

A

abandoned sessions, Optimistic Offline Lock pattern 494
AbstractCouponImpl
 and Hibernate 208
 and JDO 157
AccountDAO 6
AccountRepository 18
Acegi Security
 implementing security with 249
 overview 262
Acknowledge Order use case 490
 example scenario 491
 impact of the Pessimistic Offline Lock pattern 530
 implementing with Hibernate 495
 implementing with JDO 495
 overview 56
acknowledgeOrder() 502
AcknowledgeOrderResult 502
AcknowledgeOrderService 501
AcknowledgeOrderServiceImpl 502
acquireLock() 514
addEntity() 449
addField() 187
addGroup() 186
adding behavior to a domain model 69–79
addJoin() 449
Address class 66
anemic domain model 10
annotation 363
anonymous callback class. *See* callback class
Ant, JDO bytecode enhancer 158
AOP. *See* Spring AOP

application
 identity 151
 responsibilities of 72
application server, replacing with a web container 27
application transaction. *See* long transaction
application-level locking 508
Aspect-Oriented Programming. *See* Spring AOP
assertAllFieldsMapped()
 Hibernate version 214
 JDO version 160
assertApplicationIdentity(), JDO version 160
assertClassMapping()
 Hibernate version 214
 JDO version 160
assertDatabaseSchema() 161
 Hibernate version 215
assertField(), Hibernate version 214
assertIdField()
 Hibernate version 214
 JDO Version 160
assertXPathEvaluatesTo() 139
association. *See* mapping
attachCopy() 498
@AttributeOverride 376

B

BankingTransaction 15
BankingTransactionRepository 18
BasicDataSource, example configuration 356
bean factory 21
BeanNameAutoProxyCreator
 example configuration 23
 example of use 22

- BeanNameAutoProxyCreator (*continued*)
 - using to apply an interceptor to a POJO façade 282
 - begin() 120
 - beginTransaction() 119–120
 - bloated session state 499
 - Burlap 263
 - business logic
 - design options 32–34
 - encapsulating with a POJO façade 244–250
 - implementing simple business logic 324
 - business logic encapsulation
 - with an EJB 3 session bean 385–388
 - options 37–41
 - using a POJO façade 39
 - using an EJB session façade 38
 - using an exposed domain model 40
 - using the Exposed Domain Model pattern 290–294
 - business logic organization
 - options 35, 37
 - procedural approach. *See* Transaction Script pattern
 - with the Transaction Script pattern 318–324
 - using the Domain Model pattern. *See* Domain Model pattern
 - using the Transaction Script pattern. *See* Transaction Script pattern
 - business transaction. *See* long transaction
 - bytecode enhancer, in JDO 158
- C**
-
- <cache> element 240
 - caching overview. *See* process-level caching; query cache
 - callback class
 - example of a named callback 130
 - example of an anonymous callback 130
 - improving testability with named callback classes 130
 - problems with anonymous 129
 - CannotAcquireLockException 469
 - CannotSerializeTransactionException 469
 - @Cascade
 - JBoss extension 370
 - cascade attribute 203, 205–206
 - CascadeType.DELETE_ORPHAN
 - JBoss extension 370
 - checkConsistency() 500
 - checkout request 72
 - <class>
 - detachable attribute 505
 - optimistic-lock attribute 479
 - class, well-designed 36
 - <class> element
 - in JDO metadata 152
 - class hierarchy, persisting with EJB 3 378
 - classic EJB architecture. *See* heavyweight approach
 - clear() 303
 - close() 119
 - coarse-grained API, benefits 38
 - Coarse-Grained Lock pattern 512
 - collaborator 69
 - challenges caused by 76
 - determining 70
 - commit() 120
 - defined by PlatformTransactionManager 259
 - component collection, in Hibernate 204
 - <component> element 227
 - <composite-element> element 204
 - concurrency
 - in databases 452
 - See also* database concurrency
 - ConcurrencyFailureException 469
 - recovering from 483
 - concurrent access to shared data 452
 - concurrent updates 453–461, 472–486
 - signalling failures 468
 - in Hibernate 482
 - in JDO 477
 - Configuration 214
 - generateSchemaUpdateScript() 216
 - getClassMapping() 214
 - openSessionFactory() 214
 - connection management
 - in a POJO façade 257
 - using a servlet filter 295
 - connection pool 358
 - using BasicDataSource 356
 - constructor injection 26
 - domain model service example 83
 - and entities 89
 - and object/relational mapping frameworks 89
 - container-managed transactions, replacing with Spring-managed transactions 20
 - ContextLoaderServlet 314
 - Coupon 66, 103
 - mapping with Hibernate 208
 - persisting with EJB 3 378
 - persisting with JDO 156
 - createCriteria() 427
 - createEntityManager() 367

- createEntityManagerFactory() 367
 - createNamedQuery 381
 - createNamedQuery() 119, 367
 - createQuery() 119, 367
 - createSQLQuery() 448
 - Criteria
 - scroll() 437
 - setFetchMode() 430
 - setFirstResult() 437
 - setMaxResults 437
 - criteria query 120, 427
 - eager loading 430
 - implementing paging 437
 - currentTransaction() 119–120
- D**
-
- DAO. *See* data access object
 - data access object
 - iBATIS example 418–424
 - implementing with iBATIS SQL Maps 337–354
 - Spring 337, 354
 - and the Transaction Script pattern 320
 - data transfer object
 - defined 10
 - implementing 335
 - replacing with a detached object 18
 - and the Transaction Script pattern 319
 - DataAccessException 469
 - database, in-memory. *See* in-memory database
 - database access
 - design options 32–34
 - with EJB 3 365
 - with Hibernate and JDO 117, 125
 - with iBATIS SQL Maps 42
 - with an object/relational mapping framework 108–116
 - problems with using JDBC directly 41
 - using an object/relational mapping framework 43
 - database concurrency 452
 - Optimistic Offline Lock pattern 46
 - options 44–46
 - overview 44–46
 - Pessimistic Offline Lock pattern 47
 - Database Connection Pool. *See* DBCP
 - database schema
 - testing in Hibernate 215
 - testing in JDO 161
 - DataSource, configuring the isolation level 467
 - DataSourceTransactionManager 260, 354
 - example configuration 356
 - datastore identity 153
 - datastore transactions 474
 - using 476
 - <datastore-identity> element 154
 - DBCP 358
 - configuring the isolation level 467
 - DbUnit 136, 351
 - default fetch group 184
 - attribute 185
 - delete() 119
 - defined by HibernatePersistenceTests 216
 - deletePersistent() 119
 - defined by JDOPersistenceTests 164
 - denormalization 415
 - and object/relational mapping framework 116, 434
 - reasons to 116
 - dependency injection 26
 - @EJB 3 annotation 365
 - assembling an EJB 3 application 390
 - assembling applications with 25–27
 - constructor injection domain model service example 83
 - See also* constructor injection
 - in EJB 3 365
 - entities 89
 - object/relational mapping frameworks 89
 - as a replacement for JNDI 26
 - setter injection. *See* setter injection
 - @Depends, JBoss extension 396
 - deployment, of a POJO application 27
 - design, improving by refactoring 75
 - design decision 48, 51
 - overview 33
 - summary 47
 - detachable attribute 505
 - detachCopy() 254, 498
 - detachCopyAll() 254
 - detached object
 - attaching and detaching in JDO and Hibernate 498
 - avoiding by using the Exposed Domain Model pattern 290
 - bloated session state 499
 - configuring detachable Hibernate objects 505
 - configuring detachable JDO objects 504
 - detaching in a façade 19
 - detaching in EJB 3 387
 - detaching in Hibernate 273
 - detaching in JDO 275

