistent Hibernate objects 217 • Making a class persistent 224
6.4 Implementing a repository using Hibernate 228
   Writing a mock object test for a repository method 228 • Implementing HibernateRestaurantRepositoryImpl 231 • Writing the query that finds the restaurants 232 • Writing tests for a query 233
6.5 Hibernate performance tuning 234
   Using eager loading 235 • Using a process-level cache 240
   Using a query cache 241
6.6 Summary 242

7 Encapsulating the business logic with a POJO façade 243
7.1 Overview of a POJO façade 244
   An example POJO façade 245 • Benefits of a POJO façade 247
   Drawbacks of a POJO façade 248 • When to use a POJO façade and detached domain objects 250
7.2 POJO façade design decisions 251
   Encapsulating the domain objects 251 • Detaching objects 254
   Exceptions versus status codes 256 • Managing transactions and
   connections 257 • Implementing security 261
   Supporting remote clients 263
7.3 Designing a POJO façade's interface 264
   Determining the method signatures 264
7.4 Implementing the POJO façade 267
   Writing a test for a POJO façade method 267
   Implementing updateRestaurant() 270
7.5 Implementing a result factory 272
   Implementing a Hibernate result factory 273
   Implementing a JDO result factory 275
7.6 Deploying the POJO façade with Spring 279
   Generic bean definitions 280 • JDO-specific bean definitions 282
   Hibernate bean definitions 284
7.7 Summary 286

PART 3 VARIATIONS ................................................................. 287

8 Using an exposed domain model 289
8.1 Overview of the Exposed Domain Model pattern 290
   Applying the Exposed Domain Model pattern 291 • Benefits and
   drawbacks of this pattern 293 • When to use the Exposed Domain
   Model pattern 294
8.2 Managing connections using a Spring filter 295
8.3 Managing transactions 296
   Managing transactions in the presentation tier 297
   Managing transactions in the business tier 299
8.4 An example of the Exposed Domain Model pattern 304
   Servlet design 306 • JSP page design 309
   PlaceOrderService configuration 310
8.5 Using JDO with an exposed domain model 311
   Defining the Spring beans 311
   Configuring the web application 312
8.6 Using Hibernate with an exposed domain model 314
   Defining the Spring beans 314
   Configuring the web application 314

8.7 Summary 316

9 Using the Transaction Script pattern 317
   9.1 Overview of the Transaction Script pattern 318
      Applying the Transaction Script pattern 319
      Benefits and drawbacks of the Transaction Script pattern 322
      When to use the Transaction Script pattern 324
   9.2 Identifying the transaction scripts 325
      Analyzing the use case 325 • Analyzing the user interface
design 326 • The PlaceOrderTransactionScripts interface 327
   9.3 Implementing a POJO transaction script 329
      Writing a test for the transaction script 329
      Writing the transaction script 333
   9.4 Implementing the DAOS with iBATIS and Spring 337
      Overview of using iBATIS with Spring 339
      Implementing a DAO method 343
   9.5 Configuring the transaction scripts using Spring 354
      How Spring manages JDBC connections and transactions 354
      The Spring bean definitions 355
   9.6 Summary 358

10 Implementing POJOs with EJB 3 360
   10.1 Overview of EJB 3 361
      Key improvements in EJB 3 362 • Key limitations of EJB 3 368
   10.2 Implementing a domain model with EJB 3 372
      Mapping the classes to the database 372 • Implementing
repositories 380 • Testing the persistent EJB domain model 382
   10.3 Implementing a façade with EJB 3 385
      Turning a POJO façade into a session bean 386
      Detaching objects 387
   10.4 Assembling the components 389
      Using EJB dependency injection 390 • Integrating Spring and EJB
dependency injection 392 • Using Spring dependency injection 398
10.5 Implementing other patterns with EJB 3 400

Implementing the Exposed Domain Model pattern 400 • Implementing the Transaction Script pattern 401 • Implementing dynamic paged queries 401 • Implementing the concurrency patterns 403

10.6 Summary 403

PART 4 DEALING WITH DATABASES

AND CONCURRENCY 405

11 Implementing dynamic paged queries 407

11.1 Key design issues 408

Implementing a paging mechanism 410 • Generating queries dynamically 413 • Improving the performance of SQL queries 414

11.2 Implementing dynamic paged queries with iBATIS 418

Using queryForList() to select the rows 420
Using ROWNUM to select the rows 422

11.3 Implementing paged queries with JDO and Hibernate 424

Generating Hibernate and JDO queries dynamically 426 • Loading the data with a single SELECT statement 428 • Loading a subset of an object's fields 431 • Working with a denormalized schema 434
Implementing paging 435

11.4 A JDO design example 438

The JDOOrderRepositoryImpl class 439
The ExecuteFindOrdersQuery class 441

11.5 A Hibernate design example 442

The HibernateOrderRepositoryImpl class 443
The FindOrdersHibernateCallback class 444

11.6 Using JDO and Hibernate native SQL queries 446

Using JDO native SQL queries 446
Using Hibernate SQL queries 448

11.7 Summary 449
12 Database transactions and concurrency 451

12.1 Handling concurrent access to shared data 452

Using fully isolated transactions 453 • Optimistic locking 454
Pessimistic locking 458 • Using a combination of locking mechanisms 461

12.2 Handling concurrent updates in a JDBC/iBATIS application 462

Design overview 462 • Using optimistic locking 464 • Using pessimistic locking 466 • Using serializable or repeatable read transactions 466 • Signaling concurrent update failures 468

12.3 Handling concurrent updates with JDO and Hibernate 472

Example domain model design 472 • Handling concurrent updates with JDO 474 • Handling concurrent updates with Hibernate 478

12.4 Recovering from data concurrency failures 483

Using an AOP interceptor to retry transactions 484
Configuring the AOP interceptor 485

12.5 Summary 486

13 Using offline locking patterns 488

13.1 The need for offline locking 489

An example of an edit-style use case 490
Handling concurrency in an edit-style use case 490

13.2 Overview of the Optimistic Offline Lock pattern 492

Applying the Optimistic Offline Lock pattern 493 • Benefits and drawbacks 494 • When to use this pattern 494

13.3 Optimistic offline locking with JDO and Hibernate 495

Using version numbers or timestamps 495
Using detached objects 497

13.4 Optimistic offline locking with detached objects example 501.

Implementing the domain service 502 • Implementing the persistent domain class 504 • Detaching and attaching orders 505

13.5 The Pessimistic Offline Lock pattern 508

Motivation 508 • Using the Pessimistic Offline Lock pattern 509
Benefits and drawbacks 510 • When to use this pattern 511
13.6 Pessimistic offline locking design decisions  511

Deciding what to lock  512 • Determining when to lock and unlock the data  512 • Choosing the type of lock  512 • Identifying the lock owner  513 • Maintaining the locks  513 • Handling locking failures  519 • Using pessimistic offline locking in a domain model  520 • Implementing a lock manager with iBATIS  520
Implementing the domain service  522 • Adapting the other use cases  529

13.7 Summary  532

references  535
index  539