The Java EE 7 Tutorial

Volume 2

Fifth Edition

Eric Jendrock, Ricardo Cervera-Navarro, Ian Evans,
Kim Haase, William Markito

The Java EE 7 Tutorial
Volume 2

Fifth Edition

This page intentionally left blank

The Java EE 7 Tutorial

Volume 2

Fifth Edition

Eric Jendrock
Ricardo Cervera-Navarro
Ian Evans
Kim Haase
William Markito

Addison-Wesley

Upper Saddle River, NJ ● Boston ● Indianapolis ● San Francisco
New York ● Toronto ● Montreal ● London ● Munich ● Paris ● Madrid
Capetown ● Sydney ● Tokyo ● Singapore ● Mexico City

Visit us on the Web: informit.com/aw

Library of Congress control Number: 2014933972

ISBN-13: 978-0-321-98008-3
ISBN-10: 0-321-98008-5
Text printed in the United States on recycled paper at Edwards Brothers Malloy in Ann Arbor, Michigan.
First printing, May 2014

Contents

Preface xxxi

Audience xxxii
Before You Read This Book xxxii
Related Documentation xxxii
The Oracle Accessibility Program xxxiii
Conventions xxxiii
Default Paths and File Names xxxiii
Acknowledgments xxxiv

Part I Introduction 1

1 Overview 3

1.1 Java EE 7 Platform Highlights 4
1.2 Java EE Application Model 5
1.3 Distributed Multitiered Applications 6
1.3.1 Security 7
1.3.2 Java EE Components 8
1.3.3 Java EE Clients 8
1.3.3.1 Web Clients 8
1.3.3.2 Application Clients 8
1.3.3.3 Applets 9
1.3.3.4 The JavaBeans Component Architecture 9
1.3.3.5 Java EE Server Communications 10
1.3.4 Web Components 10
1.3.5 Business Components 11

1.3.6 Enterprise Information System Tier ... 12
1.4 Java EE Containers ... 13
1.4.1 Container Services ... 13
1.4.2 Container Types ... 14
1.5 Web Services Support ... 15
1.5.1 XML ... 16
1.5.2 SOAP Transport Protocol ... 16
1.5.3 WSDL Standard Format ... 16
1.6 Java EE Application Assembly and Deployment ... 17
1.7 Development Roles ... 17
1.7.1 Java EE Product Provider ... 18
1.7.2 Tool Provider ... 18
1.7.3 Application Component Provider ... 18
1.7.3.1 Enterprise Bean Developer ... 18
1.7.3.2 Web Component Developer ... 18
1.7.3.3 Application Client Developer ... 19
1.7.4 Application Assembler ... 19
1.7.5 Application Deployer and Administrator ... 19
1.8 Java EE 7 APIs ... 20
1.8.1 Enterprise JavaBeans Technology ... 23
1.8.2 Java Servlet Technology ... 24
1.8.3 JavaServer Faces Technology ... 24
1.8.4 JavaServer Pages Technology ... 25
1.8.5 JavaServer Pages Standard Tag Library ... 25
1.8.6 Java Persistence API ... 26
1.8.7 Java Transaction API ... 26
1.8.8 Java API for RESTful Web Services ... 26
1.8.9 Managed Beans ... 26
1.8.10 Contexts and Dependency Injection for Java EE ... 27
1.8.11 Dependency Injection for Java ... 27
1.8.12 Bean Validation ... 27
1.8.13 Java Message Service API ... 27
1.8.14 Java EE Connector Architecture ... 28
1.8.15 JavaMail API ... 28
1.8.16 Java Authorization Contract for Containers ... 29
1.8.17 Java Authentication Service Provider Interface for Containers ... 29
1.8.18 Java API for WebSocket ... 29

1.8.19 Java API for JSON Processing 29
1.8.20 Concurrency Utilities for Java EE 30
1.8.21 Batch Applications for the Java Platform 30
1.9 Java EE 7 APIs in the Java Platform, Standard Edition 7 30
1.9.1 Java Database Connectivity API 30
1.9.2 Java Naming and Directory Interface API 31
1.9.3 JavaBeans Activation Framework 31
1.9.4 Java API for XML Processing 32
1.9.5 Java Architecture for XML Binding 32
1.9.6 Java API for XML Web Services 32
1.9.7 SOAP with Attachments API for Java 33
1.9.8 Java Authentication and Authorization Service 33
1.9.9 Common Annotations for the Java Platform 33
1.10 GlassFish Server Tools 33

2 Using the Tutorial Examples 35

2.1 Required Software 35
2.1.1 Java EE 7 Software Development Kit 36
2.1.1.1 SDK Installation Tips 36
2.1.2 Java Platform, Standard Edition 37
2.1.3 Java EE 7 Tutorial Component 37
2.1.3.1 To Obtain the Tutorial Component Using the Update Tool 37
2.1.4 NetBeans IDE 37
2.1.4.1 To Install NetBeans IDE without GlassFish Server 38
2.1.4.2 To Add GlassFish Server as a Server Using NetBeans IDE 38
2.1.5 Apache Maven 38
2.2 Starting and Stopping GlassFish Server 39
2.2.1 To Start GlassFish Server Using NetBeans IDE 39
2.2.2 To Stop GlassFish Server Using NetBeans IDE 39
2.2.3 To Start GlassFish Server Using the Command Line 39
2.2.4 To Stop GlassFish Server Using the Command Line 40
2.3 Starting the Administration Console 40
2.3.1 To Start the Administration Console Using NetBeans IDE 40
2.4 Starting and Stopping the Java DB Server 40
2.4.1 To Start the Database Server Using NetBeans IDE 41
2.5 Building the Examples 41
2.6 Tutorial Example Directory Structure 41

2.7 Java EE 7 Maven Archetypes in the Tutorial ... 42
2.7.1 Installing the Tutorial Archetypes ... 42
2.7.1.1 Installing the Tutorial Archetypes Using NetBeans IDE ... 43
2.7.1.2 Installing the Tutorial Archetypes Using Maven ... 43
2.8 Getting the Latest Updates to the Tutorial ... 43
2.8.1 To Update the Tutorial Using NetBeans IDE ... 43
2.8.2 To Update the Tutorial Using the Command Line ... 43
2.9 Debugging Java EE Applications ... 44
2.9.1 Using the Server Log ... 44
2.9.1.1 To Use the Administration Console Log Viewer ... 44
2.9.2 Using a Debugger ... 45
2.9.2.1 To Debug an Application Using a Debugger ... 45

Part II Enterprise Beans ... 47

3 Enterprise Beans ... 49

3.1 What Is an Enterprise Bean? ... 49
3.1.1 Benefits of Enterprise Beans ... 50
3.1.2 When to Use Enterprise Beans ... 50
3.1.3 Types of Enterprise Beans ... 51
3.2 What Is a Session Bean? ... 51
3.2.1 Types of Session Beans ... 51
3.2.1.1 Stateful Session Beans ... 51
3.2.1.2 Stateless Session Beans ... 52
3.2.1.3 Singleton Session Beans ... 52
3.2.2 When to Use Session Beans ... 53
3.3 What Is a Message-Driven Bean? ... 53
3.3.1 What Makes Message-Driven Beans Different from Session Beans? ... 54
3.3.2 When to Use Message-Driven Beans ... 55
3.4 Accessing Enterprise Beans ... 55
3.4.1 Using Enterprise Beans in Clients ... 56
3.4.1.1 Portable JNDI Syntax ... 56
3.4.2 Deciding on Remote or Local Access ... 57
3.4.3 Local Clients ... 58
3.4.3.1 Accessing Local Enterprise Beans Using the No-Interface View ... 59
3.4.3.2 Accessing Local Enterprise Beans That Implement Business Interfaces ... 60
3.4.4 Remote Clients ... 60

3.4.5 Web Service Clients 61
3.4.6 Method Parameters and Access 62
3.4.6.1 Isolation 62
3.4.6.2 Granularity of Accessed Data 62
3.5 The Contents of an Enterprise Bean 62
3.6 Naming Conventions for Enterprise Beans 63
3.7 The Lifecycles of Enterprise Beans 63
3.7.1 The Lifecycle of a Stateful Session Bean 63
3.7.2 The Lifecycle of a Stateless Session Bean 64
3.7.3 The Lifecycle of a Singleton Session Bean 65
3.7.4 The Lifecycle of a Message-Driven Bean 65
3.8 Further Information about Enterprise Beans 66

4 Getting Started with Enterprise Beans 67

4.1 Creating the Enterprise Bean 68
4.1.1 Coding the Enterprise Bean Class 68
4.1.2 Creating the converter Web Client 69
4.1.3 Running the converter Example 70
4.1.3.1 To Run the converter Example Using NetBeans IDE 70
4.1.3.2 To Run the converter Example Using Maven 70
4.2 Modifying the Java EE Application 71
4.2.1 To Modify a Class File 71

5 Running the Enterprise Bean Examples 73

5.1 The cart Example 73
5.1.1 The Business Interface 74
5.1.2 Session Bean Class 75
5.1.2.1 Lifecycle Callback Methods 77
5.1.2.2 Business Methods 77
5.1.3 The @Remove Method 78
5.1.4 Helper Classes 79
5.1.5 Running the cart Example 79
5.1.5.1 To Run the cart Example Using NetBeans IDE 79
5.1.5.2 To Run the cart Example Using Maven 80
5.2 A Singleton Session Bean Example: counter 81
5.2.1 Creating a Singleton Session Bean 81

5.2.1.1 Initializing Singleton Session Beans 81
5.2.1.2 Managing Concurrent Access in a Singleton Session Bean 82
5.2.1.3 Handling Errors in a Singleton Session Bean 86
5.2.2 The Architecture of the counter Example 86
5.2.3 Running the counter Example 88
5.2.3.1 To Run the counter Example Using NetBeans IDE 88
5.2.3.2 To Run the counter Example Using Maven 89
5.3 A Web Service Example: helloservice 89
5.3.1 The Web Service Endpoint Implementation Class 89
5.3.2 Stateless Session Bean Implementation Class 90
5.3.3 Running the helloservice Example 91
5.3.3.1 To Build, Package, and Deploy the helloservice Example Using NetBeans IDE 91
5.3.3.2 To Build, Package, and Deploy the helloservice Example Using Maven 91
5.3.3.3 To Test the Service without a Client 92
5.4 Using the Timer Service 92
5.4.1 Creating Calendar-Based Timer Expressions 93
5.4.1.1 Specifying Multiple Values in Calendar Expressions 94
5.4.2 Programmatic Timers 95
5.4.2.1 The @Timeout Method 96
5.4.2.2 Creating Programmatic Timers 96
5.4.3 Automatic Timers 97
5.4.4 Canceling and Saving Timers 98
5.4.5 Getting Timer Information 99
5.4.6 Transactions and Timers 99
5.4.7 The timersession Example 100
5.4.8 Running the timersession Example 102
5.4.8.1 To Run the timersession Example Using NetBeans IDE 103
5.4.8.2 To Build, Package, and Deploy the timersession Example Using Maven 103
5.4.8.3 To Run the Web Client 103
5.5 Handling Exceptions 104

6 Using the Embedded Enterprise Bean Container 105

6.1 Overview of the Embedded Enterprise Bean Container 105
6.2 Developing Embeddable Enterprise Bean Applications 106

6.2.1 Running Embedded Applications ... 106
6.2.2 Creating the Enterprise Bean Container ... 107
6.2.2.1 Explicitly Specifying Enterprise Bean Modules to Be Initialized ... 107
6.2.3 Looking Up Session Bean References ... 108
6.2.4 Shutting Down the Enterprise Bean Container ... 108
6.3 The standalone Example Application ... 109
6.3.1 To Run the standalone Example Application Using NetBeans IDE ... 110
6.3.2 To Run the standalone Example Application Using Maven ... 110

7 Using Asynchronous Method Invocation in Session Beans ... 113

7.1 Asynchronous Method Invocation ... 113
7.1.1 Creating an Asynchronous Business Method ... 114
7.1.2 Calling Asynchronous Methods from Enterprise Bean Clients ... 115
7.1.2.1 Retrieving the Final Result from an Asynchronous Method Invocation ... 115
7.1.2.2 Cancelling an Asynchronous Method Invocation ... 115
7.1.2.3 Checking the Status of an Asynchronous Method Invocation ... 116
7.2 The async Example Application ... 116
7.2.1 Architecture of the async-war Module ... 116
7.2.2 Running the async Example ... 118
7.2.2.1 To Run the async Example Application Using NetBeans IDE ... 118
7.2.2.2 To Run the async Example Application Using Maven ... 119

Part III Persistence ... 121

8 Introduction to the Java Persistence API ... 123

8.1 Entities ... 123
8.1.1 Requirements for Entity Classes ... 124
8.1.2 Persistent Fields and Properties in Entity Classes ... 124
8.1.2.1 Persistent Fields ... 125
8.1.2.2 Persistent Properties ... 125
8.1.2.3 Using Collections in Entity Fields and Properties ... 126
8.1.2.4 Validating Persistent Fields and Properties ... 128
8.1.3 Primary Keys in Entities ... 130
8.1.4 Multiplicity in Entity Relationships ... 132
8.1.5 Direction in Entity Relationships ... 133

8.1.5.1 Bidirectional Relationships ... 133
8.1.5.2 Unidirectional Relationships ... 133
8.1.5.3 Queries and Relationship Direction ... 134
8.1.5.4 Cascade Operations and Relationships ... 134
8.1.5.5 Orphan Removal in Relationships ... 135
8.1.6 Embeddable Classes in Entities ... 135
8.2 Entity Inheritance ... 136
8.2.1 Abstract Entities ... 136
8.2.2 Mapped Superclasses ... 137
8.2.3 Non-Entity Superclasses ... 138
8.2.4 Entity Inheritance Mapping Strategies ... 138
8.2.4.1 The Single Table per Class Hierarchy Strategy ... 138
8.2.4.2 The Table per Concrete Class Strategy ... 140
8.2.4.3 The Joined Subclass Strategy ... 140
8.3 Managing Entities ... 141
8.3.1 The EntityManager Interface ... 141
8.3.1.1 Container-Managed Entity Managers ... 141
8.3.1.2 Application-Managed Entity Managers ... 141
8.3.1.3 Finding Entities Using the EntityManager ... 143
8.3.1.4 Managing an Entity Instance's Lifecycle ... 143
8.3.1.5 Persisting Entity Instances ... 143
8.3.1.6 Removing Entity Instances ... 144
8.3.1.7 Synchronizing Entity Data to the Database ... 145
8.3.2 Persistence Units ... 145
8.4 Querying Entities ... 146
8.5 Database Schema Creation ... 147
8.5.1 Configuring an Application to Create or Drop Database Tables ... 148
8.5.2 Loading Data Using SQL Scripts ... 149
8.6 Further Information about Persistence ... 150

