Java Data Objects

Java Data Objects revolutionizes the way Java developers interact with databases and other datastores. JDO allows you to store and retrieve objects in a way that’s natural to Java programmers. Instead of working with JDBC or EJB’s container-managed persistence, you work directly with your Java objects. You don’t have to copy data to and from database tables or issue SELECTs to perform queries: your JDO implementation takes care of persistence behind-the-scenes, and you make queries based on the fields of your Java objects, using normal Java syntax.

The result is software that is truly object-oriented, not code that is partially object-oriented, with a large database-shaped lump on the back end. JDO lets you save plain, ordinary Java objects and does not force you to use different data models and types for dealing with storage. As a result, your code becomes easier to maintain, easier to reuse, and easier to test. And you’re not tied to a specific database vendor: your JDO code is entirely database-independent. You don’t even need to know whether the datastore is a relational database, an object database, or just a set of files.

This book, written by the JDO Specification Lead and one of the key contributors to the JDO Specification, is the definitive work on the JDO API. It gives you a thorough introduction to JDO, starting with a simple application that demonstrates many of JDO’s capabilities. It shows you how to make classes persistent, how JDO maps persistent classes to the database, how to configure JDO at runtime, how to perform transactions, and how to make queries. More advanced chapters cover optional features such as nontransactional access and optimistic transactions. The book concludes by discussing the use of JDO in web applications and J2EE environments.

Whether you only want to read up on an interesting new technology, or are seriously considering an alternative to JDBC or EJB CMP, you’ll find that this book is essential. It provides by far the most authoritative and complete coverage available.

David Jordan is a key contributor to the JDO expert group and the moderator of JDOcentral.com. He is the principal of Object Identity, Inc., a firm that provides JDO consulting services. Craig Russell is the Specification Lead for Java Data Objects, and the architect for the Container Managed Persistence component of the Sun ONE Application Server.

“A thorough yet clear guide to using JDO’s persistence mechanisms.”

—Martin Fowler
Java™ Data Objects
Related titles from O'Reilly

Ant: The Definitive Guide
Building Java™ Enterprise Applications
Database Programming with JDBC and Java™
Developing JavaBeans™
Enterprise JavaBeans™
J2ME in a Nutshell
Java™ 2D Graphics
Java™ and SOAP
Java™ & XML
Java™ and XML Data Binding
Java™ and XSLT
Java™ Cookbook
Java™ Cryptography
Java™ Data Objects
Java™ Distributed Computing
Java™ Enterprise in a Nutshell
Java™ Examples in a Nutshell
Java™ Foundation Classes in a Nutshell
Java™ I/O
Java™ in a Nutshell
Java™ Internationalization
Java™ Message Service
Java™ Network Programming
Java™ NIO
Java™ Performance Tuning
Java™ Programming with Oracle SQLJ
Java™ Security
JavaServer™ Pages
Java™ Servlet Programming
Java™ Swing
Java™ Threads
Java™ Web Services
JXTA in a Nutshell
Learning Java™
Mac OS X for Java™ Geeks
NetBeans: The Definitive Guide
Programming Jakarta Struts
To my wife Tina,
whose emotional and financial support
made this book possible;
and to Jennifer and Jeremy, who now think that
their daddy has become addicted to his computer.
—David Jordan

To Kathy, Chris, Ali, and Juliana.
—Craig Russell
# Table of Contents

## Foreword ................................................................. xv
## Preface ...................................................................... xvii

### 1. An Initial Tour ......................................................... 1
   - Defining a Persistent Object Model .......................... 2
     - The Classes to Persist ........................................ 3
     - Declaring Classes to Be Persistent ....................... 7
   - Project Build Environment ..................................... 8
     - Jars Needed to Use the JDO Reference Implementation  9
   - Project Directory Structure .................................. 10
   - Enhancing Classes for Persistence ......................... 10
   - Establish a Datastore Connection and Transaction .......... 11
     - Acquiring a PersistenceManager ................................ 11
     - Creating a FOStore Datastore .............................. 12
   - Operations on Instances ......................................... 14
     - Making Instances Persistent ............................... 15
     - Accessing Instances ......................................... 18
     - Modifying an Instance ....................................... 22
     - Deleting Instances .......................................... 26
   - Summary ................................................................ 27

