Hibernate is an amazing piece of software. With a little experience and the power of Java 5
annotations, you can build a complex database-backed system with disturbing ease. Once
you have built a system using Hibernate, you will never want to go back to the traditional
approaches.
While Hibernate is incredibly powerful, it presents a steep learning curve when you first
encounter it—steep learning curves are actually a good thing, as they impart profound insight
once you have scaled them. Yet gaining that insight takes some perseverance and assistance.
Our aim in this book is to help you up that learning curve by presenting you with the minimal
requirements of a discrete Hibernate application, explaining the basis of those requirements,
and walking you through an example application built according to them. We then provide additional
material to be digested once the fundamentals are firmly understood. Throughout, we
provide examples rather than relying upon pure discourse.
We hope that you will continue to find this book useful as a reference text long after you
have become an expert on the subject.
Who This Book Is For
This book assumes a good understanding of Java fundamentals and some familiarity with database
programming using the Java Database Connectivity (JDBC) API. We don’t expect you to
know anything about Hibernate—but if you buy this book, it will probably be because you have
some exposure to the painful process of building a large database-based system.
All of our examples use open source software—primarily the Hibernate API itself—so you
will not need to purchase any software to get started with Hibernate development.
This book is not an academic text. Our focus is instead on providing extensive examples
and taking a pragmatic approach to the technology that it covers.
To true newcomers to the Hibernate API, we recommend that you read at least the first
three chapters in order before diving into the juicy subjects of later chapters. Very experienced
developers or those with experience with tools similar to Hibernate will want to skim through
the latter half of the book for interesting chapters. Readers familiar with Hibernate will want
to turn to the appendixes for discussion of more arcane topics.
How This Book Is Structured
This book is informally divided into three parts. Chapters 1 through 8 describe the fundamentals
of Hibernate, including configuration, the creation of mapping files, and the basic APIs.
Chapters 9 through 11 then describe the use of queries, criteria, and filters to access the
persistent information in more sophisticated ways.
xix
Finally, the appendixes discuss features that you will use less often, or that are peripheral
to the core Hibernate functionality. The following list describes more fully the contents of
each chapter:
Chapter 1 outlines the purpose of persistence tools and presents excerpts from a simple
example application to show how Hibernate can be applied. It also introduces core terminology
and concepts.
Chapter 2 discusses the fundamentals of configuring a Hibernate application. It presents
the basic architecture of Hibernate and discusses how a Hibernate application is integrated
into an application.
Chapter 3 presents the example application from Chapter 1 in its entirety, walking you
through the complete process of creating and running the application. It then looks at
a slightly more complex example and introduces the notion of generating the database
schema directly from the mapping files.
Chapter 4 covers the Hibernate life cycle in depth. It discusses the life cycle in the context
of the methods available on the core interfaces. It also introduces key terminology and
discusses the need for cascading and lazy loading.
Chapter 5 explains why mapping information must be retained by Hibernate, and
demonstrates the various types of associations that can be represented by a relational
database. It briefly discusses the other information that can be maintained within a
Hibernate mapping.
Chapter 6 explains how Hibernate lets you use the Java 5 Annotations feature to represent
mapping information. It provides detailed examples for the most important annotations,
and discusses the distinctions between the standard EJB 3 annotations and the proprietary
Hibernate 3 ones.
Chapter 7 explains how the XML-based mapping files can be used to represent mapping
information in Hibernate. It provides examples for all of the most common mapping
types and reference notes for the more obscure ones.
Chapter 8 revisits the Hibernate Session object in detail, explaining the various methods
that it provides. The chapter also discusses the use of transactions, locking, and caching,
and how to use Hibernate in a multithreaded environment.
Chapter 9 discusses how Hibernate can be used to make sophisticated queries against the
underlying relational database using the built-in Hibernate Query Language (HQL).
Chapter 10 introduces the Criteria API, which is a programmatic analog of the query language
discussed in Chapter 9.
Chapter 11 discusses how the filter API can be used to restrict the results of the queries
introduced in Chapters 9 and 10.
Appendix A presents a large number of peripheral features that do not warrant more
extensive coverage in a beginners’ text. Basic discussion is given, with examples, of the
use of the Hibernate EntityManager and EJB 3, the support for versioning and optimistic
locking, the provision for persisting and retrieving Dom4J document models directly, the
support for persisting and retrieving maps of information, and some of the obscure limitations
of Hibernate and various ways that these can be worked around. It also discusses
the use of events and interceptors.
Appendix B discusses how the Hibernate Tools toolset can be used to enhance development
with the Eclipse development environment and the Ant build tool. It also explains
how the Ant code-generation tasks can be customized.
Appendix C discusses how Hibernate can be integrated into the Spring API. The integration
of Hibernate as the persistence layer of a Spring application is complex, so we present
a working example, including the entire bean definition file, with discussions of the appropriate
way to manage the session in the Spring MVC environment, and how Spring can
enforce the proper transactional boundaries when using Hibernate.
Appendix D discusses some topics of interest to developers who are working with a preexisting
base of code that was built using version 2 of Hibernate. We present the various
approaches to coexisting with Hibernate 3 code and to migrating a Hibernate 2 code base
to the Hibernate 3 API.
Downloading the Code
The source code for this book is available to readers from www.apress.com, in the Source
Code/Download section. Please feel free to visit the Apress web site and download all the
code from there.
Contacting the Authors
We welcome feedback from our readers. If you have any queries or suggestions about this
book, or technical questions about Hibernate, or if you just want to share a really good joke,
you can e-mail Dave Minter at dave@paperstack.com and Jeff Linwood at jlinwood@gmail.com.
Most significant development projects involve a relational database. The mainstay of most
commercial applications is the large-scale storage of ordered information, such as catalogs,
customer lists, contract details, published text, and architectural designs.