- detached object (*continued*)
 - drawbacks of 249
 - edit-style use cases 497
 - in EJB 3 368
 - encapsulating detachment code 505
 - in Hibernate 255
 - Hibernate example 507
 - how to use 254
 - in JDO 254
 - JDO example 506
 - Optimistic Offline Lock pattern 497
 - overview 18, 114
 - problems with 290
 - storing in HttpSession 498
 - development
 - accelerating by using the Exposed Domain Model pattern 293
 - simplifying by using a POJO façade 247
 - simplifying development by using POJOs 16
 - <discriminator> element
 - in Hibernate 208
 - in JDO 157
 - @DiscriminatorColumn 379
 - discriminatorValue member 379
 - distributed transaction
 - initiated by a remote client 248
 - dogma, avoiding 5
 - doInTransaction()
 - defined by HibernatePersistenceTests 216
 - defined by JDOPersistenceTests 163
 - domain model
 - adding behavior 69–79
 - anemic 10
 - developing 68–79
 - encapsulating with adapters 253
 - encapsulating with interfaces 251
 - entity. *See* entity
 - example of implementing 80–92
 - factory. *See* factory
 - identifying classes, attributes and relationships 69
 - implementing with EJB 3 372–385
 - mapping to a database 96–108
 - persisting with EJB 3 372–380
 - persisting with Hibernate 212–227
 - persisting with JDO 159–173
 - place in architecture 63
 - repository. *See* repository
 - roles 66–68
 - service. *See* service
 - testing a service 84
 - testing an entity 91
 - testing with EJB 3 382–385
 - value object. *See* value object
 - See also* persistent domain model
 - Domain Model pattern
 - example structure 37
 - overview 36, 62–68
 - domain model service
 - converting to an EJB 3 session bean 391
 - detached objects example 502
 - Pessimistic Offline Lock pattern example 525
 - DomainRestaurantNotificationService 473
 - downcasting, and Hibernate 209
 - DTO. *See* data transfer object
 - dumb data objects 35
 - dynamic mapped statement
 - example 421
 - overview 413
 - dynamic paged queries
 - in EJB 3 401
 - Hibernate example 442–446
 - implementing 424–438
 - JDO example 438–442
-
- E**
- eager loading 428
 - benefits of 113
 - configuring 142–144, 235
 - difficulty using 113
 - and Hibernate 235–240
 - Hibernate criteria queries 430
 - and JDO fetch groups 184–191
 - overview 112
 - problems caused by excessive eager loading 113
 - using fetch groups 429
 - using joins 113
 - See also* lazy loading
 - EasyMock 78
 - Eclipse plugin, JDO bytecode enhancer 159
 - edit-compile-debug cycle
 - accelerating by using POJOs 16
 - problems with long cycles 10
 - edit-style use case
 - handling concurrency 490
 - implementing with detached objects 497
 - EHCACHE 240, 535
 - @EJB 365
 - EJB 3
 - application assembly 389–400
 - automatic detachment 371
 - collections of non-entities 368

- EJB 3 (*continued*)
 - compared to EJB 2 362–368
 - concurrency options 403
 - deleting orphans 369
 - dependency injection 365, 390–392
 - dependency injection limitations 371
 - detached objects 368, 387
 - development environment complexity 372
 - domain model service as a session bean 391
 - dynamic paged queries 401
 - EJB QL query example 382
 - encapsulating business logic 385–388
 - entity bean example 373, 376
 - entity beans 362
 - Exposed Domain Model pattern 400
 - fetch joins 370
 - Hybrid heavyweight and POJO approach 32
 - implementing a façade 385–388
 - implementing repositories 380–382
 - injecting Spring beans 392–398
 - injecting the EntityManager into repositories 397
 - integrating with Spring dependency injection 392–398
 - key improvements 362–368
 - limitations 368–372, 379
 - mapping a domain model 372–380
 - mapping an interface 378
 - mapping lists 369
 - object/relational mapping 365
 - object/relational mapping rules 364
 - overview 361–372
 - persistence API 366
 - persisting a class hierarchy 378
 - promise of 11
 - repositories as session beans 392
 - session bean example 386
 - session façade 385
 - simplified configuration 363
 - testing domain model 382–385
 - Transaction Script pattern 401
 - using Spring dependency injection 398
- EJB 3 annotation
 - @AttributeOverride 376
 - @DiscriminatorColumn 379
 - @EJB 365
 - @Embedded 376
 - @Entity 363
 - @Id 364
 - @Inheritance 379
 - @Local 363
 - @ManyToOne 364
 - @NamedQuery annotation 382
 - @OneToMany 364
 - @OrderBy 369
 - @PersistenceContexts 396
 - @PostConstruct 400
 - @Resource 396
 - @Stateless 363
- EJB container
 - eliminating by using a POJO façade 247
 - eliminating with the Exposed Domain Model pattern 293
- EJB QL query, example 382
- EJB. *See* Enterprise JavaBeans
- EJB session façade, overview 38
- EJB3PendingOrderPersistenceTests 383
- EJB3PendingOrderRepository 392
- EJB3PlaceOrderFacadeResultFactory 387
- EJB3RestaurantRepository 380
- @Embedded 376
- Embedded Value pattern 98
- encapsulation
 - issues with POJO façade 250
 - issues with the Exposed Domain Model pattern 293
- enterprise application
 - development challenges 7
 - example architecture 33
- Enterprise JavaBeans
 - avoiding entity beans 7
 - disillusionment with 5
 - EJB 3 compared to EJB 2 362–368
 - history of 5
 - inhibiting test-driven development 75
 - limitations of entity beans 9
- Enterprise JavaBeans (*continued*)
 - making development for difficult 7
 - pain of development 10
 - POJOs as an alternative 12
 - problems 7–11
 - reasons for a procedural design 9
 - session façade example 6
 - session façade with EJB 3 385
 - typical application architecture 6
 - when to use 12
- @Entity 363
- entity
 - constructor injection 89
 - defined 67
 - example test 91
 - implementing 87–92

- entity (*continued*)
 - mocking 84
 - persisting with EJB 3 372–380
 - persisting with Hibernate 212–227
 - persisting with JDO 159–173
 - in Hibernate 202
 - See also* Coupon; Order; PendingOrder; Restaurant entity
- EntityManager 366
 - createNamedQuery 381
 - createNamedQuery() 367
 - createQuery() 367
 - find() 381
 - injecting into repositories 397
- EntityManagerFactory 367
 - createEntityManager() 367
 - example 383
- EntityTransaction 367
 - example 383
- eq() 79
- evict, objects from PersistenceManagerFactory-
 - level cache 192
- exception handling
 - in Hibernate 209
 - in a POJO façade 256
- exception mapping, SqlMapClientTemplate 470
- exclusive lock 513
- exclusive write lock 512
- execute(), defined by JDO Query 120
- ExecuteFindOrdersQuery 441
- ExecuteNamedQueryWithMapCallback 174
- Exposed Domain Model pattern
 - benefits 293
 - compared to a POJO façade 247
 - connection management 295
 - drawbacks 293
 - encapsulation problems 293
 - example 304
 - with Hibernate 314–315
 - in EJB 3 400
 - with JDO 311–314
 - overview 40, 290–294
 - Spring beans for a domain service 310
 - transaction management 296–311
 - when to use 294
- fetch group 184–191
 - alternative to. *See* projection query
 - configuring with Spring AOP 187
 - eager loading 429
 - partial object loading 432
 - using to detach objects 275
- fetch join 237, 430
- FetchConfiguration 186
 - addField() 187
- <fetch-group> element 185
- FetchPlan 186
 - addGroup() 186
 - setFetchGroup() 432
- find() 381
- findByNamedQueryAndNamedParam() 228
- findOrCreatePendingOrder(), of the
 - PendingOrderRepository 82
- findOrders()
 - defined by OrderDAO 419
 - defined by OrderRepository 438
- FindOrdersHibernateCallback 444
- FIRST_ROWS 415
- flush() 479, 500
- Food to Go application
 - architecture 49
 - business logic 51
 - database access 52
 - database schema 146
 - domain model 64
 - handling concurrency 52
 - high-level decisions 51–53
 - overview 48–50
 - requirements 48
 - users 48
- FoodToGoDomainMappingTests, JDO version 164
- FoodToGoHibernateMappingTests, Hibernate
 - version 217
- FoodToGoSchemaTests 219
- FoodToGoSchemaValidationTests 166
- Fowler, Martin
 - anemic domain model 10
 - naming POJO 12
- FreeShippingCoupon
 - and Hibernate 208
 - mapping to the database 103
 - persisting with JDO 156
- fully isolated transactions 454