### 2. An Overview of JDO Interfaces ................................... 28
   - The javax.jdo Package ........................................... 28
     - JDO Exception Classes ....................................... 31
   - The javax.jdo.spi Package .................................... 34
Optional Features
- Identity Options 36
- Optional Collections 36
- Transaction-Related Optional Features 37

3. **JDO Architectures** ............................................................... 38

   Architecture Within Application JVM 40
   - Single PersistenceManager 40
   - Multiple PersistenceManagers Accessing the Same datastore 41
   - Multiple PersistenceManagers Accessing Different Datastores 43
   - Shared Implementation Cache 44

   Datastore Access 46
   - Direct Access of Filesystem or Local datastore 46
   - Remote Access of a JDO Server 46
   - Remote Access of a SQL datastore 46

   System Architectures with a JDO Application 48
   - JDO Rich Client with Local datastore 48
   - JDO Applications in a Web Server 48
   - JDO Applications as Web Services 48
   - Rich Client Connecting to Application Server with EJB Components 49
   - Web Server with EJB Server 50
   - EJB Session Beans Using Session Bean Façades 50
   - JDO Providing Container-Managed Persistence 51

4. **Defining Persistent Classes** .................................................. 52

   Kinds of Classes and Instances 52
   - Kinds of Classes 53
   - Kinds of Instances 53

   Java Classes and Metadata 54
   - JDO Metadata 55
   - Inheritance 58
   - The Media Mania Object Model 60

   Fields 63
   - Supported Types 64
   - Persistence of Fields 66
   - Complete Metadata for the Media Mania Model 68
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>5. Datastore Mappings</strong></td>
<td>70</td>
</tr>
<tr>
<td>Mapping Approaches</td>
<td>71</td>
</tr>
<tr>
<td>Relational Modeling Constructs</td>
<td>72</td>
</tr>
<tr>
<td>SQL 99</td>
<td>73</td>
</tr>
<tr>
<td>Modeling Constructs in Java and Relational Models</td>
<td>74</td>
</tr>
<tr>
<td>Mapping Classes to Tables</td>
<td>75</td>
</tr>
<tr>
<td>Mapping a Single-Valued Field to a Column</td>
<td>76</td>
</tr>
<tr>
<td>Name-Mapping</td>
<td>76</td>
</tr>
<tr>
<td>Type-Mapping</td>
<td>77</td>
</tr>
<tr>
<td>Indexes</td>
<td>78</td>
</tr>
<tr>
<td>Identity</td>
<td>78</td>
</tr>
<tr>
<td>Inheritance</td>
<td>80</td>
</tr>
<tr>
<td>References</td>
<td>82</td>
</tr>
<tr>
<td>Collections and Relationships</td>
<td>83</td>
</tr>
<tr>
<td>Using a Foreign Key</td>
<td>83</td>
</tr>
<tr>
<td>Using a Join Table</td>
<td>89</td>
</tr>
<tr>
<td>One-to-One Relationships</td>
<td>90</td>
</tr>
<tr>
<td>Representing Lists and Maps</td>
<td>92</td>
</tr>
<tr>
<td><strong>6. Class Enhancement</strong></td>
<td>94</td>
</tr>
<tr>
<td>Enhancement Approaches</td>
<td>95</td>
</tr>
<tr>
<td>Reference Enhancer</td>
<td>96</td>
</tr>
<tr>
<td>Vendor-Specific Enhancement</td>
<td>96</td>
</tr>
<tr>
<td>Binary Compatibility</td>
<td>97</td>
</tr>
<tr>
<td>Enhancement Effects on Your Code</td>
<td>98</td>
</tr>
<tr>
<td>Changes Made by the Enhancer</td>
<td>98</td>
</tr>
<tr>
<td>Metadata</td>
<td>99</td>
</tr>
<tr>
<td>Instance-Level Data</td>
<td>100</td>
</tr>
<tr>
<td>Field Mediation</td>
<td>101</td>
</tr>
<tr>
<td><strong>7. Establishing a JDO Runtime Environment</strong></td>
<td>104</td>
</tr>
<tr>
<td>Configuring a PersistenceManagerFactory</td>
<td>105</td>
</tr>
<tr>
<td>Connection Properties</td>
<td>108</td>
</tr>
<tr>
<td>Optional Feature Properties</td>
<td>110</td>
</tr>
<tr>
<td>Flags</td>
<td>111</td>
</tr>
<tr>
<td>Flags Settings in Multiple Interfaces</td>
<td>111</td>
</tr>
<tr>
<td>Determining Optional Feature Properties and Default Flag Settings</td>
<td>112</td>
</tr>
</tbody>
</table>
Advanced Topics 203
   Choosing an Identity Type 204
   Using Identity Versus a Query 204
   Identity Across Persistence Managers 204