With the advent of the World Wide Web, the demand for databases has increased. Though
they may not know it, the customers of online bookshops and newspapers are using databases.
Somewhere in the guts of the application a database is being queried and a response is offered.
While the demand for such applications has grown, their creation has not become noticeably
simpler. Some standardization has occurred—the most successful being the Enterprise
JavaBeans (EJB) standard of Java 2 Enterprise Edition (J2EE), which provides for containerand
bean-managed persistence of entity bean classes. Unfortunately, this and other persistence
models all suffer to one degree or another from the mismatch between the relational
model and the object-oriented model. In short, database persistence is difficult.
There are solutions for which EJBs are appropriate, some for which some sort of objectrelational
mapping (ORM) like Hibernate is appropriate, and some for which the traditional
approach of direct access via the Java Database Connectivity (JDBC) API is appropriate. We
think that Hibernate represents a good first choice, as it does not preclude the simultaneous
use of these alternative approaches.
To illustrate some of Hibernate’s strengths, in this chapter we will show you a brief example
using Hibernate and contrast this with the traditional JDBC approach.
Plain Old Java Objects (POJOs)
In our ideal world, it would be trivial to take any Java object and persist it to the database. No
special coding would be required to achieve this, no performance penalty would ensue, and
the result would be totally portable.
In this ideal world, we would perhaps perform such an operation in a manner like that
shown in Listing 1-1.
Listing 1-1. A Rose-Tinted View of Object Persistence
POJO pojo = new POJO();
ORMSolution magic = ORMSolution.getInstance();
magic.save(pojo);
There would be no nasty surprises, no additional work to correlate the class with tables in
the database, and no performance problems
Hibernate As a Persistence Solution
Hibernate addresses a lot of these points, or alleviates some of the pain where it can’t, so we’ll
address the points in turn.
Hibernate does not require you to map one POJO to one table. A POJO can be constructed
out of a selection of table columns, or several POJOs can be persisted into a single table.
Hibernate directly supports inheritance relationships and the various other relationships
between classes.
Though there is some performance overhead while Hibernate starts up and processes its
configuration files, it is generally perceived as being a fast tool. This is very hard to quantify, and,
to some extent, the poor reputation of entity beans may have been earned less from their own
faults than from the mistakes of those designing and deploying such applications. As with all
performance questions, you should carry out tests rather than relying on anecdotal evidence.
In Hibernate it is possible, but not necessary, to specify the mappings at deployment
time. The EJB solution ensures portability of applications across environments, but the Hibernate
approach tends to reduce the pain of deploying an application to a new environment.
Hibernate persistence has no requirement for a J2EE application server or any other special
environment. It is, therefore, a much more suitable solution for stand-alone applications,
client-side application storage, and other environments in which a J2EE server is not immediately
available.
Hibernate uses POJOs that can very easily and naturally be generalized for use in other
applications. There is no direct dependency upon the Hibernate libraries, so POJOs can be put
to any use that does not require persistence; or they can be persisted using any other “POJOfriendly”
mechanism.
Hibernate presents no problems when handling serializable POJOs.
There is a very large body of preexisting code. Any Java object capable of being persisted
to a database is a candidate for Hibernate persistence. Therefore, Hibernate is a natural
replacement for ad hoc solutions, or as the persistence engine for an application that has not
yet had database persistence incorporated into it. Furthermore, by choosing Hibernate persistence,
you are not tying yourself to any particular design decisions for the business objects
in your application.
Hibernate is an amazing piece of software. With a little experience and the power of Java 5
annotations, you can build a complex database-backed system with disturbing ease. Once
you have built a system using Hibernate, you will never want to go back to the traditional
approaches.
While Hibernate is incredibly powerful, it presents a steep learning curve when you first
encounter it—steep learning curves are actually a good thing, as they impart profound insight
once you have scaled them. Yet gaining that insight takes some perseverance and assistance.
Our aim in this book is to help you up that learning curve by presenting you with the minimal
requirements of a discrete Hibernate application, explaining the basis of those requirements,
and walking you through an example application built according to them. We then provide additional
material to be digested once the fundamentals are firmly understood. Throughout, we
provide examples rather than relying upon pure discourse.
We hope that you will continue to find this book useful as a reference text long after you
have become an expert on the subject.
Who This Book Is For
This book assumes a good understanding of Java fundamentals and some familiarity with database
programming using the Java Database Connectivity (JDBC) API. We don’t expect you to
know anything about Hibernate—but if you buy this book, it will probably be because you have
some exposure to the painful process of building a large database-based system.
All of our examples use open source software—primarily the Hibernate API itself—so you
will not need to purchase any software to get started with Hibernate development.
This book is not an academic text. Our focus is instead on providing extensive examples
and taking a pragmatic approach to the technology that it covers.
To true newcomers to the Hibernate API, we recommend that you read at least the first
three chapters in order before diving into the juicy subjects of later chapters. Very experienced
developers or those with experience with tools similar to Hibernate will want to skim through
the latter half of the book for interesting chapters. Readers familiar with Hibernate will want
to turn to the appendixes for discussion of more arcane topics.
How This Book Is Structured
This book is informally divided into three parts. Chapters 1 through 8 describe the fundamentals
of Hibernate, including configuration, the creation of mapping files, and the basic APIs.
Chapters 9 through 11 then describe the use of queries, criteria, and filters to access the
persistent information in more sophisticated ways.
xix
Finally, the appendixes discuss features that you will use less often, or that are peripheral
to the core Hibernate functionality. The following list describes more fully the contents of
each chapter:
Chapter 1 outlines the purpose of persistence tools and presents excerpts from a simple
example application to show how Hibernate can be applied. It also introduces core terminology
and concepts.