F

- façade. *See* EJB Session Façade; POJO façade
- factory, defined 67
- fetch attribute 235

G

- gatekeeper. *See* POJO façade
- generateSchemaUpdateScript() 216

- <generator> element 200
- getCause() 478
- getClass(), defined by Hibernate 210
- getFetchConfiguration() 187
- getFetchPlan() 186
- getNamedQuery(), to create an SQL query 448
- getObject() 154
- getObjectById() 119
 - defined by JDOPersistenceTests 164
- getOrderToAcknowledge() 502
- getResultList() 367
- getTransaction() 259
- getVersion 497
- global transaction. *See* JTA transaction
- Gold Master approach 352

H

- heavyweight approach 32
- Hessian 263
- HessianServiceExporter 263
- Hibernate 43
 - the class 210, 256
 - compared to Java Data Objects 124
 - component collections 204
 - components 198
 - concurrent updates 478–483
 - Configuration 214
 - configuring detachable objects 505
 - configuring eager loading 142
 - configuring the isolation level 481
 - criteria queries 427
 - defining the object/relational mapping 225
 - deleting orphans 203
 - denormalized schema 434
 - detached objects 124
 - detaching and attaching objects 255, 498
 - dynamic eager loading with fetch joins 237
 - eager loading 235–240
 - entities 198
 - entity collection 202
 - example HQL query 232
 - example of a persistent class 224
 - example of detaching and attaching objects 507
 - example of dynamic paged query 442–446
 - example of persistence tests 219
 - example of pessimistic locking 480
 - example of query generation 444
 - example repository 231, 443
 - exception handling issues 209
 - executing queries 120
 - Exposed Domain Model pattern 314–315
 - extra SQL statement when persisting a
 - collection 204
 - fetch attribute 235
 - fetch join 430
 - fields versus JavaBean properties 196
 - generating persistent identifiers 201
 - getClass() 210
 - identifier properties 201
 - implementing a result factory 273
 - implementing dynamic paged queries 424–438
 - implementing repositories 228–234
 - implementing the Acknowledge Order use case 495
 - implementing the Modify Order use case 522
 - initialize() 256
 - lazy loading inheritance hierarchies 209
 - lazy property loading 211
 - and long transactions 495
 - mapping issues and decisions 196–208
 - mapping JavaBean properties 196
 - mapping one-to-many relationships 199, 202
 - native SQL queries 120, 448
 - nontransactional reads 301
 - object identity 200
 - optimistic locking 479
 - Optimistic Offline Lock Pattern 495–500
 - overview of API 118
 - overview of eager loading 121
 - overview of mapping 117
 - overview of performance tuning 141–146
 - overview of process-level caching 122
 - overview of transaction management API 120
 - pagination 437
 - performance tuning 234–241
 - pessimistic locking 480–481
 - problems with downcasting 209
 - problems with instanceof 209
 - process-level caching 240
 - process-level caching. *See* process-level caching
 - projection queries 434
 - proxy 209
 - Query 118
 - query cache 123, 241
 - query generation 427
 - recovering from HibernateExceptions 303
 - Session 118
 - SessionFactory 118
 - signalling concurrent update failures 482
 - surrogate key 201
 - testing an HQL query 233

- testing persistent objects 216–224
 - Transaction 118
 - using native SQL queries 446–449
 - using with Spring 126
 - writing tests for 213
 - cascade attribute 203, 205
 - <component> element 227
 - <composite-element> element 204
 - <discriminator> element 208
 - example 225
 - fetch attribute 235
 - <generator> element 200
 - <hibernate-mapping> 226
 - <id> element 200
 - <key> element 203
 - <list> element 203
 - <many-to-one> element 227
 - <property> element 197
 - <query> element 232
 - <subclass> element 208
 - hibernate.connection.isolation property 481
 - <hibernate-mapping> 226
 - HibernateMappingTests 140, 214
 - HibernateOrderRepositoryImpl 442
 - HibernatePendingOrderPersistenceTests 219
 - HibernatePersistenceTests 140, 216
 - HibernatePlaceOrderFacadeResultFactory 273
 - HibernateRestaurantRepositoryImpl 231
 - HibernateRestaurantRepositoryImplMockTest 229
 - assertDatabaseSchema 215
 - configuring to use a custom SQLExceptionHandler 482
 - example configuration 284
 - example of a repository using 231
 - exception mapping 482
 - findByNamedQueryAndNamedParam() 228
 - implementing a repository with 128
 - update() 498
 - example configuration 284
 - HQL 119
 - example of 232
 - obstacles to using for testing 138
 - using for testing 138
 - HTTP request, and POJO façade 264
 - HttpServletResponse, transaction rollbacks 298
 - bloated session state 499
 - storing detached objects 498
 - storing version number 496
-
- I**
- 342
 - iBatis descriptor file
 - benefits 42
 - dynamic mapped statements 413
 - example DAO 418, 424
 - example of a dynamic mapped statement 421
 - implementing a lock manager 520
 - implementing concurrency 462
 - implementing DAOs 337–354
 - implementing dynamic paged queries 418–424
 - nested SELECT statement example 348
 - nested SQL statement 342
 - optimistic locking 464
 - overview 339
 - pessimistic locking 466
 - repeatable read transactions 466
 - serializable transactions 466
 - testing 343, 350
 - when to use 115
 - iBatis SQL Maps configuration file
 - <sqlMapConfig> 358
 - iBatis SQL Maps descriptor file
 - 342, 422
 - example 348
 - writing 341
 - @Id 364
 - <id> element 200
 - identifying classes, attributes and relationships 69
 - identifying methods 72
 - Identity Map pattern 109
 - identity-type attribute
 - and application identity 152
 - implementing a domain entity 87–92
 - implementing a domain service method 80–86
 - <implements> element 157
 - Implicit Lock pattern 511
 - inconsistent reads 453
 - preventing with fully isolated transactions 453
 - <index> element 203
 - @Inheritance 379
 - discriminatorValue 379
 - initialize() 256
 - in-memory database 138
 - insert() 341