11. Lifecycle States and Transitions 206
   Lifecycle States 206
      Transient 207
      Persistent-New 207
      Hollow 208
      Persistent-Clean 210
      Persistent-Dirty 210
      Persistent-Deleted 210
      Persistent-New-Deleted 210
   State Interrogation 211
   State Transitions 212
      State Transitions During a Datastore Transaction 212
      State Transitions When a Transaction Completes 213
      States Between Transactions 214

12. Field Management 215
   Transactional Fields 215
   null Values 215
   Retrieval of Fields 216
      Default Fetch Group 216
      Retrieving All Fields 217
      The Management of Fields 219
   Serialization 219
   Managing Fields During Lifecycle Events 220
   First- and Second-Class Objects 221
      Specifying a Second-Class Object 224
      Embedding Collection Elements 225
      Persistent Classes as Second-Class Objects 226
      Sharing of Instances 226
<table>
<thead>
<tr>
<th>Chapter</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>13</td>
<td><strong>Cache Management</strong></td>
<td>228</td>
</tr>
<tr>
<td></td>
<td>Explicit Management of Instances in the Cache</td>
<td>228</td>
</tr>
<tr>
<td></td>
<td>Refreshing Instances</td>
<td>229</td>
</tr>
<tr>
<td></td>
<td>Evicting Instances</td>
<td>229</td>
</tr>
<tr>
<td></td>
<td>Cloning</td>
<td>230</td>
</tr>
<tr>
<td></td>
<td>Transient-Transactional Instances</td>
<td>231</td>
</tr>
<tr>
<td></td>
<td>Transient-Transactional Lifecycle States</td>
<td>231</td>
</tr>
<tr>
<td></td>
<td>State Interrogation</td>
<td>232</td>
</tr>
<tr>
<td></td>
<td>State Transitions</td>
<td>232</td>
</tr>
<tr>
<td></td>
<td>Making a Persistent Instance Transient</td>
<td>233</td>
</tr>
<tr>
<td>14</td>
<td><strong>Nontransactional Access</strong></td>
<td>235</td>
</tr>
<tr>
<td></td>
<td>Nontransactional Features</td>
<td>235</td>
</tr>
<tr>
<td></td>
<td>Reading Outside a Transaction</td>
<td>238</td>
</tr>
<tr>
<td></td>
<td>Persistent-Nontransactional State</td>
<td>239</td>
</tr>
<tr>
<td></td>
<td>Retaining Values at Transaction Commit</td>
<td>241</td>
</tr>
<tr>
<td></td>
<td>Restoring Values at Transaction Rollback</td>
<td>242</td>
</tr>
<tr>
<td></td>
<td>Before Image</td>
<td>243</td>
</tr>
<tr>
<td></td>
<td>Restoring Persistent Instances</td>
<td>244</td>
</tr>
<tr>
<td></td>
<td>Restoring Persistent-New Instances</td>
<td>244</td>
</tr>
<tr>
<td></td>
<td>Modifying Persistent Instances Outside a Transaction</td>
<td>245</td>
</tr>
<tr>
<td></td>
<td>Hot Cache Example</td>
<td>246</td>
</tr>
<tr>
<td>15</td>
<td><strong>Optimistic Transactions</strong></td>
<td>251</td>
</tr>
<tr>
<td></td>
<td>Verification at Commit</td>
<td>252</td>
</tr>
<tr>
<td></td>
<td>Recovery from a Failed Transaction</td>
<td>253</td>
</tr>
<tr>
<td></td>
<td>Setting Optimistic Transaction Behavior</td>
<td>254</td>
</tr>
<tr>
<td></td>
<td>Optimistic Example</td>
<td>254</td>
</tr>
<tr>
<td></td>
<td>Optimistic Transaction State Transitions</td>
<td>256</td>
</tr>
<tr>
<td></td>
<td>Deleting Instances</td>
<td>257</td>
</tr>
<tr>
<td></td>
<td>Making Instances Transactional</td>
<td>257</td>
</tr>
<tr>
<td></td>
<td>Modifying Instances</td>
<td>257</td>
</tr>
<tr>
<td></td>
<td>Commit</td>
<td>258</td>
</tr>
<tr>
<td></td>
<td>Rollback</td>
<td>258</td>
</tr>
</tbody>
</table>
16. **The Web-Server Environment** ........................................ 259  
   Web Servers  259  
   Accessing the PersistenceManagerFactory  261  
   Servicing Requests  263  
   PersistenceManager per Request  264  
   PersistenceManager per Application  264  
   PersistenceManager per Transactional Request  265  
   PersistenceManager per Session  265  
   Transactions  265  
   JavaServer Pages  266  
   Struts with JDO  267  