Chapter 2 discusses the fundamentals of configuring a Hibernate application. It presents
the basic architecture of Hibernate and discusses how a Hibernate application is integrated
into an application.
Chapter 3 presents the example application from Chapter 1 in its entirety, walking you
through the complete process of creating and running the application. It then looks at
a slightly more complex example and introduces the notion of generating the database
schema directly from the mapping files.
Chapter 4 covers the Hibernate life cycle in depth. It discusses the life cycle in the context
of the methods available on the core interfaces. It also introduces key terminology and
discusses the need for cascading and lazy loading.
Chapter 5 explains why mapping information must be retained by Hibernate, and
demonstrates the various types of associations that can be represented by a relational
database. It briefly discusses the other information that can be maintained within a
Hibernate mapping.
Chapter 6 explains how Hibernate lets you use the Java 5 Annotations feature to represent
mapping information. It provides detailed examples for the most important annotations,
and discusses the distinctions between the standard EJB 3 annotations and the proprietary
Hibernate 3 ones.
Chapter 7 explains how the XML-based mapping files can be used to represent mapping
information in Hibernate. It provides examples for all of the most common mapping
types and reference notes for the more obscure ones.
Chapter 8 revisits the Hibernate Session object in detail, explaining the various methods
that it provides. The chapter also discusses the use of transactions, locking, and caching,
and how to use Hibernate in a multithreaded environment.
Chapter 9 discusses how Hibernate can be used to make sophisticated queries against the
underlying relational database using the built-in Hibernate Query Language (HQL).
Chapter 10 introduces the Criteria API, which is a programmatic analog of the query language
discussed in Chapter 9.
Chapter 11 discusses how the filter API can be used to restrict the results of the queries
introduced in Chapters 9 and 10.
Appendix A presents a large number of peripheral features that do not warrant more
extensive coverage in a beginners’ text. Basic discussion is given, with examples, of the
use of the Hibernate EntityManager and EJB 3, the support for versioning and optimistic
locking, the provision for persisting and retrieving Dom4J document models directly, the
support for persisting and retrieving maps of information, and some of the obscure limitations
of Hibernate and various ways that these can be worked around. It also discusses
the use of events and interceptors.
Appendix B discusses how the Hibernate Tools toolset can be used to enhance development
with the Eclipse development environment and the Ant build tool. It also explains
how the Ant code-generation tasks can be customized.
Appendix C discusses how Hibernate can be integrated into the Spring API. The integration
of Hibernate as the persistence layer of a Spring application is complex, so we present
a working example, including the entire bean definition file, with discussions of the appropriate
way to manage the session in the Spring MVC environment, and how Spring can
enforce the proper transactional boundaries when using Hibernate.
Appendix D discusses some topics of interest to developers who are working with a preexisting
base of code that was built using version 2 of Hibernate. We present the various
approaches to coexisting with Hibernate 3 code and to migrating a Hibernate 2 code base
to the Hibernate 3 API.
Downloading the Code
The source code for this book is available to readers from www.apress.com, in the Source
Code/Download section. Please feel free to visit the Apress web site and download all the
code from there.
Contacting the Authors
We welcome feedback from our readers. If you have any queries or suggestions about this
book, or technical questions about Hibernate, or if you just want to share a really good joke,
you can e-mail Dave Minter at dave@paperstack.com and Jeff Linwood at jlinwood@gmail.com.
Most significant development projects involve a relational database. The mainstay of most
commercial applications is the large-scale storage of ordered information, such as catalogs,
customer lists, contract details, published text, and architectural designs.
With the advent of the World Wide Web, the demand for databases has increased. Though
they may not know it, the customers of online bookshops and newspapers are using databases.
Somewhere in the guts of the application a database is being queried and a response is offered.
While the demand for such applications has grown, their creation has not become noticeably
simpler. Some standardization has occurred—the most successful being the Enterprise
JavaBeans (EJB) standard of Java 2 Enterprise Edition (J2EE), which provides for containerand
bean-managed persistence of entity bean classes. Unfortunately, this and other persistence
models all suffer to one degree or another from the mismatch between the relational
model and the object-oriented model. In short, database persistence is difficult.
There are solutions for which EJBs are appropriate, some for which some sort of objectrelational
mapping (ORM) like Hibernate is appropriate, and some for which the traditional
approach of direct access via the Java Database Connectivity (JDBC) API is appropriate. We
think that Hibernate represents a good first choice, as it does not preclude the simultaneous
use of these alternative approaches.
To illustrate some of Hibernate’s strengths, in this chapter we will show you a brief example
using Hibernate and contrast this with the traditional JDBC approach.
Plain Old Java Objects (POJOs)
In our ideal world, it would be trivial to take any Java object and persist it to the database. No
special coding would be required to achieve this, no performance penalty would ensue, and
the result would be totally portable.
In this ideal world, we would perhaps perform such an operation in a manner like that
shown in Listing 1-1.
Listing 1-1. A Rose-Tinted View of Object Persistence
POJO pojo = new POJO();
ORMSolution magic = ORMSolution.getInstance();
magic.save(pojo);
There would be no nasty surprises, no additional work to correlate the class with tables in
the database, and no performance problems
Hibernate As a Persistence Solution
Hibernate addresses a lot of these points, or alleviates some of the pain where it can’t, so we’ll
address the points in turn.
Hibernate does not require you to map one POJO to one table. A POJO can be constructed
out of a selection of table columns, or several POJOs can be persisted into a single table.
Hibernate directly supports inheritance relationships and the various other relationships
between classes.
Though there is some performance overhead while Hibernate starts up and processes its
configuration files, it is generally perceived as being a fast tool. This is very hard to quantify, and,
to some extent, the poor reputation of entity beans may have been earned less from their own
faults than from the mistakes of those designing and deploying such applications. As with all
performance questions, you should carry out tests rather than relying on anecdotal evidence.