- instanceof, and Hibernate 209
 - interceptor. *See* proxy
 - interface
 - mapping with EJB 3 378
 - mapping with Hibernate 207
 - mapping with JDO 155
 - IntIdentity 153
 - <isEmpty> 422
 - isolated transactions
 - overview 44
 - isolation level
 - configuring 467
 - repeatable read 453
 - serializable 453
 - specifying in Hibernate 481
 - specifying in JDO 477
 - isRestaurantAvailable()
 - method of the Restaurant Repository class 88
 - <iteration> 422
- J**
-
- Java Data Objects
 - jdo file 171
 - .orm file 171
 - API overview 118
 - application identity 151
 - bytecode enhancer 158
 - compared to Hibernate 124
 - configuring detachable objects 504
 - configuring eager loading 142
 - configuring optimistic locking 476
 - configuring the isolation level 477
 - configuring the transaction type 474
 - datastore identity 153
 - datastore transactions 474
 - default fetch group 184
 - defining a custom fetch group 185
 - defining the object/relational mapping 170
 - deleting orphans 173
 - denormalized schema 434
 - detached objects 124
 - detaching and attaching objects 254, 498, 506
 - dynamic paged query example 438–442
 - eager loading overview 121
 - executing queries 120
 - Exposed Domain Model pattern 311–314
 - fetch group 184–191
 - handling concurrent updates 474–478
 - implementing a result factory 275
 - implementing dynamic paged queries 424–438
 - implementing repositories 173–183
 - implementing the Acknowledge Order use case 495
 - implementing the Modify Order use case 522
 - IntIdentity 153
 - issues and limitations 150–159
 - javax.jdo.option.Optimistic 474
 - and long transactions 495
 - mapping interfaces 155
 - mapping overview 117
 - named JDO query example 180
 - native SQL queries 120
 - nontransactional reads 301
 - object identity 151
 - Optimistic Offline Lock Pattern 495–500
 - optimistic transactions 474
 - overview 117–125
 - package.jdo 171
 - paging 436
 - partial object loading 432
 - performance tuning 141–146, 183–193
 - persistence tests example 166
 - PersistenceManager 118
 - PersistenceManagerFactory 118
 - persistent class example 170
 - process-level caching 122, 191
 - See also* process-level caching
 - projection queries 433
 - Query 118
 - query cache 123, 193
 - query generation 426, 439
 - repository example 178, 439
 - ResultSet handling 436
 - signalling concurrent update failures 477
 - single field identity 152
 - SQL projection query 447
 - testing a JDOQL query 180
 - testing persistent objects 164–170
 - Transaction 118
 - transaction management API overview 120
 - using datastore transactions 476
 - using fetch groups 429
 - using native SQL queries 446–449
 - using optimistic transactions 475
 - using with Spring 126
 - writing tests for 159, 164
 - Java Naming and Directory Interface. *See* JNDI
 - javax.jdo.option.NontransactionalRead
 - property 301
 - javax.jdo.option.Optimistic 474

- JBoss
 - @Cascade 370
 - @Depends 396
 - @OrderBy 369
 - @Service 395–396
 - service POJO 395
 - when to use 28
 - JBoss Cache 240
 - JDBC
 - implementing concurrency 462
 - problems with using directly 41
 - replacing with a mapping framework 41
 - replacing with iBATIS SQL Maps. *See* iBATIS SQL Maps
 - JDBCException 482
 - .jdo file 171
 - JDO. *See* Java Data Objects
 - JDO version 160
 - JDO XML metadata
 - JDO metadata example 170
 - writing JDO XML metadata 170
 - JDODetachedObjectAccessException 254
 - JDOException 477
 - getCause() 478
 - mapped by JdoTemplate 478
 - JDOHelper 497
 - getObject() 154
 - getVersion 497
 - JDOMappingStrategy 159
 - JDOMappingTests 140, 160
 - JDOOptimisticVerificationException 478
 - JDOOptimisticVerificationFailedException 475
 - JDOOrderAttachmentManager 506
 - JDOOrderRepositoryImpl 439
 - JDOPendingOrderPersistenceTests 166
 - JDOPersistenceTests 140
 - deletePersistent() 164
 - doInTransaction() 163
 - getObjectById() 164
 - makePersistent() 163
 - overview of 163
 - JDOPlaceOrderFacadeResultFactory 277
 - JDOQL 119
 - example of 180
 - range clause 436
 - JDORestaurantRepositoryImpl 178
 - JDORestaurantRepositoryImplTests 176
 - JDORestaurantRepositoryQueryTests 181
 - JDOSchemaTests 161
 - assertDatabaseSchema() 161
 - JdoTemplate 126
 - attachCopy() 498
 - configuration example 282
 - configuring to use a custom SQLExceptionHandler 478
 - detachCopy() 498
 - implementing a repository with 128
 - mapping JDOExceptions 478
 - repository using example 178
 - using to implement a repository 174
 - JdoTransactionManager 260
 - example configuration 282
 - Jetty 27
 - jMock 78
 - implementing mock objects with 78
 - Mock 79
 - MockObjectTestCase 79
 - testing a POJO façade 268
 - testing a transaction script 376
 - testing an iBATIS DAO 390
 - See also* mock object
 - JNDI
 - binding references 395
 - Reference 393
 - replacing lookups with dependency injection 365
 - replacing with dependency injection 25
 - JndiObjectFactoryBean 397
 - lookupOnStartup property 397
 - proxyInterface property 397
 - joins 113
 - JPOXMappingStrategy 161
 - JSR12. *See* Java Data Objects
 - JSR243. *See* Java Data Objects
 - JTA transaction
 - and the Exposed Domain Model pattern 301
 - managing with Spring 257, 261
 - JtaTransactionManager 261
 - JUnit 76
 - DbUnit extension 351
 - XmlUnit extension 139
- K**
-
- <key> element 203
 - Kodo JDO 186
 - configuring optimistic locking 475
 - detaching objects 276
 - fetch groups 186
 - process-level caching 191

Kodo JDO (*continued*)
 query caching 193
 ResultSet handling 436
 kodo.DataCache property 192
 kodo.QueryCache property 193
 kodo.RemoteCommitProvider 192
 KodoFetchGroupInterceptor 187
 KodoJDOAttachObjectCallback 506
 KodoJDODetachObjectCallback 276, 506
 KodoPersistenceManager 187
 setQueryCacheEnabled() 193
 KodoQuery 193

L

law of unintended consequences 318
 lazy loading
 benefits of 113
 in Hibernate 209
 Hibernate inheritance hierarchies 209
 overview 112
See also eager loading
 lazy property loading, in Hibernate 211
 lightweight approach 13
 lightweight container. *See* Spring framework
 lightweight framework
 making POJOs transactional 19–25
 persistence 12
 persisting POJOs 16–18
 providing services for POJOs 12
See also Hibernate; iBATIS SQL maps; JDO; non
 intrusive framework; Spring framework
 transaction management 13
 limitations 461
 <list> element
 mapping a component collection 204
 mapping an entity collection 203
 list(), defined by Hibernate Query 120
 load() 119
 defined by HibernatePersistenceTests 216
 and pessimistic locking 480
 @Local 363
 local transaction, managing with Spring 257
 LocalPersistenceManagerFactoryBean
 configuring the JDO transaction type 475
 example configuration 282
 exposed domain model example
 configuration 311
 LocalSessionFactoryBean, example
 configuration 284

lock manager
 alternative to object locks 514
 benefits and drawbacks 516
 iBATIS implementation 520
 implementing 516
 Pessimistic Offline Lock pattern 514
 using 515
 lock(), and pessimistic locking 480
 locking
 application-level 508
 database-level 458–461
 LockManager, interface 514
 LockManagerIBatisImpl 520
 LockMode.UPGRADE 480
 LockMode.UPGRADE_NO_WAIT 480
 long database transactions, drawbacks of 491
 long transaction
 handling concurrency 46–47
 with Hibernate 495
 with JDO 495
 overview of handling concurrency 489–492
See also edit-style use case; Optimistic Offline
 Lock pattern; Pessimistic Offline Lock pattern
 lookupOnStartup property 397
 loop back calls 9
 lost updates 452
 preventing with fully isolated transactions 453
 preventing with optimistic locking 454–458
 preventing with pessimistic locking 458–461