17. **J2EE Application Servers** ........................................... 272  
   Enterprise JavaBeans Architecture  273  
   Stateless Session Beans  274  
   Configuring the PersistenceManagerFactory  275  
   Stateless Session Beans with Container-Managed Transactions  278  
   Stateful Session Beans with Container-Managed Transactions  280  
   Bean-Managed Transactions  281  
   javax.transaction.UserTransaction  282  
   javax.jdo.Transaction  282  
   Stateless Session Beans with Bean-Managed Transactions  283  
   Stateful Session Beans with Bean-Managed Transactions  283  
   Message-Driven Beans  283  
   Persistent Entities and JDO  284  
   Local Persistent Storage  284  
   Remote Persistent Storage  286  

   A. **Lifecycle States and Transitions** .............................. 293  
   B. **JDO Metadata DTD** ............................................. 296  
   C. **JDO Interfaces and Exception Classes** .................... 298  
   D. **JDO Query Language BNF** ................................ 308  
   E. **Source Code for Examples** ................................ 313  

**Index** ................................................................. 345
Java Data Objects (JDO) is an important innovation for the Java platform. At a time when developers were using JDBC almost exclusively for database access, and expert groups from major enterprise vendors were devising the much-touted Enterprise Java Beans APIs for entity beans and container-managed persistence, Craig Russell and David Jordan had the courage to take a different course. With a handful of others, they looked for a simpler way to provide persistence in the Java platform, something that would be both natural and convenient for programmers. This book describes the result of their work: JDO.

The key, unique idea behind JDO is to provide database persistence in Java with a minimum of extra stuff for the programmer to do. The programmer doesn’t need to learn SQL, doesn’t need to tediously copy data into and out of their Java objects using JDBC calls, and can use Java classes, fields, and references in a way that is natural to them, without lots of extra method calls and coding that is extraneous to the programmer’s focus and intent. Even queries can be written using Java predicates instead of SQL. In other words, the programmer just writes Java; the persistence part is automatic.