9 Running the Persistence Examples ... 151

9.1 The order Application ... 151
9.1.1 Entity Relationships in the order Application ... 152
9.1.1.1 Self-Referential Relationships ... 152
9.1.1.2 One-to-One Relationships ... 153
9.1.1.3 One-to-Many Relationship Mapped to Overlapping Primary and Foreign Keys ... 154

9.1.1.4 Unidirectional Relationships ... 155
9.1.2 Primary Keys in the order Application ... 155
9.1.2.1 Generated Primary Keys ... 155
9.1.2.2 Compound Primary Keys ... 156
9.1.3 Entity Mapped to More Than One Database Table ... 159
9.1.4 Cascade Operations in the order Application ... 160
9.1.5 BLOB and CLOB Database Types in the order Application ... 160
9.1.6 Temporal Types in the order Application ... 161
9.1.7 Managing the order Application's Entities ... 162
9.1.7.1 Creating Entities ... 162
9.1.7.2 Finding Entities ... 162
9.1.7.3 Setting Entity Relationships ... 163
9.1.7.4 Using Queries ... 163
9.1.7.5 Removing Entities ... 164
9.1.8 Running the order Example ... 164
9.1.8.1 To Run the order Example Using NetBeans IDE ... 164
9.1.8.2 To Run the order Example Using Maven ... 165
9.2 The roster Application ... 165
9.2.1 Relationships in the roster Application ... 166
9.2.1.1 The Many-To-Many Relationship in roster ... 167
9.2.2 Entity Inheritance in the roster Application ... 167
9.2.3 Criteria Queries in the roster Application ... 169
9.2.3.1 Metamodel Classes in the roster Application ... 169
9.2.3.2 Obtaining a CriteriaBuilder Instance in RequestBean ... 170
9.2.3.3 Creating Criteria Queries in RequestBean's Business Methods ... 170
9.2.4 Automatic Table Generation in the roster Application ... 171
9.2.5 Running the roster Example ... 172
9.2.5.1 To Run the roster Example Using NetBeans IDE ... 172
9.2.5.2 To Run the roster Example Using Maven ... 173
9.3 The address-book Application ... 174
9.3.1 Bean Validation Constraints in address-book ... 174
9.3.2 Specifying Error Messages for Constraints in address-book ... 175
9.3.3 Validating Contact Input from a JavaServer Faces Application ... 176
9.3.4 Running the address-book Example ... 176
9.3.4.1 To Run the address-book Example Using NetBeans IDE ... 177
9.3.4.2 To Run the address-book Example Using Maven ... 177

10 The Java Persistence Query Language 179
10.1 Query Language Terminology 180
10.2 Creating Queries Using the Java Persistence Query Language 180
10.2.1 Named Parameters in Queries 181
10.2.2 Positional Parameters in Queries 181
10.3 Simplified Query Language Syntax 182
10.3.1 Select Statements 182
10.3.2 Update and Delete Statements 183
10.4 Example Queries 183
10.4.1 Simple Queries 183
10.4.1.1 A Basic Select Query 183
10.4.1.2 Eliminating Duplicate Values 184
10.4.1.3 Using Named Parameters 184
10.4.2 Queries That Navigate to Related Entities 185
10.4.2.1 A Simple Query with Relationships 185
10.4.2.2 Navigating to Single-Valued Relationship Fields 185
10.4.2.3 Traversing Relationships with an Input Parameter 185
10.4.2.4 Traversing Multiple Relationships 186
10.4.2.5 Navigating According to Related Fields 186
10.4.3 Queries with Other Conditional Expressions 187
10.4.3.1 The LIKE Expression 187
10.4.3.2 The IS NULL Expression 187
10.4.3.3 The IS EMPTY Expression 187
10.4.3.4 The BETWEEN Expression 188
10.4.3.5 Comparison Operators 188
10.4.4 Bulk Updates and Deletes 188
10.4.4.1 Update Queries 189
10.4.4.2 Delete Queries 189
10.5 Full Query Language Syntax 189
10.5.1 BNF Symbols 189
10.5.2 BNF Grammar of the Java Persistence Query Language 190
10.5.3 FROM Clause 194
10.5.3.1 Identifiers 194
10.5.3.2 Identification Variables 197
10.5.3.3 Range Variable Declarations 198
10.5.3.4 Collection Member Declarations 198
10.5.3.5 Joins 199

10.5.4 Path Expressions ... 199
10.5.4.1 Examples of Path Expressions ... 200
10.5.4.2 Expression Types ... 200
10.5.4.3 Navigation ... 201
10.5.5 WHERE Clause ... 201
10.5.5.1 Literals ... 201
10.5.5.2 Input Parameters ... 202
10.5.5.3 Conditional Expressions ... 202
10.5.5.4 Operators and Their Precedence ... 203
10.5.5.5 BETWEEN Expressions ... 203
10.5.5.6 IN Expressions ... 204
10.5.5.7 LIKE Expressions ... 204
10.5.5.8 NULL Comparison Expressions ... 205
10.5.5.9 Empty Collection Comparison Expressions ... 205
10.5.5.10 Collection Member Expressions ... 205
10.5.5.11 Subqueries ... 206
10.5.5.12 Functional Expressions ... 207
10.5.5.13 Case Expressions ... 209
10.5.5.14 NULL Values ... 209
10.5.5.15 Equality Semantics ... 210
10.5.6 SELECT Clause ... 211
10.5.6.1 Return Types ... 211
10.5.6.2 The DISTINCT Keyword ... 213
10.5.6.3 Constructor Expressions ... 213
10.5.7 ORDER BY Clause ... 213
10.5.8 GROUP BY and HAVING Clauses ... 214

11 Using the Criteria API to Create Queries ... 215

11.1 Overview of the Criteria and Metamodel APIs ... 215
11.2 Using the Metamodel API to Model Entity Classes ... 217
11.2.1 Using Metamodel Classes ... 218
11.3 Using the Criteria API and Metamodel API to Create Basic Typesafe Queries ... 219
11.3.1 Creating a Criteria Query ... 219
11.3.2 Query Roots ... 220
11.3.3 Querying Relationships Using Joins ... 221
11.3.4 Path Navigation in Criteria Queries ... 221
11.3.5 Restricting Criteria Query Results ... 222

11.3.5.1 The Expression Interface Methods 222
11.3.5.2 Expression Methods in the CriteriaBuilder Interface 222
11.3.6 Managing Criteria Query Results 224
11.3.6.1 Ordering Results 224
11.3.6.2 Grouping Results 225
11.3.7 Executing Queries 226
11.3.7.1 Single-Valued Query Results 226
11.3.7.2 Collection-Valued Query Results 226

12 Creating and Using String-Based Criteria Queries 227

12.1 Overview of String-Based Criteria API Queries 227
12.2 Creating String-Based Queries 228
12.3 Executing String-Based Queries 229

13 Controlling Concurrent Access to Entity Data with Locking 231

13.1 Overview of Entity Locking and Concurrency 231
13.1.1 Using Optimistic Locking 232
13.2 Lock Modes 233
13.2.1 Setting the Lock Mode 234
13.2.2 Using Pessimistic Locking 235
13.2.2.1 Pessimistic Locking Timeouts 235

14 Creating Fetch Plans with Entity Graphs 237

14.1 Entity Graph Basics 238
14.1.1 The Default Entity Graph 238
14.1.2 Using Entity Graphs in Persistence Operations 238
14.1.2.1 Fetch Graphs 239
14.1.2.2 Load Graphs 239
14.2 Using Named Entity Graphs 240
14.2.1 Applying Named Entity Graph Annotations to Entity Classes 240
14.2.2 Obtaining EntityGraph Instances from Named Entity Graphs 241
14.3 Using Entity Graphs in Query Operations 241

15 Using a Second-Level Cache with Java Persistence API Applications 243

15.1 Overview of the Second-Level Cache 243
15.1.1 Controlling whether Entities May Be Cached 244
15.2 Specifying the Cache Mode Settings to Improve Performance 245
15.2.1 Setting the Cache Retrieval and Store Modes 246
15.2.1.1 Cache Retrieval Mode 246
15.2.1.2 Cache Store Mode 247
15.2.1.3 Setting the Cache Retrieval or Store Mode 247
15.2.2 Controlling the Second-Level Cache Programmatically 248
15.2.2.1 Checking whether an Entity's Data Is Cached 248
15.2.2.2 Removing an Entity from the Cache 248
15.2.2.3 Removing All Data from the Cache 249

Part IV Messaging 251

16 Java Message Service Concepts 253

16.1 Overview of the JMS API 253
16.1.1 What Is Messaging? 253
16.1.2 What Is the JMS API? 254
16.1.3 When Can You Use the JMS API? 255
16.1.4 How Does the JMS API Work with the Java EE Platform? 256
16.2 Basic JMS API Concepts 257
16.2.1 JMS API Architecture 257
16.2.2 Messaging Styles 258
16.2.2.1 Point-to-Point Messaging Style 258
16.2.2.2 Publish/Subscribe Messaging Style 259
16.2.3 Message Consumption 260
16.3 The JMS API Programming Model 260
16.3.1 JMS Administered Objects 262
16.3.1.1 JMS Connection Factories 262
16.3.1.2 JMS Destinations 263
16.3.2 Connections 264
16.3.3 Sessions 264
16.3.4 JMSContext Objects 264
16.3.5 JMS Message Producers 266

16.3.6 JMS Message Consumers ... 266
16.3.6.1 JMS Message Listeners ... 267
16.3.6.2 JMS Message Selectors ... 268
16.3.6.3 Consuming Messages from Topics ... 268
16.3.6.4 Creating Durable Subscriptions ... 269
16.3.6.5 Creating Shared Subscriptions ... 272
16.3.7 JMS Messages ... 273
16.3.7.1 Message Headers ... 273
16.3.7.2 Message Properties ... 274
16.3.7.3 Message Bodies ... 275
16.3.8 JMS Queue Browsers ... 277
16.3.9 JMS Exception Handling ... 277
16.4 Using Advanced JMS Features ... 278
16.4.1 Controlling Message Acknowledgment ... 279
16.4.2 Specifying Options for Sending Messages ... 281
16.4.2.1 Specifying Message Persistence ... 281
16.4.2.2 Setting Message Priority Levels ... 282
16.4.2.3 Allowing Messages to Expire ... 282
16.4.2.4 Specifying a Delivery Delay ... 283
16.4.2.5 Using JMSProducer Method Chaining ... 283
16.4.3 Creating Temporary Destinations ... 283
16.4.4 Using JMS Local Transactions ... 284
16.4.5 Sending Messages Asynchronously ... 287
16.5 Using the JMS API in Java EE Applications ... 287
16.5.1 Creating Resources for Java EE Applications ... 288
16.5.2 Using Resource Injection in Enterprise Bean or Web Components ... 289
16.5.2.1 Injecting a ConnectionFactory, Queue, or Topic ... 290
16.5.2.2 Injecting a JMSContext Object ... 290
16.5.3 Using Java EE Components to Produce and to Synchronously Receive Messages ... 291
16.5.3.1 Managing JMS Resources in Web and EJB Components ... 291
16.5.3.2 Managing Transactions in Session Beans ... 292
16.5.4 Using Message-Driven Beans to Receive Messages Asynchronously ... 292
16.5.5 Managing JTA Transactions ... 296
16.6 Further Information about JMS ... 298

17 Java Message Service Examples 299

17.1 Overview of the JMS Examples 300
17.2 Writing Simple JMS Applications 301
17.2.1 Starting the JMS Provider 302
17.2.2 Creating JMS Administered Objects 302
17.2.2.1 To Create Resources for the Simple Examples 302
17.2.3 Building All the Simple Examples 303
17.2.3.1 To Build All the Simple Examples Using NetBeans IDE 303
17.2.3.2 To Build All the Simple Examples Using Maven 304
17.2.4 Sending Messages 304
17.2.4.1 The Producer.java Client 304
17.2.4.2 To Run the Producer Client 306
17.2.5 Receiving Messages Synchronously 307
17.2.5.1 The SynchConsumer.java Client 307
17.2.5.2 To Run the SynchConsumer and Producer Clients 308
17.2.6 Using a Message Listener for Asynchronous Message Delivery 309
17.2.6.1 Writing the AsynchConsumer.java and TextListener.java Clients 310
17.2.6.2 To Run the AsynchConsumer and Producer Clients 311
17.2.7 Browsing Messages on a Queue 313
17.2.7.1 The MessageBrowser.java Client 313
17.2.7.2 To Run the QueueBrowser Client 314
17.2.8 Running Multiple Consumers on the Same Destination 316
17.2.9 Acknowledging Messages 317
17.2.9.1 To Run the ClientAckConsumer Client 318
17.3 Writing More Advanced JMS Applications 319
17.3.1 Using Durable Subscriptions 319
17.3.1.1 To Create Resources for the Durable Subscription Example 320
17.3.1.2 To Run the Durable Subscription Example 320
17.3.1.3 To Run the unsubscriber Example 321
17.3.2 Using Local Transactions 322
17.3.2.1 To Create Resources for the transactedexample Example 325
17.3.2.2 To Run the transactedexample Clients 326
17.4 Writing High Performance and Scalable JMS Applications 328
17.4.1 Using Shared Nondurable Subscriptions 328
17.4.1.1 Writing the Clients for the Shared Consumer Example 329
17.4.1.2 To Run the SharedConsumer and Producer Clients 329
17.4.2 Using Shared Durable Subscriptions 330