M

makePersistent() 119
 defined by JDOPersistenceTests 163
 @ManyToOne 364
 targetEntityMember 379
 <many-to-one> element 227
 mapping
 class 97
 class to its own table 98
 class to multiple tables 99
 class to parent's table 98
 collections 101
 inheritance 103–106
 relationships 99–103
 using a relationship using join table 102
 mapping the Coupon hierarchy 103
 MENU_ITEM table 146–147
 MenuItem 65

message-driven bean 248
 MethodSecurityInterceptor 262
 Mock 79
 proxy() 79
 verify() 79
 mock object 76
 example of using 84
 testing a DAO method 343
 testing a Hibernate repository 228
 testing a JDO repository 174
 testing a transaction script 329
 testing repositories with mock objects 140
 using to test a POJO façade 267
 mock object. *See* jMock
 MockObjectTestCase 79
 eq() 79
 returnValue() 79
 Modify Order use
 implementing with Hibernate 522
 implementing with JDO 522
 Modify Order use case 508
 domain model implementation 522
 implementing with the Pessimistic Offline Lock
 pattern 520–532
 one approach 58
 overview 57
 ModifyOrderResult 524
 ModifyOrderService 522
 ModifyOrderServiceImpl 524
 money transfer example
 heavyweight example 6
 lightweight version 12
 MyExampleCallback 130
 MyOracleSQLException
 Translator 471

N

named query, JDO example of 180
 @NamedQuery annotation 382
 Nan 489
 native SQL query 120
 natural key 108
 newNamedQuery() 119
 newObjectIdInstance() 154
 newQuery() 119
 nondurable identity 151
 nonintrusive framework. *See* lightweight
 framework
 nontransactional reads 301

O

object identity
 in Hibernate 200
 in JDO 151
 and persistence 107
 object loading, partial 432
 object locks 518
 object model. *See* domain model
 object query 119
 See also criteria query; HSQL; JDOQL
 object/relational
 mismatch 96
 object/relational mapping
 defining with Hibernate 225
 defining with JDO 170
 Hibernate example 225
 Hibernate test example 217
 JDO example 170
 JDO test example 164
 mapping a domain model to a schema 96–108
 mapping to an existing database schema 117
 overview of performance tuning 141–146
 potential bugs 133
 testing 133, 139
 testing in Hibernate 214
 testing in JDO 160
 verifying that it matches the schema 137
 See also Hibernate; Java Data Objects; persistent
 domain model
 object/relational mapping framework 16
 API for creating, loading and deleting
 objects 111
 benefits and drawbacks
 114–116
 caching. *See* caching
 connection factory interface 118
 connection interface 118
 and DBAs 116
 declarative mapping between object model and
 schema 110
 denormalized schema 434
 dependency injection 89
 eager loading. *See* eager loading
 generating database schema from object/
 relational mapping 117
 implementing dynamic paged queries 424, 438
 key features 109–114
 lazy loading. *See* lazy loading
 limitations 116
 overview 108–116

- object/relational mapping framework (*continued*)
 - problems solved by 109
 - query generation 426
 - query interface 118
 - query language 112
 - relationship with the rest of the application 110
 - transaction interface 118
 - transaction management 112
 - using 43
 - See also* detached objects
- ObjectOptimisticLockingFailureException 500
- object-oriented analysis and design 62
- object-oriented approach. *See* Domain Model pattern
- object-oriented design 14
 - avoiding with transaction scripts 322
 - back in fashion 62
 - benefits 15
 - defined 8
 - going out of fashion 7
- offline locking. *See* long transaction
- OFFLINE_LOCK 520
- @OneToMany 364
- one-to-many relationship, mapping with
 - Hibernate 202
- OOAD. *See* object-oriented analysis and design
- Open PersistenceManager in View pattern. *See* Exposed Domain Model pattern
- Open Session in View pattern. *See* Exposed Domain Model pattern
- OpenPersistenceManagerInViewFilter 296
 - example configuration 312
- openSessionFactory() 214
- OpenSessionInViewFilter 296
 - deferred close mode 303
- OpenSessionInViewInterceptor 296
- optimistic locking
 - benefits and drawbacks 457
 - comparing columns 455
 - in EJB 3 403
 - example scenario 456
 - Hibernate 479
 - with iBATIS SQL Maps 464
 - implementing efficiently 456
 - in JDO 475
 - limitations 492
 - overview 45, 454–458
 - timestamp column 455
 - tracking changes 455
 - version column 455
 - when to use 458
- optimistic locking. *See* Optimistic Offline Lock pattern
- Optimistic Offline Lock pattern
 - benefits and drawbacks 494, 496, 498
 - with detached objects 497
 - detached objects example 508
 - in EJB 3 403
 - example scenario 493
 - extending to work with the Pessimistic Offline Lock pattern 531
 - as an extension of optimistic locking 492
 - handling locking failures 499
 - overview 46, 492–495
 - with version numbers or timestamps 495
 - when to implement with detached objects 500
 - when to use 494
- Optimistic Offline Locking pattern
 - when to implement with version numbers or timestamps 497
- optimistic transactions 474
 - using 475
- optimistic-lock attribute 479
- OptimisticLockingFailureException 469
- optimizer hint 115, 415
- ORA-00054 470
- ORA-00060 461
- ORA-08177 470
- Oracle
 - optimizer hints 415
 - ORA-00054 470
 - ORA-00060 461
 - ORA-08177 470
 - pessimistic locking 459
 - ROWNUM 416
 - SELECT FOR UPDATE 459
- Order 66, 504
 - making detachable with Hibernate 505
 - making detachable with JDO 504
- Order page 265
- OrderAttachmentManager 502, 505
- @OrderBy 369
 - JBoss extension 369
- OrderDAO 419, 464
 - iBATIS SQL Maps implementation 418–424
- OrderDAOIBatisImpl 420, 464
- OrderRepository 66, 438
 - findOrders() 438
 - Hibernate implementation 442–446
 - JDO implementation 438–442
- OrderSearchCriteria 420

- OrderSummaryDTO 420
 - .orm file 171
 - ORM. *See* object/relational mapping
 - ORMUnit
 - Hibernate version 213
 - JDO version 159–164
 - overview 140
 - orphans
 - deleting in EJB 3 369
 - deleting in Hibernate 203
 - deleting in JDO 173
 - OverdraftPolicy 15
- P**
-
- package.jdo file 171
 - PagedQueryResult 420
 - paging
 - criteria queries 437
 - Hibernate 437
 - implementation options 410–413
 - JDO 436
 - Kodo JDO 436
 - by navigating through the ResultSet 416
 - object/relational mapping framework 435
 - using ROWNUM 416
 - with ScrollableResults 437
 - partial object loading, in JDO 432
 - PaymentInformation 66
 - PENDING_ORDER table 147
 - PendingOrder 65
 - detaching with Hibernate 273
 - detaching with JDO 275
 - implementing 87–92
 - implementing with EJB 3 373
 - persistence tests 166, 219
 - persisting with Hibernate 224
 - persisting with JDO 170
 - testing 87
 - updateDeliveryInfo() 87
 - verifying the Hibernate object/relational mapping 217
 - verifying the JDO object/relational mapping 164
 - PendingOrder.hbm.xml 226
 - PendingOrder.xml 348
 - PendingOrderAdapter 253
 - PendingOrderDAOIbatisImpl 345
 - PendingOrderDAOIbatisImplMockTests 344
 - PendingOrderDetail 252
 - PendingOrderDTO 327
 - PendingOrderLineItemDetail 252
 - PendingOrderRepository 82
 - findOrCreatePendingOrder() 82
 - PendingOrderRespository 66
 - PendingOrderTests 91
 - PercentageDiscountCoupon
 - mapping to the database 103
 - persisting with JDO 156
 - per-field fetch configuration
 - in Kodo JDO 186
 - using to detach objects 276
 - performance
 - eager loading 142, 428
 - and the Hibernate fetch attribute 235
 - and Hibernate fetch joins 237
 - improving performance of an object/relational mapping framework 141–146
 - improving performance with optimizer hints 415
 - improving performance with ROWNUM 416
 - and JDO fetch groups 184–191
 - optimizing with a Hibernate process-level cache 240
 - optimizing with a JDO PersistenceManager
 - Factory-level cache 191
 - partial object loading in JDO 429
 - performance benefits of an object/relational mapping framework 115
 - and process-level caching 145
 - and query caching 145
 - rewriting SQL queries 418
 - SQL query optimization 414–418
 - tuning in a Hibernate application 234–241
 - tuning in a JDO application 183–193
 - using denormalization to improve performance 415
 - Persistence 366
 - createEntityManagerFactory() 367
 - example 383
 - persistence, transparent persistence 16
 - persistence framework. *See* object/relational mapping framework
 - @PersistenceContexts 396
 - PersistenceManager 118
 - checkConsistency() 500
 - close() 119
 - currentTransaction() 119
 - deletePersistent() 119
 - detachCopy() 254
 - detachCopyAll() 254