In addition to this transparent persistence, code written to JDO benefits from binary compatibility across implementations on different datastores. JDO can be used with an object/relational mapping, in which JDBC calls are generated automatically to map the data between Java objects and existing relational databases. Alternatively, the JDO objects can be stored directly in file pages, providing the functionality and performance of an object-oriented database.

The hard work on JDO paid off: the idea of transparent persistence has proven quite popular. JDO has its own community web site, JDOCentral.com, and on enterprise Java discussion sites such as TheServerSide.com, developers praise the simplicity and utility of JDO. Many developers use JDO as a replacement for entity beans, by using data objects from within session beans. Others use JDO as a convenient high-level replacement for JDBC calls in JSP pages and other Java code. JDO has come a long way from the JDBC interface I defined in 1995 with Graham Hamilton, and JDO is quite valuable in conjunction with J2EE.
I can’t think of two individuals better qualified to write a book about JDO. Craig is the specification lead for the JDO expert group, and Dave was one of the most active members of that group. But their qualifications go far beyond that, and JDO was well designed as a result of those qualifications. Both have over a decade of experience with issues in programming language persistence, including subtle transaction semantics, different persistence models, relationships between objects, caching performance, interactions between transient and persistent objects, and programming convenience in practice. Both had extensive experience with C++ persistence before they applied their experience to Java. Both were key members of the Object Data Management Group (http://www.odmg.org) for years. And, most importantly, both were developers who appreciated and needed the functionality that JDO provides.

Craig and Dave have put together a thorough, readable, and useful book. I hope you enjoy it as much as I did.

—Rick Cattell, Deputy Software CTO
Sun Microsystems, February 16, 2003
Preface

JDO provides transparent persistence of your Java object models in transactional datastores. It allows you to define your object model using all the capabilities provided in Java and it handles the mapping of that data to a variety of underlying datastores. You do not need to learn and understand a different data-modeling language like SQL. You will discover that JDO is very easy to use. Many development organizations are discovering the significant development productivity advantages that can be realized by using JDO.

Who Should Read This Book?

If you are a Java programmer who writes software that needs to store data beyond the duration of a single Java Virtual Machine (JVM) context, then you should read this book. We assume that you already know Java. But you don’t need to have a lot of knowledge of databases, because JDO insulates you from needing to know much about them.

Many Java developers have been using Java Database Connectivity (JDBC) to store their data in a database. JDBC requires that you learn SQL. When you interact with a database via JDBC, you must view your information model from the perspective of the relational data model, which is very different from Java. Many developers never attain the advantages of object-oriented programming because they never define an object model for their persistent data. Most of the application software becomes very procedural-like code that manages data in the tables of the relational data model.

With JDO, Java becomes your data model and you only need to deal with instances of your classes when interacting with the database. Having just the single data model of Java as the basis of your data management simplifies your development task considerably.
About the Authors

David Jordan founded Object Identity, Inc. in 2001 to provide JDO consulting services. He became interested in the integration of object type systems and databases while earning his M.S. in Computer Science in the early 1980s. At Bell Labs in 1985, he initiated the development of the first C++ object database. He has developed a variety of applications using C++ object models on top of network, relational, and object database systems. Bell Labs appointed him a Distinguished Member of Technical Staff in 1990 for his contributions in object and database technologies.

The Object Data Management Group (ODMG) asked him to serve as their C++ editor in 1993. He served in that role until 2000 and was appointed their Java editor in 1998. He coedited four books published by the ODMG. David served as a C++ Report columnist describing the ODMG standard from 1994 to 1996, at which point he stopped to write his first book, *C++ Object Databases*.

David started using Java in late 1995. Upon his selection as ODMG’s Java editor, he became a columnist for *Java Report*, initially covering ODMG and then JDO. David was part of the small group that initiated JSR-12, and he became one of the initial members of the JDO expert group. The JDO specification has a special acknowledgment of David’s contributions to the JDO standard. When the JDOcentral.com community web site was formed, David was selected to serve as its moderator. David can be reached at david.jordan@objectidentity.com.