17.4.2.1 To Run the SharedDurableConsumer and Producer Clients 331
17.5 Sending and Receiving Messages Using a Simple Web Application..... 332
17.5.1 The websimplemessage Facelets Pages..... 333
17.5.2 The websimplemessage Managed Beans..... 333
17.5.3 Running the websimplemessage Example..... 335
17.5.3.1 Creating Resources for the websimplemessage Example..... 335
17.5.3.2 To Package and Deploy websimplemessage Using NetBeans IDE..... 335
17.5.3.3 To Package and Deploy websimplemessage Using Maven..... 335
17.5.3.4 To Run the websimplemessage Example..... 335
17.6 Receiving Messages Asynchronously Using a Message-Driven Bean..... 336
17.6.1 Overview of the simplemessage Example..... 336
17.6.2 The simplemessage Application Client..... 337
17.6.3 The simplemessage Message-Driven Bean Class..... 338
17.6.3.1 The onMessage Method..... 338
17.6.4 Running the simplemessage Example..... 339
17.6.4.1 Creating Resources for the simplemessage Example..... 339
17.6.4.2 To Run the simplemessage Example Using NetBeans IDE..... 339
17.6.4.3 To Run the simplemessage Example Using Maven..... 340
17.7 Sending Messages from a Session Bean to an MDB..... 341
17.7.1 Writing the Application Components for the clientsessionmdb Example..... 341
17.7.1.1 Coding the Application Client: MyAppClient.java..... 342
17.7.1.2 Coding the Publisher Session Bean..... 343
17.7.1.3 Coding the Message-Driven Bean: MessageBean.java..... 343
17.7.2 Running the clientsessionmdb Example..... 344
17.7.2.1 To Run clientsessionmdb Using NetBeans IDE..... 344
17.7.2.2 To Run clientsessionmdb Using Maven..... 345
17.8 Using an Entity to Join Messages from Two MDBs..... 346
17.8.1 Overview of the clientmdbentity Example Application..... 347
17.8.2 Writing the Application Components for the clientmdbentity Example..... 348
17.8.2.1 Coding the Application Client: HumanResourceClient.java..... 349
17.8.2.2 Coding the Message-Driven Beans for the clientmdbentity Example..... 349
17.8.2.3 Coding the Entity Class for the clientmdbentity Example..... 350
17.8.3 Running the clientmdbentity Example..... 351
17.8.3.1 To Run clientmdbentity Using NetBeans IDE..... 351
17.8.3.2 To Run clientmdbentity Using Maven..... 352
17.8.3.3 Viewing the Application Output..... 352
17.9 Using NetBeans IDE to Create JMS Resources..... 354

17.9.1 To Create JMS Resources Using NetBeans IDE 354
17.9.2 To Delete JMS Resources Using NetBeans IDE 355

Part V Security 357

18 Introduction to Security in the Java EE Platform 359

18.1 Overview of Java EE Security 360
18.1.1 A Simple Application Security Walkthrough 360
18.1.1.1 Step 1: Initial Request 361
18.1.1.2 Step 2: Initial Authentication 361
18.1.1.3 Step 3: URL Authorization 361
18.1.1.4 Step 4: Fulfilling the Original Request 362
18.1.1.5 Step 5: Invoking Enterprise Bean Business Methods 362
18.1.2 Features of a Security Mechanism 363
18.1.3 Characteristics of Application Security 364
18.2 Security Mechanisms 365
18.2.1 Java SE Security Mechanisms 365
18.2.2 Java EE Security Mechanisms 366
18.2.2.1 Application-Layer Security 366
18.2.2.2 Transport-Layer Security 367
18.2.2.3 Message-Layer Security 368
18.3 Securing Containers 369
18.3.1 Using Annotations to Specify Security Information 369
18.3.2 Using Deployment Descriptors for Declarative Security 369
18.3.3 Using Programmatic Security 370
18.4 Securing GlassFish Server 370
18.5 Working with Realms, Users, Groups, and Roles 371
18.5.1 What Are Realms, Users, Groups, and Roles? 372
18.5.1.1 What Is a Realm? 373
18.5.1.2 What Is a User? 373
18.5.1.3 What Is a Group? 374
18.5.1.4 What Is a Role? 374
18.5.1.5 Some Other Terminology 374
18.5.2 Managing Users and Groups in GlassFish Server 375
18.5.2.1 To Add Users to GlassFish Server 375
18.5.3 Setting Up Security Roles 376
18.5.4 Mapping Roles to Users and Groups 377

18.6 Establishing a Secure Connection Using SSL 379
18.6.1 Verifying and Configuring SSL Support 380
18.7 Further Information about Security 381

19 Getting Started Securing Web Applications 383

19.1 Overview of Web Application Security 384
19.2 Securing Web Applications 385
19.2.1 Specifying Security Constraints 385
19.2.1.1 Specifying a Web Resource Collection 386
19.2.1.2 Specifying an Authorization Constraint 387
19.2.1.3 Specifying a Secure Connection 387
19.2.1.4 Specifying Security Constraints for Resources 389
19.2.2 Specifying Authentication Mechanisms 389
19.2.2.1 HTTP Basic Authentication 390
19.2.2.2 Form-Based Authentication 391
19.2.2.3 Digest Authentication 393
19.2.3 Specifying an Authentication Mechanism in the Deployment Descriptor 393
19.2.4 Declaring Security Roles 394
19.3 Using Programmatic Security with Web Applications 395
19.3.1 Authenticating Users Programmatically 395
19.3.2 Checking Caller Identity Programmatically 397
19.3.3 Example Code for Programmatic Security 398
19.3.4 Declaring and Linking Role References 400
19.4 Examples: Securing Web Applications 401
19.4.1 To Set Up Your System for Running the Security Examples 401
19.4.2 The hello2-basicauth Example: Basic Authentication with a Servlet 402
19.4.2.1 Specifying Security for Basic Authentication Using Annotations 403
19.4.2.2 To Build, Package, and Deploy the hello2-basicauth Example Using NetBeans IDE 404
19.4.2.3 To Build, Package, and Deploy the hello2-basicauth Example Using Maven 404
19.4.2.4 To Run the hello2-basicauth Example 405
19.4.3 The hello1-formauth Example: Form-Based Authentication with a JavaServer Faces Application 405
19.4.3.1 Creating the Login Form and the Error Page 406
19.4.3.2 Specifying Security for the Form-Based Authentication Example 408

19.4.3.3 To Build, Package, and Deploy the hello1-formauth Example Using NetBeans IDE ... 408
19.4.3.4 To Build, Package, and Deploy the hello1-formauth Example Using Maven ... 409
19.4.3.5 To Run the hello1-formauth Example ... 409

20 Getting Started Securing Enterprise Applications ... 411

20.1 Basic Security Tasks for Enterprise Applications ... 411
20.2 Securing Enterprise Beans ... 412
20.2.1 Securing an Enterprise Bean Using Declarative Security ... 414
20.2.1.1 Specifying Authorized Users by Declaring Security Roles ... 415
20.2.1.2 Specifying an Authentication Mechanism and Secure Connection ... 418
20.2.2 Securing an Enterprise Bean Programmatically ... 418
20.2.2.1 Accessing an Enterprise Bean Caller's Security Context ... 418
20.2.3 Propagating a Security Identity (Run-As) ... 420
20.2.3.1 Configuring a Component's Propagated Security Identity ... 421
20.2.3.2 Trust between Containers ... 422
20.2.4 Deploying Secure Enterprise Beans ... 422
20.3 Examples: Securing Enterprise Beans ... 422
20.3.1 The cart-secure Example: Securing an Enterprise Bean with Declarative Security ... 422
20.3.1.1 Annotating the Bean ... 423
20.3.1.2 To Run the cart-secure Example Using NetBeans IDE ... 425
20.3.1.3 To Run the cart-secure Example Using Maven ... 426
20.3.2 The converter-secure Example: Securing an Enterprise Bean with Programmatic Security ... 427
20.3.2.1 Modifying ConverterBean ... 428
20.3.2.2 Modifying ConverterServlet ... 429
20.3.2.3 To Run the converter-secure Example Using NetBeans IDE ... 429
20.3.2.4 To Run the converter-secure Example Using Maven ... 430
20.3.2.5 To Run the converter-secure Example ... 430

21 Java EE Security: Advanced Topics ... 431

21.1 Working with Digital Certificates ... 431
21.1.1 Creating a Server Certificate ... 433
21.1.1.1 To Use keytool to Create a Server Certificate ... 433
21.1.2 Adding Users to the Certificate Realm ... 435

21.1.3 Using a Different Server Certificate with GlassFish Server ... 435
21.1.3.1 To Specify a Different Server Certificate ... 435
21.2 Authentication Mechanisms ... 436
21.2.1 Client Authentication ... 436
21.2.2 Mutual Authentication ... 437
21.2.2.1 Enabling Mutual Authentication over SSL ... 438
21.2.2.2 Creating a Client Certificate for Mutual Authentication ... 439
21.3 Using the JDBC Realm for User Authentication ... 441
21.3.1 To Configure a JDBC Authentication Realm ... 441
21.4 Securing HTTP Resources ... 443
21.5 Securing Application Clients ... 446
21.5.1 Using Login Modules ... 447
21.5.2 Using Programmatic Login ... 448
21.6 Securing Enterprise Information Systems Applications ... 448
21.6.1 Container-Managed Sign-On ... 448
21.6.2 Component-Managed Sign-On ... 449
21.6.3 Configuring Resource Adapter Security ... 449
21.6.4 Mapping an Application Principal to EIS Principals ... 451
21.7 Configuring Security Using Deployment Descriptors ... 451
21.7.1 Specifying Security for Basic Authentication in the Deployment Descriptor ... 452
21.7.2 Specifying Non-Default Principal-to-Role Mapping in the Deployment Descriptor ... 453
21.8 Further Information about Advanced Security Topics ... 453

Part VI Java EE Supporting Technologies ... 455

22 Transactions ... 457

22.1 Transactions in Java EE Applications ... 458
22.2 What Is a Transaction? ... 458
22.3 Container-Managed Transactions ... 459
22.3.1 Transaction Attributes ... 459
22.3.1.1 Required Attribute ... 460
22.3.1.2 RequiresNew Attribute ... 461
22.3.1.3 Mandatory Attribute ... 461
22.3.1.4 NotSupported Attribute ... 461
22.3.1.5 Supports Attribute ... 461

22.3.1.6 Never Attribute ... 462
22.3.1.7 Summary of Transaction Attributes ... 462
22.3.1.8 Setting Transaction Attributes ... 463
22.3.2 Rolling Back a Container-Managed Transaction ... 464
22.3.3 Synchronizing a Session Bean's Instance Variables ... 464
22.3.4 Methods Not Allowed in Container-Managed Transactions ... 465
22.4 Bean-Managed Transactions ... 465
22.4.1 JTA Transactions ... 466
22.4.2 Returning without Committing ... 466
22.4.3 Methods Not Allowed in Bean-Managed Transactions ... 466
22.5 Transaction Timeouts ... 467
22.5.1 To Set a Transaction Timeout ... 467
22.6 Updating Multiple Databases ... 467
22.7 Transactions in Web Components ... 468
22.8 Further Information about Transactions ... 469

23 Resource Adapters and Contracts ... 471

23.1 What Is a Resource Adapter? ... 471
23.1.1 Management Contracts ... 472
23.1.1.1 Lifecycle Management ... 472
23.1.1.2 Work Management Contract ... 473
23.1.2 Generic Work Context Contract ... 474
23.1.3 Outbound and Inbound Contracts ... 474
23.2 Metadata Annotations ... 475
23.3 Common Client Interface ... 477
23.4 Using Resource Adapters with Contexts and Dependency Injection for Java EE (CDI) ... 478
23.5 Further Information about Resource Adapters ... 479

24 The Resource Adapter Examples ... 481

24.1 The trading Example ... 481
24.1.1 Using the Outbound Resource Adapter ... 482
24.1.2 Implementing the Outbound Resource Adapter ... 485
24.1.3 Running the trading Example ... 486
24.1.3.1 To Run the trading Example Using NetBeans IDE ... 487
24.1.3.2 To Run the trading Example Using Maven ... 487

24.2 The traffic Example 488
24.2.1 Using the Inbound Resource Adapter 490
24.2.2 Implementing the Inbound Resource Adapter 491
24.2.3 Running the traffic Example 493
24.2.3.1 To Run the traffic Example Using NetBeans IDE 493
24.2.3.2 To Run the traffic Example Using Maven 494

25 Using Java EE Interceptors 497

25.1 Overview of Interceptors 497
25.1.1 Interceptor Classes 498
25.1.2 Interceptor Lifecycle 499
25.1.3 Interceptors and CDI 499
25.2 Using Interceptors 499
25.2.1 Intercepting Method Invocations 500
25.2.1.1 Using Multiple Method Interceptors 500
25.2.1.2 Accessing Target Method Parameters from an Interceptor Class 501
25.2.2 Intercepting Lifecycle Callback Events 502
25.2.2.1 Using AroundConstruct Interceptor Methods 502
25.2.2.2 Using Multiple Lifecycle Callback Interceptors 503
25.2.3 Intercepting Timeout Events 503
25.2.3.1 Using Multiple Timeout Interceptors 504
25.2.4 Binding Interceptors to Components 504
25.2.4.1 Declaring the Interceptor Bindings on an Interceptor Class 505
25.2.4.2 Binding a Component to an Interceptor 505
25.2.5 Ordering Interceptors 506
25.3 The interceptor Example Application 507
25.3.1 Running the interceptor Example 508
25.3.1.1 To Run the interceptor Example Using NetBeans IDE 509
25.3.1.2 To Run the interceptor Example Using Maven 509

26 Batch Processing 511

26.1 Introduction to Batch Processing 512
26.1.1 Steps in Batch Jobs 512
26.1.2 Parallel Processing 514
26.1.3 Status and Decision Elements 514
26.1.4 Batch Framework Functionality 515