- PersistenceManager (*continued*)
 - examples of methods defined by 119
 - getFetchPlan() 186
 - getObjectById() 119
 - kodo.QueryCache property 193
 - makePersistent() 119
 - managing with the
 - OpenPersistenceManagerInViewFilter 295
 - newNamedQuery() 119
 - newObjectIdInstance() 154
 - newQuery() 119
- PersistenceManager.flush(), flush() 500
- PersistenceManagerFactory 118
 - configuring for non-transactional reads 311
 - configuring the transaction type 474
 - setNonTransactionalRead() 301
 - setOptimistic() 474
- PersistenceManagerFactory-level cache 191
 - evicting objects 192
- PersistenceManagerFactoryUtils 260
- persistent domain model
 - with EJB 3 372, 385
 - with Hibernate 212–227
 - with JDO 159–173
 - testing 132–141
- persistent object
 - assigning a primary key 108
 - change tracking 109
 - creating 107
 - database identity 108
 - deleting 107
 - ensuring Java identity matches persistent identity 108
 - generating identifiers in Hibernate 201
 - generating identifiers in JDO 151
 - generating identifiers with EJB 3 364
 - identity 107
 - managing lifecycle 107
 - nontransactional access 301
 - testing 135
- PersistentClass 214
- pessimistic locking
 - benefits and drawbacks 460
 - in EJB 3 403
 - example scenario 459
 - Hibernate 480
 - with iBATIS SQL Maps 466
 - in JDO 476
 - limitations 492
 - in Oracle 459
 - overview 45, 458–461
 - when to use 461
- Pessimistic Offline Lock pattern
 - benefits and drawbacks 510
 - cleaning up locks 511
 - design decisions 511–519
 - determining the lock owner 513
 - example scenario 509
 - handling locking failures 519
 - how to manage the locks 513
 - impact on other use cases 529
 - in EJB 3 403
 - lock types 512
 - motivation 508
 - overview 47, 508–511
 - using a lock manager 514
 - using in a domain model 520–532
 - using object locks 518
 - when to use 511
 - working with the Optimistic Offline Lock pattern 531
- PessimisticLockingFailure 469
- phantoms 453
- place order request 72
- Place Order use case
 - analyzing 69
 - developing a domain model for 68–79
 - encapsulating with a POJO façade 264–267
 - implemented using the Exposed Domain Model 304
 - implementing using the Transaction Script pattern 325
 - one approach 54
 - optimizing Hibernate eager loading 235–240
 - optimizing JDO eager loading 184–191
 - optimizing performance with Hibernate
 - process-level caching 240
 - optimizing with Hibernate query caching 241
 - optimizing with JDO process-level caching 191
 - optimizing with JDO query caching 193
 - overview 54
 - user interface 265
- PLACED_ORDER table 147
- PLACED_ORDER_LINE_ITEM table 147
- PlaceOrderFacade
 - as an EJB 3 session bean 386
 - as an EJB 3 stateless session bean 363
 - as a POJO 245
 - Spring-managed transactions 257
 - updateDeliveryInfo() 245
 - updateRestaurant() 270
 - using EJB 3 dependency injection 390

- placeOrderFacade-generic-beans.xml 281
- placeOrderFacade-hibernate-beans.xml 284
- PlaceOrderFacadeImpl 246
- PlaceOrderFacadeImplUsingSpring 399
- placeOrderFacade-jdo-beans.xml 282
- PlaceOrderFacadeMockTests 268
- PlaceOrderFacadeResult 266
- PlaceOrderFacadeResultFactory 273
 - EJB 3 implementation 387
- PlaceOrderService 65
 - as an EJB 3 session bean 391
 - implementing 80–86
 - Spring beans for the Exposed Domain Model pattern 310
 - testing 81
 - updateDeliveryInfo() 73
- placeOrderService-exposedDomain-beans.xml 310
- PlaceOrderServiceImpl 83
- PlaceOrderServiceResult 74
- PlaceOrderServiceTests 84
- PlaceOrderTransactionScripts 320, 327
 - Spring bean definitions 355
- placeOrderTransactionScripts-ibatis-beans.xml 356
- PlaceOrderTransactionScriptsImpl 333
- PlaceOrderTransactionScriptsImplTests 330
- plain old Java object. *See* POJO
- PlatformTransactionManager 259
- POJO
 - approach 13, 32
 - benefits of 15
 - defined 12
 - developing with 12, 29
 - EJB 3 enterprise Java Bean 362
 - making transactional 19–25
 - overview 15
 - persisting 16–18
 - See also* lightweight framework
- POJO façade
 - benefits 247–248
 - compared to an EJB 244–250
 - compared to the Exposed Domain Model pattern 247
 - converting to an EJB 3 session bean 386
 - deploying with Hibernate and Spring 284
 - deploying with JDO and Spring 282
 - deploying with Spring 279–286
 - design decisions 251
 - designing the interface 264–267
 - detached objects 254
 - determining method signatures 264
 - drawbacks 248, 250
 - encapsulation issues 250
 - example of 245–247
 - handling remote clients 263
 - implementing 267–271
 - implementing with Spring 257
 - managing connections 257
 - managing transactions 257
 - overview 244–250
 - reasons for using 271
 - signalling errors 256
 - similarity to transaction scripts 327
 - testing 267
 - use 39, 250
- portability
 - improving by using POJOs 16
 - improving with an object/relational mapping framework 115
- @PostConstruct 400
- presentation tier
 - and the Exposed Domain Model pattern 291
 - domain object methods that cannot be called by 250
 - example design 306
 - invoking POJO façade 264
 - managing transactions 296–311
- primary-key attribute 152
- procedural design. *See* Transaction Script pattern
- process-level caching
 - challenges with using 122
 - in Hibernate 240
 - in JDO 191
 - and optimistic locking 123
 - overview 122
 - using to improve performance 145
 - when to use 122
- productivity, improving with an object/relational mapping framework 114
- projection query
 - alternative to JDO fetch groups 433
 - in Hibernate 434
 - in JDO 433
- projectionList() 434
- Projections 434
 - setProjectionList() 434
- PROPAGATION_REQUIRED 259
- PROPAGATION_SUPPORTS 259
- <property> element 197

proxy 21
 applying with a
 BeanNameAutoProxyCreator 23
 in Hibernate 209
 trouble with downcasting in Hibernate 210
proxy() 79
proxyInterface property 397

Q

Query 118
 execute() 120
 getResultList() 367
 in EJB 3 367
 list() 120
 scroll() 120
 setLockMode() 480
 setParameter() 367
 setResultClass() 448
query, testing 136
query caching
 improving performance with 145
 overview 123
<query> element
 in Hibernate 232
 in JDO 180
query generation
 with Hibernate 427
 Hibernate example 444
 with IBATIS SQL Maps 413
 with JDO 426
 JDO example 439
 with object/relational mapping frameworks 426
 options 413–414
queryForList() 341
 dynamic paged queries 420
queryForObject() 341
querying caching
 in Hibernate 241
 in JDO 193