In Hibernate it is possible, but not necessary, to specify the mappings at deployment
time. The EJB solution ensures portability of applications across environments, but the Hibernate
approach tends to reduce the pain of deploying an application to a new environment.
Hibernate persistence has no requirement for a J2EE application server or any other special
environment. It is, therefore, a much more suitable solution for stand-alone applications,
client-side application storage, and other environments in which a J2EE server is not immediately
available.
Hibernate uses POJOs that can very easily and naturally be generalized for use in other
applications. There is no direct dependency upon the Hibernate libraries, so POJOs can be put
to any use that does not require persistence; or they can be persisted using any other “POJOfriendly”
mechanism.
Hibernate presents no problems when handling serializable POJOs.
There is a very large body of preexisting code. Any Java object capable of being persisted
to a database is a candidate for Hibernate persistence. Therefore, Hibernate is a natural
replacement for ad hoc solutions, or as the persistence engine for an application that has not
yet had database persistence incorporated into it. Furthermore, by choosing Hibernate persistence,
you are not tying yourself to any particular design decisions for the business objects
in your application.
Hibernate is an amazing piece of software. With a little experience and the power of Java 5
annotations, you can build a complex database-backed system with disturbing ease. Once
you have built a system using Hibernate, you will never want to go back to the traditional
approaches.
While Hibernate is incredibly powerful, it presents a steep learning curve when you first
encounter it—steep learning curves are actually a good thing, as they impart profound insight
once you have scaled them. Yet gaining that insight takes some perseverance and assistance.
Our aim in this book is to help you up that learning curve by presenting you with the minimal
requirements of a discrete Hibernate application, explaining the basis of those requirements,
and walking you through an example application built according to them. We then provide additional
material to be digested once the fundamentals are firmly understood. Throughout, we
provide examples rather than relying upon pure discourse.
We hope that you will continue to find this book useful as a reference text long after you
have become an expert on the subject.
Who This Book Is For
This book assumes a good understanding of Java fundamentals and some familiarity with database
programming using the Java Database Connectivity (JDBC) API. We don’t expect you to
know anything about Hibernate—but if you buy this book, it will probably be because you have
some exposure to the painful process of building a large database-based system.
All of our examples use open source software—primarily the Hibernate API itself—so you
will not need to purchase any software to get started with Hibernate development.
This book is not an academic text. Our focus is instead on providing extensive examples
and taking a pragmatic approach to the technology that it covers.
To true newcomers to the Hibernate API, we recommend that you read at least the first
three chapters in order before diving into the juicy subjects of later chapters. Very experienced
developers or those with experience with tools similar to Hibernate will want to skim through
the latter half of the book for interesting chapters. Readers familiar with Hibernate will want
to turn to the appendixes for discussion of more arcane topics.
How This Book Is Structured
This book is informally divided into three parts. Chapters 1 through 8 describe the fundamentals
of Hibernate, including configuration, the creation of mapping files, and the basic APIs.
Chapters 9 through 11 then describe the use of queries, criteria, and filters to access the
persistent information in more sophisticated ways.
xix
Finally, the appendixes discuss features that you will use less often, or that are peripheral
to the core Hibernate functionality. The following list describes more fully the contents of
each chapter:
Chapter 1 outlines the purpose of persistence tools and presents excerpts from a simple
example application to show how Hibernate can be applied. It also introduces core terminology
and concepts.
Chapter 2 discusses the fundamentals of configuring a Hibernate application. It presents
the basic architecture of Hibernate and discusses how a Hibernate application is integrated
into an application.
Chapter 3 presents the example application from Chapter 1 in its entirety, walking you
through the complete process of creating and running the application. It then looks at
a slightly more complex example and introduces the notion of generating the database
schema directly from the mapping files.
Chapter 4 covers the Hibernate life cycle in depth. It discusses the life cycle in the context
of the methods available on the core interfaces. It also introduces key terminology and
discusses the need for cascading and lazy loading.
Chapter 5 explains why mapping information must be retained by Hibernate, and
demonstrates the various types of associations that can be represented by a relational
database. It briefly discusses the other information that can be maintained within a
Hibernate mapping.
Chapter 6 explains how Hibernate lets you use the Java 5 Annotations feature to represent
mapping information. It provides detailed examples for the most important annotations,
and discusses the distinctions between the standard EJB 3 annotations and the proprietary
Hibernate 3 ones.
Chapter 7 explains how the XML-based mapping files can be used to represent mapping
information in Hibernate. It provides examples for all of the most common mapping
types and reference notes for the more obscure ones.
Chapter 8 revisits the Hibernate Session object in detail, explaining the various methods
that it provides. The chapter also discusses the use of transactions, locking, and caching,
and how to use Hibernate in a multithreaded environment.
Chapter 9 discusses how Hibernate can be used to make sophisticated queries against the
underlying relational database using the built-in Hibernate Query Language (HQL).
Chapter 10 introduces the Criteria API, which is a programmatic analog of the query language
discussed in Chapter 9.
Chapter 11 discusses how the filter API can be used to restrict the results of the queries
introduced in Chapters 9 and 10.
Appendix A presents a large number of peripheral features that do not warrant more
extensive coverage in a beginners’ text. Basic discussion is given, with examples, of the
use of the Hibernate EntityManager and EJB 3, the support for versioning and optimistic
locking, the provision for persisting and retrieving Dom4J document models directly, the
support for persisting and retrieving maps of information, and some of the obscure limitations
of Hibernate and various ways that these can be worked around. It also discusses
the use of events and interceptors.
Appendix B discusses how the Hibernate Tools toolset can be used to enhance development
with the Eclipse development environment and the Ant build tool. It also explains
how the Ant code-generation tasks can be customized.