26.2 Batch Processing in Java EE ... 516
26.2.1 The Batch Processing Framework ... 516
26.2.2 Creating Batch Applications ... 516
26.2.3 Elements of a Batch Job ... 517
26.2.4 Properties and Parameters ... 518
26.2.5 Job Instances and Job Executions ... 518
26.2.6 Batch and Exit Status ... 518
26.3 Simple Use Case ... 519
26.3.1 Chunk Step ... 520
26.3.2 Task Step ... 522
26.4 Using the Job Specification Language ... 523
26.4.1 The job Element ... 524
26.4.2 The step Element ... 524
26.4.2.1 The chunk Element ... 526
26.4.2.2 The batchlet Element ... 528
26.4.2.3 The partition Element ... 528
26.4.3 The flow Element ... 531
26.4.4 The split Element ... 532
26.4.5 The decision Element ... 532
26.5 Creating Batch Artifacts ... 533
26.5.1 Batch Artifact Interfaces ... 533
26.5.2 Dependency Injection in Batch Artifacts ... 536
26.5.3 Using the Context Objects from the Batch Runtime ... 538
26.6 Submitting Jobs to the Batch Runtime ... 539
26.6.1 Starting a Job ... 539
26.6.2 Checking the Status of a Job ... 540
26.6.3 Invoking the Batch Runtime in Your Application ... 540
26.7 Packaging Batch Applications ... 540
26.8 The webserverlog Example Application ... 541
26.8.1 Architecture of the webserverlog Example Application ... 541
26.8.1.1 The Job Definition File ... 542
26.8.1.2 The LogLine and LogFilteredLine Items ... 543
26.8.1.3 The Chunk Step Batch Artifacts ... 543
26.8.1.4 The Listener Batch Artifacts ... 545
26.8.1.5 The Task Step Batch Artifact ... 546
26.8.1.6 The JavaServer Faces Pages ... 546
26.8.1.7 The Managed Bean ... 547

26.8.2 Running the webserverlog Example Application 547
26.8.2.1 To Run the webserverlog Example Application Using NetBeans IDE ... 548
26.8.2.2 To Run the webserverlog Example Application Using Maven 548
26.9 The phonebilling Example Application 548
26.9.1 Architecture of the phonebilling Example Application 549
26.9.1.1 The Job Definition File 549
26.9.1.2 The CallRecord and PhoneBill Entities 550
26.9.1.3 The Call Records Chunk Step 552
26.9.1.4 The Phone Billing Chunk Step 553
26.9.1.5 The JavaServer Faces Pages 555
26.9.1.6 The Managed Bean 556
26.9.2 Running the phonebilling Example Application 556
26.9.2.1 To Run the phonebilling Example Application Using NetBeans IDE 557
26.9.2.2 To Run the phonebilling Example Application Using Maven 557
26.10 Further Information about Batch Processing 557

27 Concurrency Utilities for Java EE 559

27.1 Concurrency Basics 559
27.1.1 Threads and Processes 560
27.2 Main Components of the Concurrency Utilities 560
27.3 Concurrency and Transactions 561
27.4 Concurrency and Security 562
27.5 The jobs Concurrency Example 562
27.5.1 Running the jobs Example 563
27.5.1.1 To Configure GlassFish Server for the Basic Concurrency Example 563
27.5.1.2 To Build, Package, and Deploy the jobs Example Using NetBeans IDE 564
27.5.1.3 To Build, Package, and Deploy the jobs Example Using Maven 565
27.5.1.4 To Run the jobs Example and Submit Jobs with Low Priority 565
27.5.1.5 To Run the jobs Example and Submit Jobs with High Priority 566
27.6 The taskcreator Concurrency Example 567
27.6.1 Running the taskcreator Example 569
27.6.1.1 To Build, Package, and Deploy the taskcreator Example Using NetBeans IDE 569
27.6.1.2 To Build, Package, and Deploy the taskcreator Example Using Maven 569

27.6.1.3 To Run the taskcreator Example 570
27.7 Further Information about the Concurrency Utilities 570

Part VII Case Studies 571

28 Duke's Bookstore Case Study Example 573

28.1 Design and Architecture of Duke's Bookstore 573
28.2 The Duke's Bookstore Interface 575
28.2.1 The Book Java Persistence API Entity 575
28.2.2 Enterprise Beans Used in Duke's Bookstore 575
28.2.3 Facelets Pages and Managed Beans Used in Duke's Bookstore 576
28.2.4 Custom Components and Other Custom Objects Used in Duke's Bookstore 577
28.2.5 Properties Files Used in Duke's Bookstore 578
28.2.6 Deployment Descriptors Used in Duke's Bookstore 579
28.3 Running the Duke's Bookstore Case Study Application 580
28.3.1 To Build and Deploy Duke's Bookstore Using NetBeans IDE 580
28.3.2 To Build and Deploy Duke's Bookstore Using Maven 580
28.3.3 To Run Duke's Bookstore 581

29 Duke's Tutoring Case Study Example 583

29.1 Design and Architecture of Duke's Tutoring 583
29.2 Main Interface 585
29.2.1 Java Persistence API Entities Used in the Main Interface 586
29.2.2 Enterprise Beans Used in the Main Interface 587
29.2.3 WebSocket Endpoint Used in the Main Interface 587
29.2.4 Facelets Files Used in the Main Interface 587
29.2.5 Helper Classes Used in the Main Interface 588
29.2.6 Properties Files 589
29.2.7 Deployment Descriptors Used in Duke's Tutoring 589
29.3 Administration Interface 590
29.3.1 Enterprise Beans Used in the Administration Interface 590
29.3.2 Facelets Files Used in the Administration Interface 590
29.3.3 CDI Managed Beans Used in the Administration Interface 591
29.3.4 Helper Classes Used in the Administration Interface 591
29.4 Running the Duke's Tutoring Case Study Application 592

29.4.1 Running Duke's Tutoring ... 592
29.4.1.1 To Build and Deploy Duke's Tutoring Using NetBeans IDE ... 592
29.4.1.2 To Build and Deploy Duke's Tutoring Using Maven ... 593
29.4.1.3 Using Duke's Tutoring ... 593

30 Duke's Forest Case Study Example ... 595

30.1 Design and Architecture of Duke's Forest ... 596
30.1.1 The events Project ... 598
30.1.2 The entities Project ... 600
30.1.3 The dukes-payment Project ... 603
30.1.4 The dukes-resources Project ... 603
30.1.5 The Duke's Store Project ... 603
30.1.5.1 Enterprise Beans Used in Duke's Store ... 605
30.1.5.2 Facelets Files Used in the Main Interface of Duke's Store ... 605
30.1.5.3 Facelets Files Used in the Administration Interface of Duke's Store ... 606
30.1.5.4 Managed Beans Used in Duke's Store ... 607
30.1.5.5 Helper Classes Used in Duke's Store ... 607
30.1.5.6 Qualifiers Used in Duke's Store ... 608
30.1.5.7 Event Handlers Used in Duke's Store ... 608
30.1.5.8 Deployment Descriptors Used in Duke's Store ... 608
30.1.6 The Duke's Shipment Project ... 608
30.1.6.1 Enterprise Beans Used in Duke's Shipment ... 609
30.1.6.2 Facelets Files Used in Duke's Shipment ... 609
30.1.6.3 Managed Beans Used in Duke's Shipment ... 610
30.1.6.4 Helper Class Used in Duke's Shipment ... 610
30.1.6.5 Qualifier Used in Duke's Shipment ... 610
30.1.6.6 Deployment Descriptors Used in Duke's Shipment ... 610
30.2 Building and Deploying the Duke's Forest Case Study Application ... 610
30.2.1 To Build and Deploy the Duke's Forest Application Using NetBeans IDE ... 610
30.2.2 To Build and Deploy the Duke's Forest Application Using Maven ... 611
30.3 Running the Duke's Forest Application ... 611
30.3.1 To Register as a Duke's Store Customer ... 612
30.3.2 To Purchase Products ... 612
30.3.3 To Approve Shipment of a Product ... 613
30.3.4 To Create a New Product ... 613

Index ... 615

Preface

This tutorial is a guide to developing enterprise applications for the Java Platform, Enterprise Edition 7 (Java EE 7), using GlassFish Server Open Source Edition.

GlassFish Server Open Source Edition is the leading open-source and open-community platform for building and deploying next-generation applications and services. GlassFish Server Open Source Edition, developed by the GlassFish project open-source community at `https://glassfish.java.net/`, is the first compatible implementation of the Java EE 7 platform specification. This lightweight, flexible, and open-source application server enables organizations not only to leverage the new capabilities introduced within the Java EE 7 specification, but also to add to their existing capabilities through a faster and more streamlined development and deployment cycle. GlassFish Server Open Source Edition is hereafter referred to as GlassFish Server.

The following topics are addressed here:

- Audience
- Before You Read This Book
- Related Documentation
- The Oracle Accessibility Program
- Conventions
- Default Paths and File Names
- Acknowledgments

Audience

This tutorial is intended for programmers interested in developing and deploying Java EE 7 applications. It covers the technologies comprising the Java EE platform and describes how to develop Java EE components and deploy them on the Java EE Software Development Kit (SDK).

Before You Read This Book

Before proceeding with this book, you should be familiar with Volume 1 of this tutorial. Both volumes assume that you have a good knowledge of the Java programming language. A good way to get to that point is to work through the Java Tutorials (`http://docs.oracle.com/javase/tutorial/`).

Related Documentation

The Java EE 7 Tutorial, Volume 1 covers Java EE 7 technologies not included in this volume, including JavaServer Faces, Java Servlets, Bean Validation, Contexts and Dependency Injection for Java EE, and web services.

The GlassFish Server documentation set describes deployment planning and system installation. To obtain documentation for GlassFish Server Open Source Edition, go to `https://glassfish.java.net/docs/`.

The Java EE 7 API specification can be viewed at `http://docs.oracle.com/javaee/7/api/` and is also provided in the Java EE 7 SDK.

Additionally, the Java EE Specifications at `http://www.oracle.com/technetwork/java/javaee/tech/` might be useful.

For information about creating enterprise applications in the NetBeans Integrated Development Environment (IDE), see `https://netbeans.org/kb/`.

For information about the Java DB database for use with GlassFish Server, see `http://www.oracle.com/technetwork/java/javadb/overview/`.

The GlassFish Samples project is a collection of sample applications that demonstrate a broad range of Java EE technologies. The GlassFish Samples are bundled with the Java EE Software Development Kit (SDK) and are also available from the GlassFish Samples project page at `https://glassfish-samples.java.net/`.

The Oracle Accessibility Program

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at `http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc`.

Conventions

The following table describes the typographic conventions that are used in this book.

Convention	Meaning	Example
Boldface	Boldface type indicates graphical user interface elements associated with an action or terms defined in text.	From the **File** menu, choose **Open Project**. A **cache** is a copy that is stored locally.
`Monospace`	Monospace type indicates the names of files and directories, commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.	Edit your `.login` file. Use `ls -a` to list all files. `machine_name% you have mail.`
Italic	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.	Read Chapter 6 in the *User's Guide*. Do *not* save the file. The command to remove a file is `rm` *filename*.

Default Paths and File Names

The following table describes the default paths and file names that are used in this book.

Placeholder	Description	Default Value
as-install	Represents the base installation directory for GlassFish Server or the SDK of which GlassFish Server is a part.	Installations on the Solaris operating system, Linux operating system, and Mac operating system: *user's-home-directory*`/glassfish4/glassfish` Windows, all installations: *SystemDrive*`:\glassfish4\glassfish`

Placeholder	Description	Default Value
as-install-parent	Represents the parent of the base installation directory for GlassFish Server.	Installations on the Solaris operating system, Linux operating system, and Mac operating system: *user's-home-directory*`/glassfish4` Windows, all installations: *SystemDrive*`:\glassfish4`
tut-install	Represents the base installation directory for the *Java EE Tutorial* after you install GlassFish Server or the SDK and run the Update Tool.	*as-install-parent*`/docs/javaee-tutorial`
domain-dir	Represents the directory in which a domain's configuration is stored.	*as-install*`/domains/domain1`

Acknowledgments

The Java EE tutorial team would like to thank the Java EE specification leads: Linda DeMichiel, Bill Shannon, Emmanuel Bernard, Ed Burns, Shing Wai Chan, Kin-Man Chung, Danny Coward, Nigel Deakin, Rod Johnson, Roger Kitain, Jitendra Kotamraju, Anthony Lai, Bob Lee, Ron Monzillo, Rajiv Mordani, Pete Muir, Paul Parkinson, Santiago Pericas-Geertsen, Marek Potociar, Sivakumar Thyagarajan, Marina Vatkina, and Chris Vignola.

We would also like to thank the Java EE 7 SDK team, especially Snjezana Sevo-Zenzerovic, Adam Leftik, Michael Chen, and John Clingan.

The JavaServer Faces technology chapters benefited greatly from suggestions by Manfred Riem as well as by the spec leads.

We would like to thank our manager, Alan Sommerer, for his support and steadying influence.

We also thank Jordan Douglas and Dawn Tyler for developing and updating the illustrations. Edna Elle provided invaluable help with tools. Sheila Cepero helped smooth our path in many ways.

Finally, we would like to express our profound appreciation to Greg Doench, Elizabeth Ryan, Caroline Senay, and the production team at Addison-Wesley for graciously seeing our manuscript to publication.

This page intentionally left blank

3

Enterprise Beans

Enterprise beans are Java EE components that implement Enterprise JavaBeans (EJB) technology. Enterprise beans run in the EJB container, a runtime environment within GlassFish Server (see Section 1.4.2, "Container Types"). Although transparent to the application developer, the EJB container provides system-level services, such as transactions and security, to its enterprise beans. These services enable you to quickly build and deploy enterprise beans, which form the core of transactional Java EE applications.

The following topics are addressed here:

- What Is an Enterprise Bean?
- What Is a Session Bean?
- What Is a Message-Driven Bean?
- Accessing Enterprise Beans
- The Contents of an Enterprise Bean
- Naming Conventions for Enterprise Beans
- The Lifecycles of Enterprise Beans
- Further Information about Enterprise Beans

3.1 What Is an Enterprise Bean?

Written in the Java programming language, an enterprise bean is a server-side component that encapsulates the business logic of an application. The **business logic** is the code that fulfills the purpose of the application. In an inventory control application, for example, the enterprise beans might implement the

business logic in methods called `checkInventoryLevel` and `orderProduct`. By invoking these methods, clients can access the inventory services provided by the application.

3.1.1 Benefits of Enterprise Beans

For several reasons, enterprise beans simplify the development of large, distributed applications. First, because the EJB container provides system-level services to enterprise beans, the bean developer can concentrate on solving business problems. The EJB container, rather than the bean developer, is responsible for system-level services, such as transaction management and security authorization.