R

read/write locks 513
readOnly 259
refactoring 75
Reference 393
relationship. *See* mapping
releaseLock() 514

remote client
 supporting with a POJO façade 263
repeatable read
 drawbacks 454
 isolation level 453
repeatable read transactions
 benefits 454
 iBATIS SQL Maps 466
 in Hibernate 481
 in JDO 477
 when to use 454
repository
 accessing from an entity 88
 accessing via static methods and variables 89
 accessing via ThreadLocal 89
 defined 67
 dependency injection 89
 as an EJB 3 session bean 392
 eliminating boilerplate code 128
 encapsulating the persistence framework 17
 example of a Hibernate repository 231
 example of a JDO repository 178
 example of. *See* PendingOrderRepository; RestaurantRepository
 Hibernate example 443
 implementing with EJB 3 380–382
 implementing with Hibernate 228–234
 implementing with Hibernate and Spring 125
 implementing with JDO 173–183
 implementing with JDO and Spring 125
 implementing with Spring 125–132
 improving testability 129
 injecting the EntityManager 397
 JDO example 439
 mock object testing 140
 mocking 84
 overview 17
 passing as method parameter 89
 writing mock object tests for a JDO
 repository 174
request
 defining a service method for 73
 identifying 71
 See also checkout request; enter delivery info request; enter payment information; place order request; select restaurant request; update quantities request
@Resource 396
responsibility 69
 of an application 72
 assigning to classes and methods 73

- responsibility (*continued*)
 - determining 70
 - example of assigning 82
 - Restaurant 65
 - implementing with EJB 3 376
 - Restaurant List page 265
 - RESTAURANT table 147
 - RESTAURANT_TIME_RANGE table 147
 - RESTAURANT_ZIP_CODE table 146
 - RestaurantMother 233
 - RestaurantNotificationGateway 464
 - RestaurantNotificationTransactionScripts 463
 - RestaurantRepository 66
 - EJB 3 implementation 380
 - Hibernate implementation 228–234
 - isRestaurantAvailable() 88
 - JDO implementation 173–183
 - mocking 92
 - result factory
 - implementing 272–279
 - motivation 268
 - <resultMap> 342
 - returnValue() 79
 - RMIServiceExporter 263
 - rollback rule 24, 259
 - rollback(), defined by
 - PlatformTransactionManager 259
 - ROWNUM 416
 - example of using 422
- S**
-
- save()
 - defined by HibernatePersistenceTests 216
 - defined by Session 119
 - scroll()
 - defined by Criteria 437
 - defined by Hibernate Query 120
 - ScrollableResults 437
 - security
 - and POJOs 261
 - business tier 262
 - implementing with ACEGI security 249
 - POJO façade 248, 261
 - web tier 262
 - SecurityContextHolder 262
 - getting user id 513
 - SELECT FOR UPDATE 459
 - example 466
 - select restaurant request 72
 - Send Orders to Restaurant use case
 - domain model 472
 - impact of the Pessimistic Offline Lock pattern 529
 - implementing with iBATIS SQL Maps 462
 - one approach 56
 - optimistic locking 464
 - overview 56
 - pessimistic locking 466
 - Send Orders use case
 - implementing with Hibernate and JDO 472–483
 - separation of concerns
 - lack of 10
 - serializable transactions
 - benefits 454
 - drawbacks 454
 - in EJB 3 403
 - in Hibernate 481
 - iBATIS SQL Maps 466
 - in JDO 477
 - limitations 492
 - when to use 454
 - @Service 395–396
 - service 68
 - example test case 85
 - implementing 80–86
 - testing 81
 - See also* PlaceOrderService
 - service POJO 395
 - Servlet API, and transaction rollbacks 298
 - servlet filter
 - managing connections with 291
 - See also* OpenPersistenceManagerInViewFilter; OpenSessionInViewFilter
 - Session 118
 - beginTransaction() 119
 - clear() 303
 - close() 119
 - createCriteria() 427
 - createNamedQuery() 119
 - createQuery() 119
 - createSQLQuery() 448
 - flush() 479
 - getNamedQuery() 448
 - load() 119
 - load() and pessimistic locking 480
 - lock() 480
 - managing with OpenSessionInViewFilter 295

- Session (*continued*)
 - methods defined by examples 119
 - save() 119
- session bean, encapsulating the business logic
 - with 385–388
- session façade. *See* EJB Session Façade
- SessionFactory 118
- SessionFactoryUtils 126, 260
 - using directly 126
- setFetchGroup() 432
- setFetchMode() 430
- setFirstResult() 437
- setLockMode() 480
- setMaxResults() 437
- setNontransactionalRead() 301
- setOptimistic() 474
- setParameter() 367
- setQueryCacheEnabled() 193
- setResultClass() 448
- setter injection 26
- shared data, concurrent access to 452
- single field identity 152
- Spring
 - bean factory 21
 - connection management with a servlet filter 295
 - data access exceptions 469
 - deploying a POJO façade 279–286
 - extending `SQLException` mapping 470
 - and iBATIS SQL maps 339
 - implementing a POJO façade 257
 - implementing a repository with Hibernate and Spring 231
 - implementing a repository with JDO and Spring 178
 - implementing DAOs 337–354
 - implementing repositories 125–132
 - integrating dependency injection with EJB 3 392–398
 - `JndiObjectFactoryBean` 397
 - managing connections in the Transaction Script pattern 354–358
 - managing JDBC connections and transactions 354
 - managing transactions 21–25
 - managing transactions in the Transaction Script pattern 354–358
 - supporting remote clients 263
 - using dependency injection with EJB 3 398
 - using the ORM classes 126
- Spring AOP
 - benefits 25
 - compared to other AOP implementations 25
 - configuring JDO fetch groups with 187
 - example of a custom interceptor.
 - See* `TransactionRetryInterceptor`
 - for transaction management 248
 - managing 257
 - recovering from current update failures 484
 - security 262
- Spring bean
 - accessing using EJB 3 dependency injection 392–398
 - configuring a Hibernate POJO façade 284
 - configuring a JDO POJO façade 282
 - configuring a POJO façade 280
 - configuring a transaction script 355
 - defined 21
 - exposing via JNDI 393
 - for configuring the Exposed Domain Model pattern 310
 - for transaction scripts 354–358
 - verbosity compared to EJB 3 286
- Spring configuration file
 - exposed domain model example 310
 - exposed JDO domain model example 311
 - Hibernate POJO façade example 284
 - JDO POJO façade example 282
 - POJO façade example 281
 - transaction script example 356
- Spring data access exception, mapping using `SessionFactoryUtil` 127
- Spring HTTP 263
- SpringObjectFactory 394
- SQL
 - lack of portability 323
 - optimizer hint 115
 - problems with hand-written SQL 41
 - reasons to use hand-written SQL 42, 115
 - and transaction scripts 322
 - using directly with transaction scripts 324
- SQL projection query 447
- SQL query
 - in Hibernate 446, 448–449
 - in JDO 446–449
 - optimization 414–418
 - rewriting to improve performance 418
- `SQLExceptionCodesSQLExceptionTranslator` 471
- subclassing 471
- `SQLExceptionTranslator` 471