Appendix C discusses how Hibernate can be integrated into the Spring API. The integration
of Hibernate as the persistence layer of a Spring application is complex, so we present
a working example, including the entire bean definition file, with discussions of the appropriate
way to manage the session in the Spring MVC environment, and how Spring can
enforce the proper transactional boundaries when using Hibernate.
Appendix D discusses some topics of interest to developers who are working with a preexisting
base of code that was built using version 2 of Hibernate. We present the various
approaches to coexisting with Hibernate 3 code and to migrating a Hibernate 2 code base
to the Hibernate 3 API.
Downloading the Code
The source code for this book is available to readers from www.apress.com, in the Source
Code/Download section. Please feel free to visit the Apress web site and download all the
code from there.
Contacting the Authors
We welcome feedback from our readers. If you have any queries or suggestions about this
book, or technical questions about Hibernate, or if you just want to share a really good joke,
you can e-mail Dave Minter at dave@paperstack.com and Jeff Linwood at jlinwood@gmail.com.
Most significant development projects involve a relational database. The mainstay of most
commercial applications is the large-scale storage of ordered information, such as catalogs,
customer lists, contract details, published text, and architectural designs.
With the advent of the World Wide Web, the demand for databases has increased. Though
they may not know it, the customers of online bookshops and newspapers are using databases.
Somewhere in the guts of the application a database is being queried and a response is offered.
While the demand for such applications has grown, their creation has not become noticeably
simpler. Some standardization has occurred—the most successful being the Enterprise
JavaBeans (EJB) standard of Java 2 Enterprise Edition (J2EE), which provides for containerand
bean-managed persistence of entity bean classes. Unfortunately, this and other persistence
models all suffer to one degree or another from the mismatch between the relational
model and the object-oriented model. In short, database persistence is difficult.
There are solutions for which EJBs are appropriate, some for which some sort of objectrelational
mapping (ORM) like Hibernate is appropriate, and some for which the traditional
approach of direct access via the Java Database Connectivity (JDBC) API is appropriate. We
think that Hibernate represents a good first choice, as it does not preclude the simultaneous
use of these alternative approaches.
To illustrate some of Hibernate’s strengths, in this chapter we will show you a brief example
using Hibernate and contrast this with the traditional JDBC approach.
Plain Old Java Objects (POJOs)
In our ideal world, it would be trivial to take any Java object and persist it to the database. No
special coding would be required to achieve this, no performance penalty would ensue, and
the result would be totally portable.
In this ideal world, we would perhaps perform such an operation in a manner like that
shown in Listing 1-1.
Listing 1-1. A Rose-Tinted View of Object Persistence
POJO pojo = new POJO();
ORMSolution magic = ORMSolution.getInstance();
magic.save(pojo);
There would be no nasty surprises, no additional work to correlate the class with tables in
the database, and no performance problems
Hibernate As a Persistence Solution
Hibernate addresses a lot of these points, or alleviates some of the pain where it can’t, so we’ll
address the points in turn.
Hibernate does not require you to map one POJO to one table. A POJO can be constructed
out of a selection of table columns, or several POJOs can be persisted into a single table.
Hibernate directly supports inheritance relationships and the various other relationships
between classes.
Though there is some performance overhead while Hibernate starts up and processes its
configuration files, it is generally perceived as being a fast tool. This is very hard to quantify, and,
to some extent, the poor reputation of entity beans may have been earned less from their own
faults than from the mistakes of those designing and deploying such applications. As with all
performance questions, you should carry out tests rather than relying on anecdotal evidence.
In Hibernate it is possible, but not necessary, to specify the mappings at deployment
time. The EJB solution ensures portability of applications across environments, but the Hibernate
approach tends to reduce the pain of deploying an application to a new environment.
Hibernate persistence has no requirement for a J2EE application server or any other special
environment. It is, therefore, a much more suitable solution for stand-alone applications,
client-side application storage, and other environments in which a J2EE server is not immediately
available.
Hibernate uses POJOs that can very easily and naturally be generalized for use in other
applications. There is no direct dependency upon the Hibernate libraries, so POJOs can be put
to any use that does not require persistence; or they can be persisted using any other “POJOfriendly”
mechanism.
Hibernate presents no problems when handling serializable POJOs.
There is a very large body of preexisting code. Any Java object capable of being persisted
to a database is a candidate for Hibernate persistence. Therefore, Hibernate is a natural
replacement for ad hoc solutions, or as the persistence engine for an application that has not
yet had database persistence incorporated into it. Furthermore, by choosing Hibernate persistence,
you are not tying yourself to any particular design decisions for the business objects
in your application.
Hibernate is an amazing piece of software. With a little experience and the power of Java 5
annotations, you can build a complex database-backed system with disturbing ease. Once
you have built a system using Hibernate, you will never want to go back to the traditional
approaches.
While Hibernate is incredibly powerful, it presents a steep learning curve when you first
encounter it—steep learning curves are actually a good thing, as they impart profound insight
once you have scaled them. Yet gaining that insight takes some perseverance and assistance.
Our aim in this book is to help you up that learning curve by presenting you with the minimal
requirements of a discrete Hibernate application, explaining the basis of those requirements,
and walking you through an example application built according to them. We then provide additional
material to be digested once the fundamentals are firmly understood. Throughout, we
provide examples rather than relying upon pure discourse.
We hope that you will continue to find this book useful as a reference text long after you
have become an expert on the subject.
Who This Book Is For
This book assumes a good understanding of Java fundamentals and some familiarity with database
programming using the Java Database Connectivity (JDBC) API. We don’t expect you to
know anything about Hibernate—but if you buy this book, it will probably be because you have
some exposure to the painful process of building a large database-based system.
