THE EXPERT'S VOICE® IN DATABASES

Pro MongoDB™ Development

Deepak Vohra

Apress®

Pro MongoDB™ Development

Deepak Vohra

Apress®

Pro MongoDB™ Development

Copyright © 2015 by Deepak Vohra

ISBN-13 (pbk): 978-1-4842-1599-9

ISBN-13 (electronic): 978-1-4842-1598-2

Distributed to the book trade worldwide by Springer Science+Business Media New York, 233 Spring Street, 6th Floor, New York, NY 10013. Phone 1-800-SPRINGER, fax (201) 348-4505, e-mail orders-ny@springer-sbm.com, or visit www.springeronline.com. Apress Media, LLC is a California LLC and the sole member (owner) is Springer Science + Business Media Finance Inc (SSBM Finance Inc). SSBM Finance Inc is a Delaware corporation.

For information on translations, please e-mail rights@apress.com, or visit www.apress.com.

Apress and friends of ED books may be purchased in bulk for academic, corporate, or promotional use. eBook versions and licenses are also available for most titles. For more information, reference our Special Bulk Sales–eBook Licensing web page at www.apress.com/bulk-sales.

Any source code or other supplementary materials referenced by the author in this text is available to readers at www.apress.com/9781484215999. For detailed information about how to locate your book's source code, go to www.apress.com/source-code/. Readers can also access source code at SpringerLink in the Supplementary Material section for each chapter.

Contents at a Glance

Contents

About the Author

Deepak Vohra is a consultant and a principal member of the NuBean.com software company. Deepak is a Sun-certified Java programmer and Web component developer. He has worked in the fields of XML, Java programming, and Java EE for over ten years. Deepak is the author of *Pro Couchbase Development* (Apress, 2015) and the coauthor of *Pro XML Development with Java Technology* (Apress, 2006). Deepak is also the author of the *JDBC 4.0* and *Oracle JDeveloper for J2EE Development, Processing XML Documents with Oracle JDeveloper 11g, EJB 3.0 Database Persistence with Oracle Fusion Middleware 11g*, and *Java EE Development in Eclipse IDE* (Packt Publishing). He also served as the technical reviewer on *WebLogic: The Definitive Guide* (O'Reilly Media, 2004) and *Ruby Programming for the Absolute Beginner* (Cengage Learning PTR, 2007).

About the Technical Reviewers

Manuel Jordan is an autodidactic developer and researcher who enjoys learning new technologies so that he can experiment with new ways to integrate them.

Manuel won the 2010 Springy Award – Community Champion and Spring Champion 2013. In his little free time, he reads the Bible and composes music on his bass and guitar.

Massimo Nardone holds a Master of Science degree in Computing Science from the University of Salerno, Italy. He worked as a PCI QSA and Senior Lead IT Security/Cloud/SCADA Architect for many years and currently works as Security, Cloud and SCADA Lead IT Architect for Hewlett Packard Finland. He has more than twenty years of work experience in IT including Security, SCADA, Cloud Computing, IT Infrastructure, Mobile, Security, and WWW technology areas for both national and international projects. Massimo has worked as a Project Manager, Cloud/SCADA Lead IT Architect, Software Engineer, Research Engineer, Chief Security Architect, and Software Specialist. He worked as visiting a lecturer and supervisor for exercises at the Networking Laboratory of the Helsinki University of Technology (Aalto University). Massimo has been programming and teaching how to program with Perl, PHP, Java, VB, Python, C/C++, and MySQL for more than twenty years. He holds four international patents (PKI, SIP, SAML, and Proxy areas).

Massimo is the author of *Pro Android Games* (Apress, 2015).

Introduction

MongoDB server is ranked first in NoSQL databases and fourth in all databases (relational or NoSQL). While several books on MongoDB administration are available, none on MongoDB-based development are available.

This Pro *MongoDB™ Development* book is about MongoDB server, a NoSQL database based on the BSON (binary JSON) document model. As already noted, MongoDB is the most commonly used NoSQL database and is ranked fourth in all databases (relational or NoSQL) according to DB-Engines.com (Reference: `http://db-engines.com/en/ranking`). The book discusses all aspects of using MongoDB database in web development. Java, PHP, Ruby, and JavaScript are the most commonly used programming/scripting languages, and the book discusses accessing MongoDB database with these languages. The book also discusses using Java EE frameworks Kundera and Spring Data with MongoDB. As NoSQL databases are commonly used with the Hadoop ecosystem, the book discusses using MongoDB with Apache Hadoop and Apache Hive. An Oracle Data Integrator-based integration of MongoDB data into Oracle Database is also discussed. Migration from other NoSQL databases (Apache Cassandra and Couchbase) and from relational database (Oracle Database) is discussed.