Second, because the beans rather than the clients contain the application's business logic, the client developer can focus on the presentation of the client. The client developer does not have to code the routines that implement business rules or access databases. As a result, the clients are thinner, a benefit that is particularly important for clients that run on small devices.

Third, because enterprise beans are portable components, the application assembler can build new applications from existing beans. Provided that they use the standard APIs, these applications can run on any compliant Java EE server.

3.1.2 When to Use Enterprise Beans

You should consider using enterprise beans if your application has any of the following requirements.

- The application must be scalable. To accommodate a growing number of users, you may need to distribute an application's components across multiple machines. Not only can the enterprise beans of an application run on different machines, but also their location will remain transparent to the clients.
- Transactions must ensure data integrity. Enterprise beans support transactions, the mechanisms that manage the concurrent access of shared objects.
- The application will have a variety of clients. With only a few lines of code, remote clients can easily locate enterprise beans. These clients can be thin, various, and numerous.

3.1.3 Types of Enterprise Beans

Table 3–1 summarizes the two types of enterprise beans. The following sections discuss each type in more detail.

Table 3–1 Enterprise Bean Types

Enterprise Bean Type	Purpose
Session	Performs a task for a client; optionally, may implement a web service
Message-driven	Acts as a listener for a particular messaging type, such as the Java Message Service API

3.2 What Is a Session Bean?

A **session bean** encapsulates business logic that can be invoked programmatically by a client over local, remote, or web service client views. To access an application that is deployed on the server, the client invokes the session bean's methods. The session bean performs work for its client, shielding it from complexity by executing business tasks inside the server.

A session bean is not persistent. (That is, its data is not saved to a database.)

For code samples, see Chapter 5, "Running the Enterprise Bean Examples."

3.2.1 Types of Session Beans

Session beans are of three types: stateful, stateless, and singleton.

3.2.1.1 Stateful Session Beans

The state of an object consists of the values of its instance variables. In a **stateful session bean**, the instance variables represent the state of a unique client/bean session. Because the client interacts ("talks") with its bean, this state is often called the **conversational state**.

As its name suggests, a session bean is similar to an interactive session. A session bean is not shared; it can have only one client, in the same way that an interactive session can have only one user. When the client terminates, its session bean appears to terminate and is no longer associated with the client.

The state is retained for the duration of the client/bean session. If the client removes the bean, the session ends and the state disappears. This transient nature of the state is not a problem, however, because when the conversation between the client and the bean ends, there is no need to retain the state.

3.2.1.2 Stateless Session Beans

A **stateless session bean** does not maintain a conversational state with the client. When a client invokes the methods of a stateless bean, the bean's instance variables may contain a state specific to that client but only for the duration of the invocation. When the method is finished, the client-specific state should not be retained. Clients may, however, change the state of instance variables in pooled stateless beans, and this state is held over to the next invocation of the pooled stateless bean. Except during method invocation, all instances of a stateless bean are equivalent, allowing the EJB container to assign an instance to any client. That is, the state of a stateless session bean should apply across all clients.

Because they can support multiple clients, stateless session beans can offer better scalability for applications that require large numbers of clients. Typically, an application requires fewer stateless session beans than stateful session beans to support the same number of clients.

A stateless session bean can implement a web service, but a stateful session bean cannot.

3.2.1.3 Singleton Session Beans

A **singleton session bean** is instantiated once per application and exists for the lifecycle of the application. Singleton session beans are designed for circumstances in which a single enterprise bean instance is shared across and concurrently accessed by clients.

Singleton session beans offer similar functionality to stateless session beans but differ from them in that there is only one singleton session bean per application, as opposed to a pool of stateless session beans, any of which may respond to a client request. Like stateless session beans, singleton session beans can implement web service endpoints.

Singleton session beans maintain their state between client invocations but are not required to maintain their state across server crashes or shutdowns.

Applications that use a singleton session bean may specify that the singleton should be instantiated upon application startup, which allows the singleton to perform initialization tasks for the application. The singleton may perform cleanup tasks on application shutdown as well, because the singleton will operate throughout the lifecycle of the application.

3.2.2 When to Use Session Beans

Stateful session beans are appropriate if any of the following conditions are true.

- The bean's state represents the interaction between the bean and a specific client.
- The bean needs to hold information about the client across method invocations.
- The bean mediates between the client and the other components of the application, presenting a simplified view to the client.
- Behind the scenes, the bean manages the work flow of several enterprise beans.

To improve performance, you might choose a stateless session bean if it has any of these traits.

- The bean's state has no data for a specific client.
- In a single method invocation, the bean performs a generic task for all clients. For example, you might use a stateless session bean to send an email that confirms an online order.
- The bean implements a web service.

Singleton session beans are appropriate in the following circumstances.

- State needs to be shared across the application.
- A single enterprise bean needs to be accessed by multiple threads concurrently.
- The application needs an enterprise bean to perform tasks upon application startup and shutdown.
- The bean implements a web service.

3.3 What Is a Message-Driven Bean?

A **message-driven bean** is an enterprise bean that allows Java EE applications to process messages asynchronously. This type of bean normally acts as a JMS message listener, which is similar to an event listener but receives JMS messages instead of events. The messages can be sent by any Java EE component (an application client, another enterprise bean, or a web component) or by a JMS application or system that does not use Java EE technology. Message-driven beans can process JMS messages or other kinds of messages.

3.3.1 What Makes Message-Driven Beans Different from Session Beans?

The most visible difference between message-driven beans and session beans is that clients do not access message-driven beans through interfaces. Interfaces are described in Section 3.4, "Accessing Enterprise Beans." Unlike a session bean, a message-driven bean has only a bean class.

In several respects, a message-driven bean resembles a stateless session bean.

- A message-driven bean's instances retain no data or conversational state for a specific client.
- All instances of a message-driven bean are equivalent, allowing the EJB container to assign a message to any message-driven bean instance. The container can pool these instances to allow streams of messages to be processed concurrently.
- A single message-driven bean can process messages from multiple clients.

The instance variables of the message-driven bean instance can contain some state across the handling of client messages, such as a JMS API connection, an open database connection, or an object reference to an enterprise bean object.

Client components do not locate message-driven beans and invoke methods directly on them. Instead, a client accesses a message-driven bean through, for example, JMS by sending messages to the message destination for which the message-driven bean class is the `MessageListener`. You assign a message-driven bean's destination during deployment by using GlassFish Server resources.

Message-driven beans have the following characteristics.

- They execute upon receipt of a single client message.
- They are invoked asynchronously.
- They are relatively short-lived.
- They do not represent directly shared data in the database, but they can access and update this data.
- They can be transaction-aware.
- They are stateless.

When a message arrives, the container calls the message-driven bean's `onMessage` method to process the message. The `onMessage` method normally casts the message to one of the five JMS message types and handles it in accordance with the application's business logic. The `onMessage` method can call helper methods or can invoke a session bean to process the information in the message or to store it in a database.

A message can be delivered to a message-driven bean within a transaction context, so all operations within the `onMessage` method are part of a single transaction. If message processing is rolled back, the message will be redelivered. For more information, see Section 17.6, "Receiving Messages Asynchronously Using a Message-Driven Bean," and Chapter 22, "Transactions."

3.3.2 When to Use Message-Driven Beans

Session beans allow you to send JMS messages and to receive them synchronously but not asynchronously. To avoid tying up server resources, do not to use blocking synchronous receives in a server-side component; in general, JMS messages should not be sent or received synchronously. To receive messages asynchronously, use a message-driven bean.

3.4 Accessing Enterprise Beans

Note: The material in this section applies only to session beans and not to message-driven beans. Because they have a different programming model, message-driven beans do not have interfaces or no-interface views that define client access.

Clients access enterprise beans either through a no-interface view or through a business interface. A **no-interface view** of an enterprise bean exposes the public methods of the enterprise bean implementation class to clients. Clients using the no-interface view of an enterprise bean may invoke any public methods in the enterprise bean implementation class or any superclasses of the implementation class. A **business interface** is a standard Java programming language interface that contains the business methods of the enterprise bean.

A client can access a session bean only through the methods defined in the bean's business interface or through the public methods of an enterprise bean that has a no-interface view. The business interface or no-interface view defines the client's view of an enterprise bean. All other aspects of the enterprise bean (method implementations and deployment settings) are hidden from the client.

Well-designed interfaces and no-interface views simplify the development and maintenance of Java EE applications. Not only do clean interfaces and no-interface views shield the clients from any complexities in the EJB tier, but they also allow the enterprise beans to change internally without affecting the clients. For example, if you change the implementation of a session bean business method, you won't have to alter the client code. But if you were to change the

method definitions in the interfaces, you might have to modify the client code as well. Therefore, it is important that you design the interfaces and no-interface views carefully to isolate your clients from possible changes in the enterprise beans.

Session beans can have more than one business interface. Session beans should, but are not required to, implement their business interface or interfaces.

3.4.1 Using Enterprise Beans in Clients

The client of an enterprise bean obtains a reference to an instance of an enterprise bean through either **dependency injection**, using Java programming language annotations, or **JNDI lookup**, using the Java Naming and Directory Interface syntax to find the enterprise bean instance.

Dependency injection is the simplest way of obtaining an enterprise bean reference. Clients that run within a Java EE server-managed environment, JavaServer Faces web applications, JAX-RS web services, other enterprise beans, or Java EE application clients support dependency injection using the `javax.ejb.EJB` annotation.

Applications that run outside a Java EE server-managed environment, such as Java SE applications, must perform an explicit lookup. JNDI supports a global syntax for identifying Java EE components to simplify this explicit lookup.

3.4.1.1 Portable JNDI Syntax

Three JNDI namespaces are used for portable JNDI lookups: `java:global`, `java:module`, and `java:app`.

- The `java:global` JNDI namespace is the portable way of finding remote enterprise beans using JNDI lookups. JNDI addresses are of the following form:

  `java:global[/`*`application-name`*`]/`*`module-name`*`/`
  `enterprise-bean-name``[/`*`interface-name`*`]`

  Application name and module name default to the name of the application and module minus the file extension. Application names are required only if the application is packaged within an EAR. The interface name is required only if the enterprise bean implements more than one business interface.

- The `java:module` namespace is used to look up local enterprise beans within the same module. JNDI addresses using the `java:module` namespace are of the following form:

  `java:module/`*`enterprise-bean-name`*`/[`*`interface-name`*`]`

The interface name is required only if the enterprise bean implements more than one business interface.

- The `java:app` namespace is used to look up local enterprise beans packaged within the same application. That is, the enterprise bean is packaged within an EAR file containing multiple Java EE modules. JNDI addresses using the `java:app` namespace are of the following form:

  `java:app[/`*`module-name`*`]/`*`enterprise-bean-name`*`[/`*`interface-name`*`]`

  The module name is optional. The interface name is required only if the enterprise bean implements more than one business interface.

For example, if an enterprise bean, `MyBean`, is packaged within the web application archive `myApp.war`, the module name is `myApp`. The portable JNDI name is `java:module/MyBean`. An equivalent JNDI name using the `java:global` namespace is `java:global/myApp/MyBean`.

3.4.2 Deciding on Remote or Local Access

When you design a Java EE application, one of the first decisions you make is the type of client access allowed by the enterprise beans: remote, local, or web service.

Whether to allow local or remote access depends on the following factors.

- **Tight or loose coupling of related beans**: Tightly coupled beans depend on one another. For example, if a session bean that processes sales orders calls a session bean that emails a confirmation message to the customer, these beans are tightly coupled. Tightly coupled beans are good candidates for local access. Because they fit together as a logical unit, they typically call each other often and would benefit from the increased performance that is possible with local access.
- **Type of client**: If an enterprise bean is accessed by application clients, it should allow remote access. In a production environment, these clients almost always run on machines other than those on which GlassFish Server is running. If an enterprise bean's clients are web components or other enterprise beans, the type of access depends on how you want to distribute your components.
- **Component distribution**: Java EE applications are scalable because their server-side components can be distributed across multiple machines. In a distributed application, for example, the server that the web components run on may not be the one on which the enterprise beans they access are deployed. In this distributed scenario, the enterprise beans should allow remote access.

- **Performance**: Owing to such factors as network latency, remote calls may be slower than local calls. On the other hand, if you distribute components among different servers, you may improve the application's overall performance. Both of these statements are generalizations; performance can vary in different operational environments. Nevertheless, you should keep in mind how your application design might affect performance.

If you aren't sure which type of access an enterprise bean should have, choose remote access. This decision gives you more flexibility. In the future, you can distribute your components to accommodate the growing demands on your application.

Although it is uncommon, it is possible for an enterprise bean to allow both remote and local access. If this is the case, either the business interface of the bean must be explicitly designated as a business interface by being decorated with the `@Remote` or `@Local` annotations, or the bean class must explicitly designate the business interfaces by using the `@Remote` and `@Local` annotations. The same business interface cannot be both a local and a remote business interface.

3.4.3 Local Clients

A local client has these characteristics.

- It must run in the same application as the enterprise bean it accesses.
- It can be a web component or another enterprise bean.
- To the local client, the location of the enterprise bean it accesses is not transparent.

The no-interface view of an enterprise bean is a local view. The public methods of the enterprise bean implementation class are exposed to local clients that access the no-interface view of the enterprise bean. Enterprise beans that use the no-interface view do not implement a business interface.

The **local business interface** defines the bean's business and lifecycle methods. If the bean's business interface is not decorated with `@Local` or `@Remote`, and if the bean class does not specify the interface using `@Local` or `@Remote`, the business interface is by default a local interface.

To build an enterprise bean that allows only local access, you may, but are not required to, do one of the following.

- Create an enterprise bean implementation class that does not implement a business interface, indicating that the bean exposes a no-interface view to clients. For example:

```
@Session
public class MyBean { ... }
```

- Annotate the business interface of the enterprise bean as a `@Local` interface. For example:

```
@Local
public interface InterfaceName { ... }
```

- Specify the interface by decorating the bean class with `@Local` and specify the interface name. For example:

```
@Local(InterfaceName.class)
public class BeanName implements InterfaceName  { ... }
```

3.4.3.1 Accessing Local Enterprise Beans Using the No-Interface View

Client access to an enterprise bean that exposes a local, no-interface view is accomplished through either dependency injection or JNDI lookup.