All of our examples use open source software—primarily the Hibernate API itself—so you
will not need to purchase any software to get started with Hibernate development.
This book is not an academic text. Our focus is instead on providing extensive examples
and taking a pragmatic approach to the technology that it covers.
To true newcomers to the Hibernate API, we recommend that you read at least the first
three chapters in order before diving into the juicy subjects of later chapters. Very experienced
developers or those with experience with tools similar to Hibernate will want to skim through
the latter half of the book for interesting chapters. Readers familiar with Hibernate will want
to turn to the appendixes for discussion of more arcane topics.
How This Book Is Structured
This book is informally divided into three parts. Chapters 1 through 8 describe the fundamentals
of Hibernate, including configuration, the creation of mapping files, and the basic APIs.
Chapters 9 through 11 then describe the use of queries, criteria, and filters to access the
persistent information in more sophisticated ways.
xix
Finally, the appendixes discuss features that you will use less often, or that are peripheral
to the core Hibernate functionality. The following list describes more fully the contents of
each chapter:
Chapter 1 outlines the purpose of persistence tools and presents excerpts from a simple
example application to show how Hibernate can be applied. It also introduces core terminology
and concepts.
Chapter 2 discusses the fundamentals of configuring a Hibernate application. It presents
the basic architecture of Hibernate and discusses how a Hibernate application is integrated
into an application.
Chapter 3 presents the example application from Chapter 1 in its entirety, walking you
through the complete process of creating and running the application. It then looks at
a slightly more complex example and introduces the notion of generating the database
schema directly from the mapping files.
Chapter 4 covers the Hibernate life cycle in depth. It discusses the life cycle in the context
of the methods available on the core interfaces. It also introduces key terminology and
discusses the need for cascading and lazy loading.
Chapter 5 explains why mapping information must be retained by Hibernate, and
demonstrates the various types of associations that can be represented by a relational
database. It briefly discusses the other information that can be maintained within a
Hibernate mapping.
Chapter 6 explains how Hibernate lets you use the Java 5 Annotations feature to represent
mapping information. It provides detailed examples for the most important annotations,
and discusses the distinctions between the standard EJB 3 annotations and the proprietary
Hibernate 3 ones.
Chapter 7 explains how the XML-based mapping files can be used to represent mapping
information in Hibernate. It provides examples for all of the most common mapping
types and reference notes for the more obscure ones.
Chapter 8 revisits the Hibernate Session object in detail, explaining the various methods
that it provides. The chapter also discusses the use of transactions, locking, and caching,
and how to use Hibernate in a multithreaded environment.
Chapter 9 discusses how Hibernate can be used to make sophisticated queries against the
underlying relational database using the built-in Hibernate Query Language (HQL).
Chapter 10 introduces the Criteria API, which is a programmatic analog of the query language
discussed in Chapter 9.
Chapter 11 discusses how the filter API can be used to restrict the results of the queries
introduced in Chapters 9 and 10.
Appendix A presents a large number of peripheral features that do not warrant more
extensive coverage in a beginners’ text. Basic discussion is given, with examples, of the
use of the Hibernate EntityManager and EJB 3, the support for versioning and optimistic
locking, the provision for persisting and retrieving Dom4J document models directly, the
support for persisting and retrieving maps of information, and some of the obscure limitations
of Hibernate and various ways that these can be worked around. It also discusses
the use of events and interceptors.
Appendix B discusses how the Hibernate Tools toolset can be used to enhance development
with the Eclipse development environment and the Ant build tool. It also explains
how the Ant code-generation tasks can be customized.
Appendix C discusses how Hibernate can be integrated into the Spring API. The integration
of Hibernate as the persistence layer of a Spring application is complex, so we present
a working example, including the entire bean definition file, with discussions of the appropriate
way to manage the session in the Spring MVC environment, and how Spring can
enforce the proper transactional boundaries when using Hibernate.
Appendix D discusses some topics of interest to developers who are working with a preexisting
base of code that was built using version 2 of Hibernate. We present the various
approaches to coexisting with Hibernate 3 code and to migrating a Hibernate 2 code base
to the Hibernate 3 API.
Downloading the Code
The source code for this book is available to readers from www.apress.com, in the Source
Code/Download section. Please feel free to visit the Apress web site and download all the
code from there.
Contacting the Authors
We welcome feedback from our readers. If you have any queries or suggestions about this
book, or technical questions about Hibernate, or if you just want to share a really good joke,
you can e-mail Dave Minter at dave@paperstack.com and Jeff Linwood at jlinwood@gmail.com.
Most significant development projects involve a relational database. The mainstay of most
commercial applications is the large-scale storage of ordered information, such as catalogs,
customer lists, contract details, published text, and architectural designs.
With the advent of the World Wide Web, the demand for databases has increased. Though
they may not know it, the customers of online bookshops and newspapers are using databases.
Somewhere in the guts of the application a database is being queried and a response is offered.
While the demand for such applications has grown, their creation has not become noticeably
simpler. Some standardization has occurred—the most successful being the Enterprise
JavaBeans (EJB) standard of Java 2 Enterprise Edition (J2EE), which provides for containerand
bean-managed persistence of entity bean classes. Unfortunately, this and other persistence
models all suffer to one degree or another from the mismatch between the relational
model and the object-oriented model. In short, database persistence is difficult.
There are solutions for which EJBs are appropriate, some for which some sort of objectrelational
mapping (ORM) like Hibernate is appropriate, and some for which the traditional
approach of direct access via the Java Database Connectivity (JDBC) API is appropriate. We
think that Hibernate represents a good first choice, as it does not preclude the simultaneous
use of these alternative approaches.
To illustrate some of Hibernate’s strengths, in this chapter we will show you a brief example
using Hibernate and contrast this with the traditional JDBC approach.