Craig Russell studied applied mathematics at Harvard University (B.A. ’70–’71) and acquired practical experience working as a technician repairing mainframes with the computing power of a 1997 model laptop. His early experiences in distributed computing included building applications with CICS and DL/I and designing heterogeneous network server implementations for file, print, and communications sharing among personal computers. Craig later dealt with issues of persistent object interoperability among Smalltalk, C++, and Java and integrating object and relational databases using X/OPEN XA protocols and commercial transaction processing systems.

Craig served as the Java Chair of the Object Data Management Group and played a key role in the development of the ODMG 3.0 Java binding. For the next standard for database access from Java, the ODMG decided to support the Java Community Process as the delivery vehicle. With support from all major relational database and middleware vendors, Java Specification Request 12, Java Data Objects, was proposed in May 1999 and released in April 2002.

Concurrent with the development of the JDO standard, Craig’s primary responsibility was to lead the implementation of the object-relational database engine for several Sun products, including Java Blend and Forte for Java Transparent Persistence. Craig currently serves as chief architect of Container Managed Persistence implementation of Sun ONE Application Server, where he deals with the reality of implementing the high-performance, specification-compliant version of the object-relational database component of the J2EE application server.
Colophon

Our look is the result of reader comments, our own experimentation, and feedback from distribution channels. Distinctive covers complement our distinctive approach to technical topics, breathing personality and life into potentially dry subjects.

The animal on the cover of *Java Data Objects* is a bilby (*Macrotis lagotis*), also known as a ninu, dalgyte, pinky, or rabbit-eared bandicoot. Bilbies are rabbit-sized marsupials with silky, blue-gray fur; long, pointed snouts; large, rabbit-like ears; and long, black tails with white tips. This strange combination of traits may appear awkward, but its delicate and cute features have actually made the bilby one of Australia’s most attractive and celebrated mammals. For many Australians, the Easter Bilby has even replaced the rabbit as the popular Easter icon.

Bilbies have adapted well to the hot, arid climates they now habitate. Their long, slender tongues help them eat a diet of seeds, insects, bulbs, fruit, and fungi. Bilbies have well-developed forearms and long claws, which they use to dig the deep, spiraling burrows in which they live. Bilbies are strictly nocturnal, and during the day they plug the entrances to their holes with soil to protect them from extreme temperatures. Because bilbies are solitary animals, burrows usually have a single opening and a single occupant, though females often live with their young. Like other marsupials, females have a backward-opening pouch with eight teats, used to carry and protect their young for about 80 days. Bilbies usually have no more than two young at a time.

Once common throughout Australia, disease, agriculture clearing, spreading of the fox and feral cat, and the control campaign against the destructive rabbit (which was often unfairly grouped with the innocent bilby it resembles) have limited bilbies’ habitats to isolated populations in Western Australia, the Northern Territory, and southwestern Queensland. Bilbies are now listed as endangered species by many Australian and international conservation groups.

Brian Sawyer was the production editor and copyeditor for *Java Data Objects*. Colleen Gorman was the proofreader. Genevieve d’Entremont and Claire Cloutier provided quality control. David Jordan and Craig Russell wrote the index, with the assistance of Reg Aubry.

Hanna Dyer designed the cover of this book, based on a series design by Edie Freedman. The cover image is a 19th-century engraving from *Animate Creation, Vol. II*. Emma Colby produced the cover layout with QuarkXPress 4.1 using Adobe’s ITC Garamond font.

David Futato designed the interior layout. Andrew Savikas prepared this book in FrameMaker 5.5.6. The text font is Linotype Birka, and the heading font is Adobe Myriad Condensed. The code font is a modified version of LucasFont’s TheSans Mono Condensed, designed by Luc(as) de Groot with modifications suggested by David Jordan. The illustrations that appear in the book were produced by Robert Romano and Jessamyn Read using FreeHand 9 and Adobe Photoshop 6. This colophon was written by Brian Sawyer.