- To obtain a reference to the no-interface view of an enterprise bean through dependency injection, use the `javax.ejb.EJB` annotation and specify the enterprise bean's implementation class:

```
@EJB
ExampleBean exampleBean;
```

- To obtain a reference to the no-interface view of an enterprise bean through JNDI lookup, use the `javax.naming.InitialContext` interface's `lookup` method:

```
ExampleBean exampleBean = (ExampleBean)
        InitialContext.lookup("java:module/ExampleBean");
```

Clients *do not* use the `new` operator to obtain a new instance of an enterprise bean that uses a no-interface view.

3.4.3.2 Accessing Local Enterprise Beans That Implement Business Interfaces

Client access to enterprise beans that implement local business interfaces is accomplished through either dependency injection or JNDI lookup.

- To obtain a reference to the local business interface of an enterprise bean through dependency injection, use the `javax.ejb.EJB` annotation and specify the enterprise bean's local business interface name:

```
@EJB
Example example;
```

- To obtain a reference to a local business interface of an enterprise bean through JNDI lookup, use the `javax.naming.InitialContext` interface's `lookup` method:

```
ExampleLocal example = (ExampleLocal)
         InitialContext.lookup("java:module/ExampleLocal");
```

3.4.4 Remote Clients

A remote client of an enterprise bean has the following traits.

- It can run on a different machine and a different JVM from the enterprise bean it accesses. (It is not required to run on a different JVM.)
- It can be a web component, an application client, or another enterprise bean.
- To a remote client, the location of the enterprise bean is transparent.
- The enterprise bean must implement a business interface. That is, remote clients *may not* access an enterprise bean through a no-interface view.

To create an enterprise bean that allows remote access, you must either

- Decorate the business interface of the enterprise bean with the `@Remote` annotation:

```
@Remote
public interface InterfaceName { ... }
```

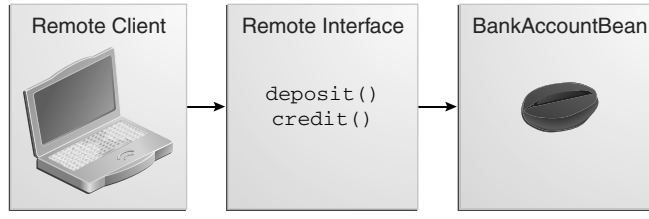
- Or decorate the bean class with `@Remote`, specifying the business interface or interfaces:

```
@Remote(InterfaceName.class)
public class BeanName implements InterfaceName { ... }
```

The **remote interface** defines the business and lifecycle methods that are specific to the bean. For example, the remote interface of a bean named `BankAccountBean`

might have business methods named `deposit` and `credit`. Figure 3–1 shows how the interface controls the client's view of an enterprise bean.

Figure 3–1 Interfaces for an Enterprise Bean with Remote Access



Client access to an enterprise bean that implements a remote business interface is accomplished through either dependency injection or JNDI lookup.

- To obtain a reference to the remote business interface of an enterprise bean through dependency injection, use the `javax.ejb.EJB` annotation and specify the enterprise bean's remote business interface name:

```
@EJB
Example example;
```

- To obtain a reference to a remote business interface of an enterprise bean through JNDI lookup, use the `javax.naming.InitialContext` interface's `lookup` method:

```
ExampleRemote example = (ExampleRemote)
        InitialContext.lookup("java:global/myApp/ExampleRemote");
```

3.4.5 Web Service Clients

A web service client can access a Java EE application in two ways. First, the client can access a web service created with JAX-WS. (For more information on JAX-WS, see Chapter 28, "Building Web Services with JAX-WS," in *The Java EE 7 Tutorial, Volume 1*.) Second, a web service client can invoke the business methods of a stateless session bean. Message beans cannot be accessed by web service clients.

Provided that it uses the correct protocols (SOAP, HTTP, WSDL), any web service client can access a stateless session bean, whether or not the client is written in the Java programming language. The client doesn't even "know" what technology implements the service: stateless session bean, JAX-WS, or some other technology. In addition, enterprise beans and web components can be clients of web services. This flexibility enables you to integrate Java EE applications with web services.

A web service client accesses a stateless session bean through the bean's web service endpoint implementation class. By default, all public methods in the bean class are accessible to web service clients. The `@WebMethod` annotation may be used to customize the behavior of web service methods. If the `@WebMethod` annotation is used to decorate the bean class's methods, only those methods decorated with `@WebMethod` are exposed to web service clients.

For a code example, see Section 5.3, "A Web Service Example: helloservice."

3.4.6 Method Parameters and Access

The type of access affects the parameters of the bean methods that are called by clients. The following sections apply not only to method parameters but also to method return values.

3.4.6.1 Isolation

The parameters of remote calls are more isolated than those of local calls. With remote calls, the client and the bean operate on different copies of a parameter object. If the client changes the value of the object, the value of the copy in the bean does not change. This layer of isolation can help protect the bean if the client accidentally modifies the data.

In a local call, both the client and the bean can modify the same parameter object. In general, you should not rely on this side effect of local calls. Perhaps someday you will want to distribute your components, replacing the local calls with remote ones.

As with remote clients, web service clients operate on different copies of parameters than does the bean that implements the web service.

3.4.6.2 Granularity of Accessed Data

Because remote calls are likely to be slower than local calls, the parameters in remote methods should be relatively coarse-grained. A coarse-grained object contains more data than a fine-grained one, so fewer access calls are required. For the same reason, the parameters of the methods called by web service clients should also be coarse-grained.

3.5 The Contents of an Enterprise Bean

To develop an enterprise bean, you must provide the following files.

- **Enterprise bean class**: Implements the business methods of the enterprise bean and any lifecycle callback methods.

- **Business interfaces**: Define the business methods implemented by the enterprise bean class. A business interface is not required if the enterprise bean exposes a local, no-interface view.
- **Helper classes**: Other classes needed by the enterprise bean class, such as exception and utility classes.

Package the programming artifacts in the preceding list either into an EJB JAR file (a stand-alone module that stores the enterprise bean) or within a web application archive (WAR) module. For more information, see Section 5.2.1, "Packaging Enterprise Beans in EJB JAR Modules," and Section 5.2.2, "Packaging Enterprise Beans in WAR Modules," both in *The Java EE 7 Tutorial, Volume 1*.

3.6 Naming Conventions for Enterprise Beans

Because enterprise beans are composed of multiple parts, it's useful to follow a naming convention for your applications. Table 3–2 summarizes the conventions for the example beans in this tutorial.

Table 3–2 Naming Conventions for Enterprise Beans

Item	Syntax	Example
Enterprise bean name	*name*`Bean`	`AccountBean`
Enterprise bean class	*name*`Bean`	`AccountBean`
Business interface	*name*	`Account`

3.7 The Lifecycles of Enterprise Beans

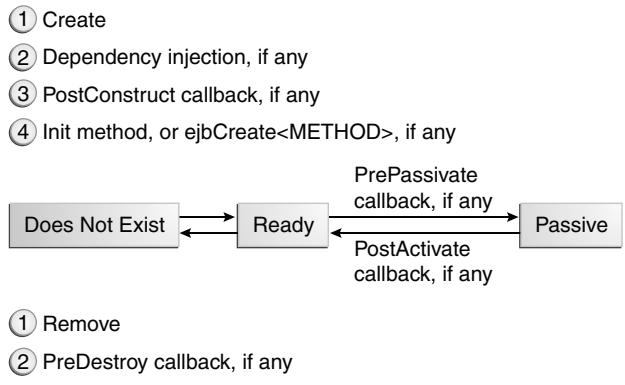
An enterprise bean goes through various stages during its lifetime, or lifecycle. Each type of enterprise bean (stateful session, stateless session, singleton session, or message-driven) has a different lifecycle.

The descriptions that follow refer to methods that are explained along with the code examples in the next two chapters. If you are new to enterprise beans, you should skip this section and run the code examples first.

3.7.1 The Lifecycle of a Stateful Session Bean

Figure 3–2 illustrates the stages that a stateful session bean passes through during its lifetime. The client initiates the lifecycle by obtaining a reference to a stateful session bean. The container performs any dependency injection and then invokes the method annotated with `@PostConstruct`, if any. The bean is now ready to have its business methods invoked by the client.

Figure 3–2 Lifecycle of a Stateful Session Bean



While in the ready stage, the EJB container may decide to deactivate, or **passivate**, the bean by moving it from memory to secondary storage. (Typically, the EJB container uses a least-recently-used algorithm to select a bean for passivation.) The EJB container invokes the method annotated `@PrePassivate`, if any, immediately before passivating it. If a client invokes a business method on the bean while it is in the passive stage, the EJB container activates the bean, calls the method annotated `@PostActivate`, if any, and then moves it to the ready stage.

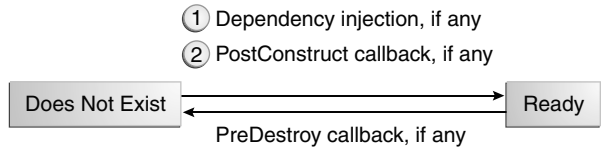
At the end of the lifecycle, the client invokes a method annotated `@Remove`, and the EJB container calls the method annotated `@PreDestroy`, if any. The bean's instance is then ready for garbage collection.

Your code controls the invocation of only one lifecycle method: the method annotated `@Remove`. All other methods in Figure 3–2 are invoked by the EJB container. See Chapter 23, "Resource Adapters and Contracts," for more information.

3.7.2 The Lifecycle of a Stateless Session Bean

Because a stateless session bean is never passivated, its lifecycle has only two stages: nonexistent and ready for the invocation of business methods. Figure 3–3 illustrates the stages of a stateless session bean.

Figure 3–3 Lifecycle of a Stateless or Singleton Session Bean



The EJB container typically creates and maintains a pool of stateless session beans, beginning the stateless session bean's lifecycle. The container performs any dependency injection and then invokes the method annotated `@PostConstruct`, if it exists. The bean is now ready to have its business methods invoked by a client.

At the end of the lifecycle, the EJB container calls the method annotated `@PreDestroy`, if it exists. The bean's instance is then ready for garbage collection.

3.7.3 The Lifecycle of a Singleton Session Bean

Like a stateless session bean, a singleton session bean is never passivated and has only two stages, nonexistent and ready for the invocation of business methods, as shown in Figure 3–3.

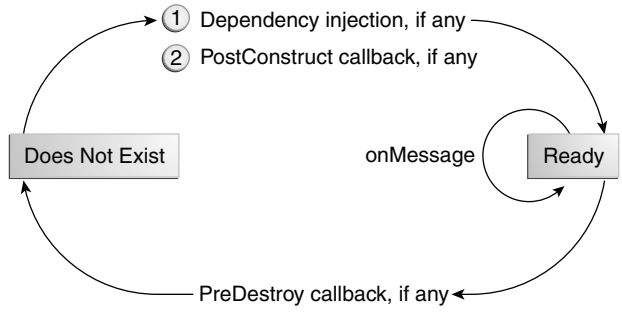
The EJB container initiates the singleton session bean lifecycle by creating the singleton instance. This occurs upon application deployment if the singleton is annotated with the `@Startup` annotation. The container performs any dependency injection and then invokes the method annotated `@PostConstruct`, if it exists. The singleton session bean is now ready to have its business methods invoked by the client.

At the end of the lifecycle, the EJB container calls the method annotated `@PreDestroy`, if it exists. The singleton session bean is now ready for garbage collection.

3.7.4 The Lifecycle of a Message-Driven Bean

Figure 3–4 illustrates the stages in the lifecycle of a message-driven bean.

Figure 3–4 Lifecycle of a Message-Driven Bean



The EJB container usually creates a pool of message-driven bean instances. For each instance, the EJB container performs these tasks.

1. If the message-driven bean uses dependency injection, the container injects these references before instantiating the instance.
2. The container calls the method annotated `@PostConstruct`, if any.

Like a stateless session bean, a message-driven bean is never passivated and has only two states: nonexistent and ready to receive messages.

At the end of the lifecycle, the container calls the method annotated `@PreDestroy`, if any. The bean's instance is then ready for garbage collection.

3.8 Further Information about Enterprise Beans

For more information on Enterprise JavaBeans technology, see

- Enterprise JavaBeans 3.2 specification:

  `http://www.jcp.org/en/jsr/detail?id=345`

- Enterprise JavaBeans 3.2 specification project:

  `https://java.net/projects/ejb-spec/`

This page intentionally left blank

Index

Symbols

`@AccessTimeout` annotation, 84
`@ActivationConfigProperty` annotation, 293
`@AroundConstruct` annotation, 498
`@AroundInvoke` annotation, 498
`@AroundTimeout` annotation, 498
`@Asynchronous` annotation, 114
`@ConcurrencyManagement` annotation, 83
`@DeclareRoles` annotation, 415, 423
`@DenyAll` annotation, 416
`@DependsOn` annotation, 81
`@DiscriminatorColumn` annotation, 138
`@DiscriminatorValue` annotation, 138
`@Embeddable` annotation, 135
`@EmbeddedId` annotation, 130
`@Entity` annotation, 124
`@HttpConstraint` annotation, 385, 403
`@HttpMethodConstraint` annotation, 385, 403
`@Id` annotation, 130
`@IdClass` annotation, 130
`@Interceptor` annotation, 498
`@Interceptors` annotation, 498
`@JMSConnectionFactoryDefinition` annotation, 288
`@JMSDestinationDefinition` annotation, 288
`@Local` annotation, 58, 74
`@Lock` annotation, 83
`@ManyToMany` annotation, 132, 133
`@ManyToOne` annotation, 132
`@MessageDriven` annotation, 293, 343
`@NamedQuery` annotation, 180
`@OneToMany` annotation, 132, 133, 134
`@OneToOne` annotation, 132, 133, 134
`@PermitAll` annotation, 416
`@PersistenceContext` annotation, 141
`@PersistenceUnit` annotation, 142
`@PostActivate` annotation, 75, 77
`@PostConstruct` annotation, 63, 75, 77, 498
`@PreDestroy` annotation, 63, 75, 77, 498
`@PrePassivate` annotation, 75, 77
`@Remote` annotation, 58, 74
`@Remove` annotation, 64, 75, 78
`@Resource` annotation
  JMS resources, 262, 263, 337
`@RolesAllowed` annotation, 415, 423
`@RunAs` annotation, 420
`@Schedule` and `@Schedules` annotations, 97
`@ServletSecurity` annotation, 385, 403
`@Singleton` annotation, 81
`@Startup` annotation, 81
`@Stateful` annotation, 75
`@Timeout` annotation, 95
`@Timeout` method, 96, 98
`@Transient` annotation, 126
`@WebMethod` annotation, 78