Plain Old Java Objects (POJOs)
In our ideal world, it would be trivial to take any Java object and persist it to the database. No
special coding would be required to achieve this, no performance penalty would ensue, and
the result would be totally portable.
In this ideal world, we would perhaps perform such an operation in a manner like that
shown in Listing 1-1.
Listing 1-1. A Rose-Tinted View of Object Persistence
POJO pojo = new POJO();
ORMSolution magic = ORMSolution.getInstance();
magic.save(pojo);
There would be no nasty surprises, no additional work to correlate the class with tables in
the database, and no performance problems
Hibernate As a Persistence Solution
Hibernate addresses a lot of these points, or alleviates some of the pain where it can’t, so we’ll
address the points in turn.
Hibernate does not require you to map one POJO to one table. A POJO can be constructed
out of a selection of table columns, or several POJOs can be persisted into a single table.
Hibernate directly supports inheritance relationships and the various other relationships
between classes.
Though there is some performance overhead while Hibernate starts up and processes its
configuration files, it is generally perceived as being a fast tool. This is very hard to quantify, and,
to some extent, the poor reputation of entity beans may have been earned less from their own
faults than from the mistakes of those designing and deploying such applications. As with all
performance questions, you should carry out tests rather than relying on anecdotal evidence.
In Hibernate it is possible, but not necessary, to specify the mappings at deployment
time. The EJB solution ensures portability of applications across environments, but the Hibernate
approach tends to reduce the pain of deploying an application to a new environment.
Hibernate persistence has no requirement for a J2EE application server or any other special
environment. It is, therefore, a much more suitable solution for stand-alone applications,
client-side application storage, and other environments in which a J2EE server is not immediately
available.
Hibernate uses POJOs that can very easily and naturally be generalized for use in other
applications. There is no direct dependency upon the Hibernate libraries, so POJOs can be put
to any use that does not require persistence; or they can be persisted using any other “POJOfriendly”
mechanism.
Hibernate presents no problems when handling serializable POJOs.
There is a very large body of preexisting code. Any Java object capable of being persisted
to a database is a candidate for Hibernate persistence. Therefore, Hibernate is a natural
replacement for ad hoc solutions, or as the persistence engine for an application that has not
yet had database persistence incorporated into it. Furthermore, by choosing Hibernate persistence,
you are not tying yourself to any particular design decisions for the business objects
in your application.
Hibernate is an amazing piece of software. With a little experience and the power of Java 5
annotations, you can build a complex database-backed system with disturbing ease. Once
you have built a system using Hibernate, you will never want to go back to the traditional
approaches.
While Hibernate is incredibly powerful, it presents a steep learning curve when you first
encounter it—steep learning curves are actually a good thing, as they impart profound insight
once you have scaled them. Yet gaining that insight takes some perseverance and assistance.
Our aim in this book is to help you up that learning curve by presenting you with the minimal
requirements of a discrete Hibernate application, explaining the basis of those requirements,
and walking you through an example application built according to them. We then provide additional
material to be digested once the fundamentals are firmly understood. Throughout, we
provide examples rather than relying upon pure discourse.
We hope that you will continue to find this book useful as a reference text long after you
have become an expert on the subject.
Who This Book Is For
This book assumes a good understanding of Java fundamentals and some familiarity with database
programming using the Java Database Connectivity (JDBC) API. We don’t expect you to
know anything about Hibernate—but if you buy this book, it will probably be because you have
some exposure to the painful process of building a large database-based system.
All of our examples use open source software—primarily the Hibernate API itself—so you
will not need to purchase any software to get started with Hibernate development.
This book is not an academic text. Our focus is instead on providing extensive examples
and taking a pragmatic approach to the technology that it covers.
To true newcomers to the Hibernate API, we recommend that you read at least the first
three chapters in order before diving into the juicy subjects of later chapters. Very experienced
developers or those with experience with tools similar to Hibernate will want to skim through
the latter half of the book for interesting chapters. Readers familiar with Hibernate will want
to turn to the appendixes for discussion of more arcane topics.
How This Book Is Structured
This book is informally divided into three parts. Chapters 1 through 8 describe the fundamentals
of Hibernate, including configuration, the creation of mapping files, and the basic APIs.
Chapters 9 through 11 then describe the use of queries, criteria, and filters to access the
persistent information in more sophisticated ways.
xix
Finally, the appendixes discuss features that you will use less often, or that are peripheral
to the core Hibernate functionality. The following list describes more fully the contents of
each chapter:
Chapter 1 outlines the purpose of persistence tools and presents excerpts from a simple
example application to show how Hibernate can be applied. It also introduces core terminology
and concepts.
Chapter 2 discusses the fundamentals of configuring a Hibernate application. It presents
the basic architecture of Hibernate and discusses how a Hibernate application is integrated
into an application.
Chapter 3 presents the example application from Chapter 1 in its entirety, walking you
through the complete process of creating and running the application. It then looks at
a slightly more complex example and introduces the notion of generating the database
schema directly from the mapping files.
Chapter 4 covers the Hibernate life cycle in depth. It discusses the life cycle in the context
of the methods available on the core interfaces. It also introduces key terminology and
discusses the need for cascading and lazy loading.
Chapter 5 explains why mapping information must be retained by Hibernate, and
demonstrates the various types of associations that can be represented by a relational
database. It briefly discusses the other information that can be maintained within a
Hibernate mapping.
Chapter 6 explains how Hibernate lets you use the Java 5 Annotations feature to represent
mapping information. It provides detailed examples for the most important annotations,
and discusses the distinctions between the standard EJB 3 annotations and the proprietary
Hibernate 3 ones.
Chapter 7 explains how the XML-based mapping files can be used to represent mapping
information in Hibernate. It provides examples for all of the most common mapping
types and reference notes for the more obscure ones.