- <sqlMap> 358
 - <sqlmap> 342
 - SqlMapClientDaoSupport 339
 - SqlMapClientFactoryBean 356
 - SqlMapClientTemplate 339
 - configuration example 356
 - configuring to use a custom
 - SQLExceptionTranslator 471
 - DAO example 345
 - exception mapping 470
 - queryForList() 420
 - SqlMapClientTemplateusing 340
 - <sqlMapConfig> 358
 - sqlMap-config.xml 358
 - StaleObjectStateException 482
 - StaleStateException 482
 - @Stateless 363
 - <statement> 342
 - static methods and variables, drawbacks 89
 - stored procedures and object/relational mapping
 - frameworks 116
 - strategy attribute 476
 - <subclass> element 208
 - surrogate key 108
 - Hibernate 201
 - in JDO 151
 - SwarmCache 240
- T**
-
- table 146
 - per class 105
 - per concrete class 106
 - per hierarchy 104
 - Table Module pattern, overview 37
 - targetEntity member 379
 - test-driven development 75
 - of a Hibernate domain model 212–227
 - of a JDO domain model 159–173
 - and refactoring 75
 - testing
 - a DAO method 343
 - with a database 135
 - DbUnit 136
 - with DbUnit 351
 - EJB 3 domain model 382–385
 - an entity 91
 - Hibernate database schema 215
 - Hibernate object/relational mapping 214
 - Hibernate objects 213, 217–224
 - Hibernate persistent objects 216
 - iBatis SQL Maps 350
 - JDO database schema 161
 - JDO example of verifying the database
 - schema 166, 219
 - JDO object/relational mapping 160
 - JDO objects 159–170
 - with jMock 78
 - with JUnit 76
 - minimize test execution time 133
 - with mock objects 76
 - object/relational mapping 139
 - object/relational mapping metadata 139
 - object/relational mapping via XML 139
 - persistent object 135
 - a POJO façade 267
 - queries 136
 - a service 84
 - testing an HQL query 233
 - testing an JDOQL query 180
 - testing persistent objects example 166, 219
 - testing the object/relational mapping 133
 - a transaction script 329
 - using an in-memory database 138
 - verifying the Hibernate object/relational
 - mapping example 217
 - verifying the JDO object/
 - relational mapping example 164
 - with DbUnit 351
 - writing mock object tests for a JDO
 - repository 174
 - with XmlUnit 139
 - testing. *See* persistent domain model
 - The ServerSide Java Symposium 2004 244
 - ThreadLocal
 - internal to Spring 260
 - providing access to repositories with 89
 - TimeRange 66
 - <timestamp> 479
 - timestamp column 455
 - TiVo moment
 - changing the TV viewing experience 4
 - with POJOs and lightweight frameworks 16
 - Tomcat 27
 - TORPEDO benchmark 124
 - Transaction 118, 120
 - setNonTransactionalRead() 301
 - transaction
 - isolation levels 453–454
 - retrying 484
 - rollbacks and the Servlet API 298
 - serializable 453

- transaction management
 - and JDBC. *See* DataSourceTransactionManager
 - and the Servlet API 298
 - with an EJB 3 session bean 385–388
 - with HibernateTransactionManager 260
 - in a POJO façade 257
 - in the Exposed Domain Model pattern 296–311
 - JDBC transactions and Spring 354
 - with JdoTransactionManager 260
 - JTA transactions 261
 - with JtaTransactionManager 261
 - local transactions 260
 - retrying with the
 - TransactionRetryInterceptor 302
 - with TransactionInterceptor 257
 - using a servlet filter 297
- transaction script
 - defined 319
 - development steps 325
 - example 333
 - identifying 325–328
 - implementing 329–337
 - lack of portability 323
 - simplifying with a DAO 320
 - and SQL 322
 - testing 329
 - writing Spring beans 354–358
- Transaction Script pattern 35
 - benefits 322–323
 - drawbacks 322–323
 - EJB 3 401
 - example structure 36
 - overview 318–324
 - testing transaction scripts 329
 - when to use 324
- transaction script. *See* Transaction Script pattern
- transactionAttributesSource property 258
- TransactionInterceptor
 - configuration example 23
 - configuring 257
 - configuring the isolation level 467
 - how it works 23
 - managing transactions for a POJO façade 246
 - transaction script example 356
 - using with the Exposed Domain Model pattern 293
- TransactionRetryInterceptor 302, 484
- transcript script, similarity to a POJO façade 327
- TransferFacade 19
- TransferFacadeImpl 19
- TransferResult 7, 19
- TransferService 14

- TransferService EJB 6
- transparent persistence 16, 63, 110
- turtles all the way down 337

U

- unforeseen effects 318
- update quantities request 72
- update() 341, 498
- updateDeliveryInfo()
 - defined by PlaceOrderFacade 245
 - defined by
 - PlaceOrderTransactionScriptsImpl 333
 - of the PendingOrder class 87
 - of the PlaceOrderService class 73
- UpdateDeliveryInfoResult, returned by
 - PlaceOrderTransactionScripts 327
- UpdateDeliveryInfoServlet 292, 306
- updateRestaurant(), defined by
 - PlaceOrderFacade 270
- use case
 - to find domain classes 69
 - to identify requests 71
 - to identify transaction scripts 325
- user interface
 - to identify POJO façade methods 264
 - to identify transaction scripts 326

V

- value object, defined 67
 - See also* Address; TimeRange
- value-strategy attribute 152
- verify() 79
- verifyLock() 514
- <version> 476, 479
 - strategy attribute 476
- version column 455
- View Orders use case 409
 - eager loading 428
 - Hibernate implementation 442–446
 - iBatis SQL implementation 418–424
 - JDO implementation 438–442
 - one approach 55
 - overview 55

W

- web application
 - configuring in a Hibernate application 314
 - configuring in a JDO application 312

Web services 263
web.xml
 configuring for an exposed Hibernate domain
 model 314
 configuring for an exposed JDO domain
 model 312
WebApplicationContext
 configuring in a Hibernate web application 314
 configuring in a JDO web application 311
WebLogic Express 27
WebLogic Server, when to use 28
whole-part relationship 102

X

XML deployment descriptor
 replaced by annotations 363
XMLTestCase 139
 assertXPathEvaluatesTo() 139
XmlUnit 139
XPath 139

Z

ZIP_CODE table 146

POJOs IN ACTION Chris Richardson

Developing Enterprise Applications with Lightweight Frameworks

There is agreement in the Java community that EJBs often introduce more problems than they solve. Now there is a major trend toward lightweight technologies such as Hibernate, Spring, JDO, iBATIS, and others, all of which allow the developer to work directly with the simpler Plain Old Java Objects, or POJOs. Bowing to the new consensus, EJB 3 now also works with POJOs.

POJOs in Action describes these new, simpler, and faster ways to develop enterprise Java applications. It shows you how to go about making key design decisions, including how to organize and encapsulate the domain logic, access the database, manage transactions, and handle database concurrency.

Written for developers and designers, this is a new-generation Java applications guide. It helps you build lightweight applications that are easier to build, test, and maintain. The book is uniquely practical with design alternatives illustrated through numerous code examples.

What's Inside

- Leverage the frameworks' strengths, avoid their weaknesses
- Apply enterprise patterns in the lightweight world
- New patterns like POJO Façade and Exposed Domain Model
- Build rich domain models
- How Aspects improve design
- Lightweight testing strategies
- How to be agile

Chris Richardson is a developer and architect with over 20 years of experience. His consulting company specializes in jumpstarting projects and mentoring teams. Chris has been a technical leader at Insignia, BEA, and elsewhere. He has a computer science degree from the University of Cambridge in England and lives in Oakland, CA.

"A good way to quickly get up to speed with today's practices for lightweight development."

—Floyd Marinescu
Founder, InfoQ.com
Creator, TheServerSide.com

"Brings back simplicity to enterprise Java applications."

—Jonas Bonér
Senior Software Architect
Terracotta, Inc.

"A valuable guide for lightweight development."

—Craig Walls
Author, *Spring in Action*

"The author definitely knows what he is talking about."

—Oliver Zeigermann
J2EE Architect and
Apache committer

"Extremely valuable, plenty of sample code... I enthusiastically recommend it!"

—Brendan Murray
Senior Software Architect
IBM



Ask the Author



Ebook edition

www.manning.com/crichardson



9 781932 394580

ISBN 1-932394-58-3



MANNING

\$44.95 US/\$60.95 Canada