A

abstract schemas, 180
access control, 364
`acknowledge` method, 280
acknowledging messages. *See* message acknowledgment
administered objects, 262
  creating and removing, 302
  definition, 257
  See also connection factories, destinations
Administration Console, 34
  starting, 40
`afterBegin` method, 464
`afterCompletion` method, 464

annotations, 3
  interceptor metadata, 498
  security, 369, 403, 412, 415
`appclient` tool, 34
applet container, 15
applets, 9, 10
application client container, 15
application clients, 8
  examples, 337
  securing, 446
application clients, JMS
  building, 303
  examples, 301
  running, 311
application-layer security, 366
`asadmin` tool, 34
asynchronous message consumption, 260
  JMS client example, 309
  See also message-driven beans
asynchronous method invocation
  calling asynchronous business methods, 115
  cancelling, 115
  checking status, 116
  creating asynchronous business methods, 114
  example, 116
  `java.util.concurrent.Future<V>` interface, 113
  retrieving results, 115
  session beans, 113
asynchronous send mechanism, 287
audit modules, pluggable, 371
auditing, 365
`auth-constraint` element, 387
`authenticate` method, 395
authenticating users, 389, 393
authentication, 364, 379
  basic, 390
  basic with EJB, 418
  certificate-based mutual, 437
  client, 436, 438
  digest, 393
  form-based, 391, 405
  mutual, 437, 438
  server, 436
  user name/password-based mutual, 438
authorization, 364
authorization constraints, 386, 387
authorization providers, pluggable, 371
auto commit, 26
`AUTO_ACKNOWLEDGE` mode, 279

B

basic authentication, 390
  EJB, 418
  example, 402
Batch Applications for the Java Platform, 30, 511
batch jobs
  checking status, 540
  defining, 519
  definition, 511
  elements, 517
  instances and executions, 518
  properties and parameters, 518
  starting, 539
  status, 518
  steps, 512
  submitting to batch runtime, 539
batch processing
  context objects, 538
  creating applications, 516
  creating batch artifacts, 533
  definition, 511
  dependency injection, 536
  examples, 541, 548
  implementing chunk steps, 520
  implementing task steps, 522
  introduction, 512
  Java EE framework, 516
  Java EE platform, 516
  Job Specification Language, 519, 523
  packaging applications, 540
  status and decision elements, 514
Bean Validation, 27
  constraints, 174
  custom constraints, 584
  examples, 174
  Java Persistence API, 128
  JavaServer Faces applications, 176
bean-managed transactions. *See* transactions, bean-managed
`beforeCompletion` method, 464
BLOBs. *See* persistence, BLOBs
business logic, 50

business methods, 60
client calls, 77
exceptions, 78
locating, 69
requirements, 78
transactions, 462, 464, 466, 467
`BytesMessage` interface, 275

C

`CallbackHandler` interface, 447
`capture-schema` tool, 34
CDI. *See* Contexts and Dependency Injection for Java EE (CDI)
certificate authorities, 432
certificates, 366
client, 439
digital, 367, 431, 432
server, 433, 435
using for authentication, 435
client certificates, generating, 439
client ID, for durable subscriptions, 270
`CLIENT_ACKNOWLEDGE` mode, 280
clients
authenticating, 436, 438
JMS, 257
securing, 446
CLOBs. *See* persistence, CLOBs
collections, persistence, 126, 226
`commit` method, 465
`commit` method (JMS), 284
commits. *See* transactions, commits
Common Client Interface, Connector architecture, 477
component-managed sign-on, 448, 449
components, Java EE, 8
concurrency
definition, 559
examples, 562, 567
security and, 562
threads and processes, 560
transactions and, 561
Concurrency Utilities for Java EE, 30, 559
concurrent access, 457
concurrent access to entity data, 231
confidentiality, 379
connection factories, 262
creating, 339, 354
injecting resources, 262, 337
`Connection` interface, 465, 468
`Connection` interface (JMS), 264
`ConnectionFactory` interface (JMS), 262
connections, JMS
introduction, 264
managing in web and EJB components, 291
connections, securing, 379
connectors. *See* Java EE Connector architecture
constructors, 502
container-managed sign-on, 448
container-managed transactions. *See* transactions, container-managed
containers, 13
application client, 15
configurable services, 13
nonconfigurable services, 14
security, 360, 369
services, 13
trust between, 422
See also EJB container, embedded enterprise bean container, web container
Contexts and Dependency Injection for Java EE (CDI), 27
events, 587
observer methods, 587
Contexts and Dependency Injection for Java EE (CDI)I
events, 585
contexts, JMS, 264
`ContextService` interface, 561
conversational state, 51
`createBrowser` method, 313
`createTimer` method, 96
credential, 374
Criteria API, 215
creating queries, 219
examples, 169
expressions, 222
path navigation, 221
query execution, 226
query results, 222, 224
criteria queries, string-based, 227
cryptography, public-key, 432

D

data encryption, 436
data integrity, 364, 457, 458
databases
  clients, 50
  connections, 78, 466
  creating tables, 148
  data recovery, 457
  dropping tables, 148
  EIS tier, 6
  loading data, 149
  message-driven beans and, 54
  multiple, 466, 467
  See also transactions
DDL scripts, 148
  loading data, 149
debugging Java EE applications, 44
declarative security, 360, 384, 412
  example, 422
delivery delay for messages, 283
delivery modes, 281
  `JMSDeliveryMode` message header field, 274
`DeliveryMode` interface, 281
Dependency Injection for Java (JSR 330), 27
deployer roles, 19
deployment, 70
deployment descriptors, 360, 369
  enterprise bean, 370, 412, 413
  `security-role-mapping` element, 377
  `security-role-ref` element, 400
  web application, 370
`Destination` interface, 263
destinations, 263
  creating, 339, 354
  injecting resources, 263, 337
  `JMSDestination` message header field, 274
  temporary, 283, 349
  See also queues, temporary JMS destinations, topics
development roles, 17
  application assemblers, 19
  application client developers, 19
  application component providers, 18
  application deployers and administrators, 19
  enterprise bean developers, 18
  Java EE product providers, 18
  tool providers, 18
  web component developers, 18
digest authentication, 393
digital signatures, 432
domains, 39
downloading GlassFish Server, 36
`DUPS_OK_ACKNOWLEDGE` mode, 280
durable subscriptions, 269
  examples, 319, 330, 341
  shared, 272

E

EIS tier, 12
  security, 448
EJB container, 15
  container-managed transactions, 459
  message-driven beans, 292
  `onMessage` method, invoking, 338
  services, 49, 50, 412
  See also embedded enterprise bean container
EJB JAR files, 63
`EJBContext` interface, 464, 465, 466
`ejb-jar.xml` file, 370, 413
embeddable classes, persistence, 135
embedded enterprise bean container
  creating, 107
  developing applications, 106
  examples, 109
  initializing enterprise bean modules, 107
  overview, 105
  running applications, 106
  session bean references, 108
  shutting down, 108
  See also EJB container, enterprise beans
end-to-end security, 368
enterprise applications, 3
  securing, 411
enterprise beans, 11, 23
  accessing, 55
  classes, 62
  compiling, 70
  contents, 62
  defined, 49
  dependency injection, 56
  deployment, 63
  distribution, 57

exceptions, 104
finding, 108
`getCallerPrincipal` method, 418
implementor of business logic, 11
interceptors, 497
interfaces, 55, 63
`isCallerInRole` method, 418
JNDI lookup, 56
lifecycles, 63
local access, 58
local interfaces, 58
packaging, 70
performance, 58
programmatic security, 418
remote access, 60
remote interfaces, 60
securing, 412
testing, 109
timer service, 92
types, 51
web services, 51, 61, 89
See also business methods, embedded enterprise bean container, message-driven beans, session beans
Enterprise Information Systems. *See* EIS tier
entities
abstract, 136
abstract schema names, 182
application-managed entity managers, 141
cascading operations, 134, 135
collections, 198
container-managed entity managers, 141
controlling caching, 244
creating, 162
discriminator columns, 138
entity manager, 141
finding, 143, 162
inheritance, 136, 167
inheritance mapping, 138
lifecycle, 143
managing, 141, 162
mapping to multiple tables, 159
non-entity superclasses, 138
overview, 123
persistent fields, 124
persistent properties, 124
persisting, 143
primary keys, 130
querying, 146
relationships, 163
removing, 144, 164
requirements, 124
superclasses, 137
synchronizing, 145
validating, 128
entity data
lock modes, 233
optimistic locking, 231, 232
pessimistic locking, 232, 235
entity graphs, 237
named, 240
entity relationships
bidirectional, 133
many-to-many, 132, 167
many-to-one, 132
multiplicity, 132
one-to-many, 132
one-to-one, 132
query language, 134
unidirectional, 133
`EntityManager` interface, 141
`equals` method, 131
examples, 35
asynchronous method invocation, session beans, 116
basic authentication, 402
batch processing, 541, 548
Bean Validation, 174
building, 41
concurrency, 562, 567
connectors, 481
Criteria API, 169
directory structure, 41
Duke's Bookstore case study, 573
Duke's Forest case study, 595
Duke's Tutoring case study, 583
embedded enterprise bean container, 109
interceptors, 507
JMS asynchronous message consumption, 309
JMS durable subscriptions, 319
JMS in a web application, 332
JMS local transactions, 322
JMS message acknowledgment, 317
JMS queue browsing, 313

examples (cont.)
JMS shared durable subscriptions, 330
JMS synchronous message consumption, 307
JMS with entities, 346
JMS with session beans, 341
message-driven beans, 336, 341, 346
persistence, 151
primary keys, 131
query language, 163, 183
required software, 35
resource adapters, 481
security, 405, 422, 427
sending JMS messages, 304
servlets, 69
session beans, 68, 73
singleton session beans, 81
timer service, 100
web clients, 69
web services, 89
exceptions
business methods, 78
enterprise beans, 104
JMS, 277
rolling back transactions, 104, 464
transactions, 461, 462
expiration of JMS messages, 282
`JMSExpiration` message header field, 274

F

fetch graphs
See persistence, fetch graphs
fetch plans, 237
foreign keys, 154
form-based authentication, 391

G

garbage collection, 66
`getBody` method, 276
`getCallerPrincipal` method, 418, 427
`getRemoteUser` method, 397
`getRollbackOnly` method, 297, 466
`getStatus` method, 466
`getUserPrincipal` method, 397
GlassFish Server
adding users to, 375
downloading, 36
enabling debugging, 45
installation tips, 36
securing, 370
server log, 44
SSL connectors, 380
starting, 39
stopping, 40
tools, 33
groups, 374
managing, 375

H

`hashCode` method, 131
helper classes, 63
session bean example, 79
HTTP
basic authentication, 390
over SSL, 436
HTTPS, 367, 380, 387, 432
`HttpServletRequest` interface, 397

I

identification, 364
`InitialContext` interface, 31
integrity, 379
of data, 364
Interceptors, 497
around-construct, 502
example, 507
invoking, 506
lifecycle callback events, 502
methods, 500
ordering, 506
`isCallerInRole` method, 418, 427
`isUserInRole` method, 397

J

JAAS, 33, 365, 447
login modules, 448
JACC, 29, 371
JAF, 31
JAR signatures, 366
JASPIC, 29

Java API for JavaBean Validation. *See* Bean Validation
Java API for JSON Processing, 29
Java API for WebSocket, 29
  endpoints, 585, 587
Java API for XML Binding (JAXB), 32
Java API for XML Processing (JAXP), 32
Java API for XML Web Services. *See* JAX-WS
Java Authentication and Authorization Service. *See* JAAS
Java Authentication Service Provider Interface for Containers (JASPIC), 29
Java Authorization Contract for Containers. *See* JACC
Java BluePrints, 41
Java Cryptography Extension (JCE), 365
Java Database Connectivity API. *See* JDBC API
Java DB, 34
  starting, 40
  stopping, 41
Java EE applications, 6
  debugging, 44
  deploying, 70
  iterative development, 71
  tiers, 6
Java EE clients, 8
  See also application clients, web clients
Java EE components, 8
Java EE Connector architecture, 28, 471
  examples, 481
Java EE modules, EJB, 63
Java EE platform, 6
  APIs, 20
  JMS and, 256
  overview, 3
Java EE security model, 13
Java EE servers, 15
Java EE transaction model, 13
Java Generic Security Services, 365
Java GSS-API, 365
Java Message Service (JMS) API, 27, 253
  See also JMS, message-driven beans
Java Naming and Directory Interface API. *See* JNDI
Java Persistence API, 26, 123
  See also persistence
Java Persistence API query language. *See* query language
Java Persistence Criteria API. *See* Criteria API
Java Secure Sockets Extension (JSSE), 365
Java Servlet technology, 24
  See also servlets
Java Transaction API, 26, 466
JavaBeans Activation Framework (JAF), 31
JavaBeans components, 9
JavaMail API, 28
JavaServer Faces technology, 10, 24
  using Ajax with, 585
JavaServer Pages Standard Tag Library (JSTL), 25
JavaServer Pages technology, 25
JAXB, 32
JAXP, 32
JAX-RS, 26
JAX-WS, 32
JCE, 365
JDBC API, 30
  resources, 584
JDBC security realm, 441
JMS, 27
  administered objects, 262
  advanced features, 278
  application client examples, 301
  architecture, 257
  basic concepts, 257
  definition, 254
  examples, 299, 332, 336, 341, 346
  introduction, 253
  Java EE platform, 256, 287
  messaging styles, 258
  programming model, 260
`JMSConsumer` interface, 266
`JMSContext` interface, 264
  injecting objects, 290
  managing in web and EJB components, 291
`JMSCorrelationID` message header field, 274
`JMSDeliveryMode` message header field, 274
`JMSDeliveryTime` message header field, 274
`JMSDestination` message header field, 274
`JMSException` class, 277
`JMSExpiration` message header field, 274
`JMSMessageID` message header field, 274
`JMSPriority` message header field, 274
`JMSProducer` interface, 266
  method chaining, 283
`JMSRedelivered` message header field, 274