Chapter 8 revisits the Hibernate Session object in detail, explaining the various methods
that it provides. The chapter also discusses the use of transactions, locking, and caching,
and how to use Hibernate in a multithreaded environment.
Chapter 9 discusses how Hibernate can be used to make sophisticated queries against the
underlying relational database using the built-in Hibernate Query Language (HQL).
Chapter 10 introduces the Criteria API, which is a programmatic analog of the query language
discussed in Chapter 9.
Chapter 11 discusses how the filter API can be used to restrict the results of the queries
introduced in Chapters 9 and 10.
Appendix A presents a large number of peripheral features that do not warrant more
extensive coverage in a beginners’ text. Basic discussion is given, with examples, of the
use of the Hibernate EntityManager and EJB 3, the support for versioning and optimistic
locking, the provision for persisting and retrieving Dom4J document models directly, the
support for persisting and retrieving maps of information, and some of the obscure limitations
of Hibernate and various ways that these can be worked around. It also discusses
the use of events and interceptors.
Appendix B discusses how the Hibernate Tools toolset can be used to enhance development
with the Eclipse development environment and the Ant build tool. It also explains
how the Ant code-generation tasks can be customized.
Appendix C discusses how Hibernate can be integrated into the Spring API. The integration
of Hibernate as the persistence layer of a Spring application is complex, so we present
a working example, including the entire bean definition file, with discussions of the appropriate
way to manage the session in the Spring MVC environment, and how Spring can
enforce the proper transactional boundaries when using Hibernate.
Appendix D discusses some topics of interest to developers who are working with a preexisting
base of code that was built using version 2 of Hibernate. We present the various
approaches to coexisting with Hibernate 3 code and to migrating a Hibernate 2 code base
to the Hibernate 3 API.
Downloading the Code
The source code for this book is available to readers from www.apress.com, in the Source
Code/Download section. Please feel free to visit the Apress web site and download all the
code from there.
Contacting the Authors
We welcome feedback from our readers. If you have any queries or suggestions about this
book, or technical questions about Hibernate, or if you just want to share a really good joke,
you can e-mail Dave Minter at dave@paperstack.com and Jeff Linwood at jlinwood@gmail.com.
Most significant development projects involve a relational database. The mainstay of most
commercial applications is the large-scale storage of ordered information, such as catalogs,
customer lists, contract details, published text, and architectural designs.
With the advent of the World Wide Web, the demand for databases has increased. Though
they may not know it, the customers of online bookshops and newspapers are using databases.
Somewhere in the guts of the application a database is being queried and a response is offered.
While the demand for such applications has grown, their creation has not become noticeably
simpler. Some standardization has occurred—the most successful being the Enterprise
JavaBeans (EJB) standard of Java 2 Enterprise Edition (J2EE), which provides for containerand
bean-managed persistence of entity bean classes. Unfortunately, this and other persistence
models all suffer to one degree or another from the mismatch between the relational
model and the object-oriented model. In short, database persistence is difficult.
There are solutions for which EJBs are appropriate, some for which some sort of objectrelational
mapping (ORM) like Hibernate is appropriate, and some for which the traditional
approach of direct access via the Java Database Connectivity (JDBC) API is appropriate. We
think that Hibernate represents a good first choice, as it does not preclude the simultaneous
use of these alternative approaches.
To illustrate some of Hibernate’s strengths, in this chapter we will show you a brief example
using Hibernate and contrast this with the traditional JDBC approach.
Plain Old Java Objects (POJOs)
In our ideal world, it would be trivial to take any Java object and persist it to the database. No
special coding would be required to achieve this, no performance penalty would ensue, and
the result would be totally portable.
In this ideal world, we would perhaps perform such an operation in a manner like that
shown in Listing 1-1.
Listing 1-1. A Rose-Tinted View of Object Persistence
POJO pojo = new POJO();
ORMSolution magic = ORMSolution.getInstance();
magic.save(pojo);
There would be no nasty surprises, no additional work to correlate the class with tables in
the database, and no performance problems
Hibernate As a Persistence Solution
Hibernate addresses a lot of these points, or alleviates some of the pain where it can’t, so we’ll
address the points in turn.
Hibernate does not require you to map one POJO to one table. A POJO can be constructed
out of a selection of table columns, or several POJOs can be persisted into a single table.
Hibernate directly supports inheritance relationships and the various other relationships
between classes.
Though there is some performance overhead while Hibernate starts up and processes its
configuration files, it is generally perceived as being a fast tool. This is very hard to quantify, and,
to some extent, the poor reputation of entity beans may have been earned less from their own
faults than from the mistakes of those designing and deploying such applications. As with all
performance questions, you should carry out tests rather than relying on anecdotal evidence.
In Hibernate it is possible, but not necessary, to specify the mappings at deployment
time. The EJB solution ensures portability of applications across environments, but the Hibernate
approach tends to reduce the pain of deploying an application to a new environment.
Hibernate persistence has no requirement for a J2EE application server or any other special
environment. It is, therefore, a much more suitable solution for stand-alone applications,
client-side application storage, and other environments in which a J2EE server is not immediately
available.
Hibernate uses POJOs that can very easily and naturally be generalized for use in other
applications. There is no direct dependency upon the Hibernate libraries, so POJOs can be put
to any use that does not require persistence; or they can be persisted using any other “POJOfriendly”
mechanism.
Hibernate presents no problems when handling serializable POJOs.
There is a very large body of preexisting code. Any Java object capable of being persisted
to a database is a candidate for Hibernate persistence. Therefore, Hibernate is a natural
replacement for ad hoc solutions, or as the persistence engine for an application that has not
yet had database persistence incorporated into it. Furthermore, by choosing Hibernate persistence,
you are not tying yourself to any particular design decisions for the business objects
in your application.