JMSReplyTo message header field, 274
JMSTimestamp message header field, 274
JMSType message header field, 274
JNDI, 31
  data source naming subcontexts, 31
  enterprise bean lookup, 56
  enterprise bean naming subcontexts, 31
  environment naming contexts, 31
  jms naming subcontext, 263
  namespace for JMS administered objects, 262
  naming contexts, 31
  naming environments, 31
  naming subcontexts, 31
Job Specification Language, 523
  batchlet element, 528
  chunk element, 526
  decision element, 532
  flow element, 531
  job element, 524
  partition element, 528
  split element, 532
  step element, 524
JSSE, 365
JSTL, 25
JTA, 26, 466
JTS API, 466
JUnit, 109

K

Kerberos, 365, 366
key pairs, 432
keystores, 366, 431, 433
  managing, 432
keytool utility, 432

L

listeners
  HTTP, 371
  IIOP, 371
load graphs
  See persistence, load graphs
local interfaces, 58
local transactions, 284
log, server, 44
login configuration, 389, 393
login method, 395
login modules, 447
logout method, 395

M

Managed Beans specification, 26
ManagedExecutorService interface, 561
ManagedScheduledExecutorService interface, 561
ManagedThreadFactory interface, 561
MapMessage interface, 275
Maven tool, 38
message acknowledgment, 279
  bean-managed transactions, 297
  example, 317
message bodies, 275
message consumers, 266
message consumption, 260
  asynchronous, 260, 309
  synchronous, 260, 307
message headers, 273
message IDs, JMS, 274
Message interface, 275
message listeners, 53, 267
  examples, 310, 349
message producers, 266
message properties, 274
message security, 384
message selectors, 268
message subscriptions
  durable, 269
  shared, 272
message-driven beans, 23, 53
  accessing, 54
  coding, 338, 343, 349
  defined, 53
  examples, 336, 341, 346
  garbage collection, 66
  introduction, 292
  onMessage method, 54, 338
  requirements, 338
  transactions, 55, 459, 465
MessageListener interface, 267
messages
  integrity, 436
  securing, 368

messages, JMS
body formats, 275
browsing, 277
definition, 257
delivery delay, 283
delivery modes, 281
expiration, 282
headers, 273
introduction, 273
persistence, 281
priority levels, 282
properties, 274
specifying options for sending, 281
messaging styles, 258
point-to-point, 258
publish/subscribe, 259
messaging, definition, 253
metadata annotations
resource adapters, 475
security, 369
Metamodel API, 215
using, 169, 217
method permissions, 414
annotations, 415
mutual authentication, 437, 438

N

naming contexts, 31
naming environments, 31
NetBeans IDE, 37
`NON_PERSISTENT` delivery mode, 281
non-repudiation, 365

O

`ObjectMessage` interface, 275
objects, administered, 262
creating and removing, 302
`onMessage` method
introduction, 267
message-driven beans, 54, 293, 338

P

`package-appclient` tool, 34
permissions, security policy, 371
persistence
BLOBs, 160
cascade operations, 160
CLOBs, 160
collections, 126
concurrent access to entity data, 231
configuration, 145
context, 141
creating database tables, 148
criteria queries, 238, 242
DDL scripts, 148
dropping database tables, 148
eager fetching, 237
embeddable classes, 135
entities, 123
entity graph properties, 241
examples, 151
fetch graphs, 239
JMS example, 346
JMS messages, 281
JPQL, 238
lazy fetching, 237
load graphs, 239
loading data, 149
locking strategies, 231
many-to-many, 167
maps, 127
named entity graphs, 240
one-to-many, 154
one-to-one, 153
overview, 123
persistence units, 145
persistent fields, 125
primary keys, 130, 155, 156
properties, 125
queries, 123, 146, 163, 180, 237
creating, 219
Criteria, 215
dynamic, 180
executing, 226
expressions, 222
joins, 221
parameters, 181
path navigation, 221
results, 222, 224
static, 180

persistence, queries (cont.)
typesafe, 215
See also query language
query hints, 241
query language, 134
relationships, 152
schema creation, 147, 584
scope, 145
second-level cache, 243
self-referential relationships, 152
string-based criteria queries, 227
temporal types, 161
persistence units, query language, 179, 197
`PERSISTENT` delivery mode, 281
pluggable audit modules, 371
pluggable authorization providers, 371
point-to-point messaging style, 258
See also queues
POJOs, 4
policy files, 366
primary keys, 154
compound, 156
defined, 130
examples, 131
generated, 155
principal, 374
priority levels, for messages, 282
`JMSPriority` message header field, 274
programmatic security, 360, 370, 384, 413
example, 427
programming model, JMS, 260
providers, JMS, 257
public key certificates, 436
public-key cryptography, 432
publish/subscribe messaging style
consuming messages, 268
durable subscriptions, 269
introduction, 259
See also topics

Q

Quality of Service, 365
queries
criteria, 238
JPQL, 238
persistence, 237
query language
`ABS` function, 208
abstract schemas, 180, 182, 198
`ALL` expression, 206
`ANY` expression, 206
arithmetic functions, 207
`ASC` keyword, 213
`AVG` function, 212
`BETWEEN` expression, 188, 203
Boolean literals, 202
Boolean logic, 210
case expressions, 209
collection member expressions, 198, 205
collections, 198, 205
compared to SQL, 185, 197, 199
comparison operators, 188, 203
`CONCAT` function, 207
conditional expressions, 187, 201, 202, 210
constructors, 213
`COUNT` function, 212
`DELETE` expression, 188, 189
`DELETE` statement, 183
`DESC` keyword, 213
`DISTINCT` keyword, 184
domain of query, 179, 194, 197
duplicate values, 184
enum literals, 202
equality, 210
`ESCAPE` clause, 204
examples, 163, 183
`EXISTS` expression, 206
`FETCH JOIN` operator, 199
`FROM` clause, 182, 194
grammar, 189
`GROUP BY` clause, 183, 214
`HAVING` clause, 183, 214
identification variables, 182, 194, 197
identifiers, 194
`IN` operator, 199, 204
`INNER JOIN` operator, 199
input parameters, 186, 202
`IS EMPTY` expression, 187
`IS FALSE` operator, 210
`IS NULL` expression, 187
`IS TRUE` operator, 210
`JOIN` statement, 185, 199
`LEFT JOIN` operator, 199

LEFT OUTER JOIN operator, 199
LENGTH function, 207
LIKE expression, 187, 204
literals, 201
LOCATE function, 207
LOWER function, 208
MAX function, 212
MEMBER expression, 205
MIN function, 212
MOD function, 208
multiple declarations, 197
multiple relationships, 186
named parameters, 184, 202
navigation, 185, 186, 198, 201
negation, 210
NOT operator, 210
null values, 205, 209
numeric comparisons, 210
numeric literals, 201
operator precedence, 203
operators, 203
ORDER BY clause, 183, 213
parameters, 184
parentheses, 202
path expressions, 180, 199
positional parameters, 202
range variables, 198
relationship fields, 180
relationships, 180, 185, 186
return types, 211
root, 198
scope, 179
SELECT clause, 182, 211
setNamedParameter method, 184
SIZE function, 208
SQRT function, 208
state fields, 180
string comparison, 210
string functions, 207
string literals, 201
subqueries, 206
SUBSTRING function, 207
SUM function, 212
syntax, 182, 189
TRIM function, 208
types, 200, 210
UPDATE expression, 183, 188, 189
  UPPER function, 208
  WHERE clause, 183, 201
  wildcards, 204
query roots, 220
Queue interface, 263
QueueBrowser interface, 277
  JMS client example, 313
queues, 263
  browsing, 277, 313
  creating, 263, 354
  injecting resources, 337
  temporary, 283, 349

R

realms, 372, 373
  admin-realm, 373
  certificate, 373, 435
  configuring, 371
  file, 373
  JDBC, 441
receiveBody method, 276
recover method, 280
redelivery of messages, 279, 280
  JMSRedelivered message header field, 274
relationship fields, query language, 180
relationships
  direction, 133
  unidirectional, 155
remote interfaces, 60
Remote Method Invocation (RMI), and messaging, 253
request/reply mechanism
  JMSCorrelationID message header field, 274
  JMSReplyTo message header field, 274
  temporary destinations and, 284
Required transaction attribute, 297
resource adapters, 28, 471
  examples, 481
  metadata annotations, 475
  security, 449
resources, 471
resources, JMS
  creating for Java EE applications, 288
  injecting in EJB and web components, 289
RESTful web services, 26

roles, 374
application, 377
declaring, 394
mapping to groups, 377
mapping to users, 377
referencing, 415
security, 376, 394, 414, 415
`rollback` method, 465, 466
`rollback` method (JMS), 284
rollbacks. *See* transactions, rollbacks
run-as identity, 420

S

SAAJ, 33
SASL, 366
`schemagen` tool, 34
schemas
creating, 147
deployment descriptors, 369
secure connections, 379
Secure Sockets Layer (SSL), 379
security
annotations, 369, 403, 412
application, 364, 366
application clients, 446
callback handlers, 447
component-managed sign-on, 449
concurrency and, 562
constraints, 385
container trust, 422
container-managed sign-on, 448
containers, 360, 369
context for enterprise beans, 418
declarative, 360, 369, 384, 412
deploying enterprise beans, 422
EIS applications, 448
end-to-end, 368
enterprise applications, 411
enterprise beans, 412
examples, 405, 422, 427
groups, 374
introduction, 359
JAAS login modules, 448
Java SE, 365
login forms, 447
login modules, 447
mechanism features, 363
mechanisms, 365, 366
message, 384
message-layer, 368
method permissions, 414, 415
overview, 360
policy domain, 374
programmatic, 360, 370, 384, 395, 413
programmatic login, 448
propagating identity, 420
realms, 373, 441
resource adapters, 449
role names, 394, 415
roles, 374, 376, 394, 414
run-as identity, 420
simple walkthrough, 360
transport-layer, 367, 379
users, 373
web applications, 383
web components, 383
security constraints, 385
multiple, 389
security domain, 374
security identity
propagating, 420
specific identity, 421
security role references, 400
security roles, 376, 414
`security-role-mapping` element, 377
`security-role-ref` element, 400
`send` method, 266
sending messages
Java EE components, 291
JMS client example, 304
sending messages asynchronously, 287
server authentication, 436
server certificates, 431
server log, 44
servlets, 10
compiling, 70
examples, 69
packaging, 70
session beans, 23, 51
activation, 64
bean-managed concurrency, 82, 85
business interfaces, 55
clients, 51

concurrent access, 82
container-managed concurrency, 82, 83
databases, 464
eager initialization, 81
examples, 68, 73, 81, 89, 341
handling errors, 86
no-interface views, 55
passivation, 64
requirements, 75
singleton, 52, 81
stateful, 51, 53
stateless, 52, 53
transactions, 459, 464, 465
web services, 62, 90
See also asynchronous method invocation
`Session` interface, 264
sessions, JMS, 264
managing in web and EJB components, 291
`SessionSynchronization` interface, 464
`setRollbackOnly` method, 297, 464, 466
shared durable subscriptions, 272
shared message subscriptions, 272
sign-on
component-managed, 448, 449
container-managed, 448
Simple Authentication and Security Layer (SASL), 366
SOAP messages, 16, 33
securing, 368
SOAP with Attachments API for Java (SAAJ), 33
SQL, 30, 185, 197, 199
loading data, 149
SQL92, 209
SSL, 367, 379, 387, 436
connectors, GlassFish Server, 380
handshake, 379
verifying support, 380
state fields, query language, 180
`StreamMessage` interface, 275
string-based criteria queries, 227
subscription names, for durable subscribers, 270
synchronous message consumption, 260
Java EE components, 291
JMS client example, 307

T

temporary JMS destinations, 283
examples, 349
testing
enterprise beans, 109
unit, 109
`TextMessage` interface, 275
timer service, 92
automatic timers, 93, 97
calendar-based timer expressions, 93
cancelling timers, 98
creating timers, 96
examples, 100
exceptions, 99
`getInfo` method, 99
`getNextTimeout` method, 99
`getTimeRemaining` method, 99
getting information, 99
programmatic timers, 93, 95
saving timers, 98
transactions, 99
timestamps for JMS messages, 274
`Topic` interface, 263
topics, 263
creating, 263, 354
durable subscriptions, 269
temporary, 283
transactions, 457, 458
application-managed, 141
attributes, 459, 463
bean-managed, 296, 465, 466
boundaries, 459, 465
business methods. *See* business methods, transactions
commits, 458, 464
concurrency and, 561
container-managed, 296, 459
container-managed transaction demarcation, 459
defined, 458
distributed, 296
examples, 322
exceptions. *See* exceptions, transactions
JDBC, 467
JMS and enterprise bean applications, 292
JTA, 465, 466
local, 284

transactions (cont.)
managers, 462, 466, 467
message-driven beans. *See* message-driven beans, transactions
nested, 459, 466
`Required` attribute, 297
rollbacks, 458, 464, 466
scope, 459
session beans. *See* session beans, transactions
timeouts, 467
timer service, 99
web components, 468
transport guarantees, 387
`transport-guarantee` element, 387
transport-layer security, 367, 379
truststores, 431, 433
managing, 432

U

user data constraints, 386, 387
`user-data-constraint` element, 387
users, 373
adding to GlassFish Server, 375
managing, 375
`UserTransaction` interface, 142, 465, 466, 468
message-driven beans, 296
utility classes, 63

V

validating entities, 128

W

W3C, 32
web applications
JMS example, 332
securing, 383
web clients, 8
examples, 69
web components, 10
applets bundled with, 10
securing, 383
transactions, 468
types, 10
utility classes bundled with, 10
web container, 15
web resource collections, 386
web resources, unprotected, 386
web services, 15
endpoint implementation classes, 89
examples, 89
See also enterprise beans
`web-resource-collection` element, 386
`web.xml` file, 370, 413
work flows, 53
WSDL, 16
`wsgen` tool, 34
`wsimport` tool, 34

X

`xjc` tool, 34
XML, 16