This book is for web developers and NoSQL developers who develop applications using MongoDB server. Prerequisite knowledge of Java, Java EE, and scripting languages PHP, Ruby, and JavaScript is essential. Familiarity with Apache Hadoop, Apache Hive, Oracle Database, and Oracle Data Integrator is also a prerequisite.

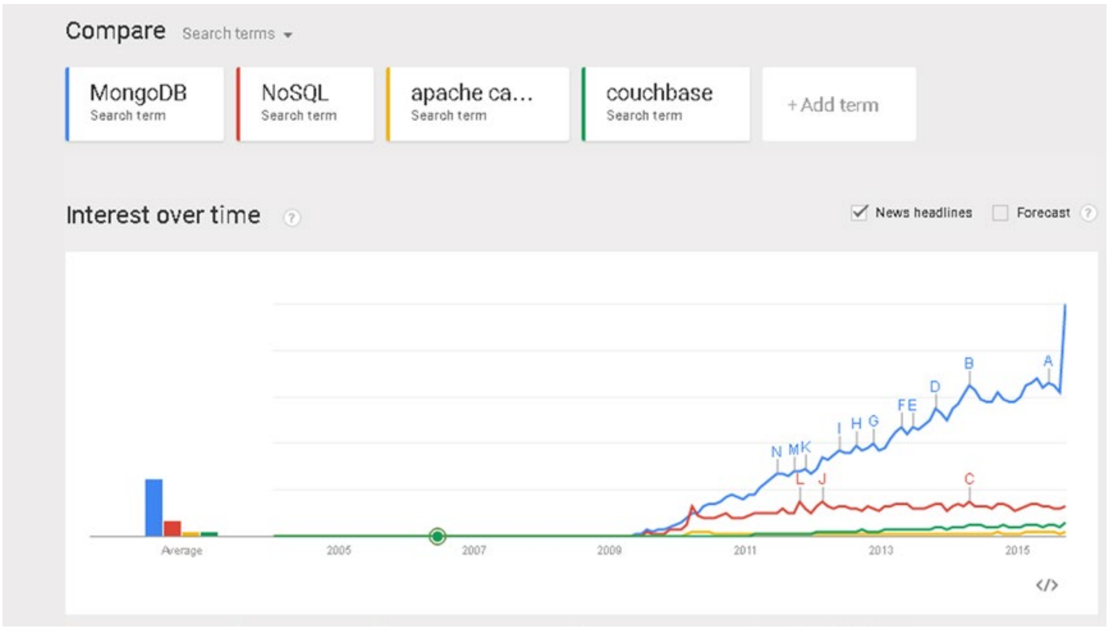
According to the Java Tools and Technologies Landscape for 2014, "In the NoSQL world... among the developers using NoSQL... MongoDB (56%) is clearly leading..." (Reference: `http://zeroturnaround.com/rebellabs/java-tools-and-technologies-landscape-for-2014/13/`).

More than 2000 organizations use MongoDB (Reference: `www.mongodb.com/who-uses-mongodb`).

Various metrics have ranked MongoDB above all other NoSQL databases (Reference: `www.mongodb.com/leading-nosql-database`).

MongoDB was named as the Database of the Year in 2013 by DB-Engines (Reference: `www.mongodb.com/blog/post/mongodb-named-2013-database-year-why-matters`).

Google Trends search term ranking for "MongoDB" as compared with "Apache Cassandra" and "Couchbase" in September 2015 is shown in the following illustration.



Google Trends Database Search Results

CHAPTER 1

Using a Java Client with MongoDB

MongoDB server provides drivers for several languages including Java. The MongoDB Java driver may be used to connect to MongoDB server from a Java application and create a collection, get a collection or a list of collections, add a document or multiple documents to a collection, and find a document or a set of documents. In this chapter we shall create a Java application in Eclipse and access MongoDB server to add a document and subsequently get data from MongoDB server and also update and delete data in MongoDB server. This chapter covers the following topics:

- Setting up the environment
- Creating a Maven project
- Creating a BSON document in MongoDB
- Using a model to create a BSON document in MongoDB
- Getting data from MongoDB
- Updating data in MongoDB
- Deleting data in MongoDB

Setting Up the Environment

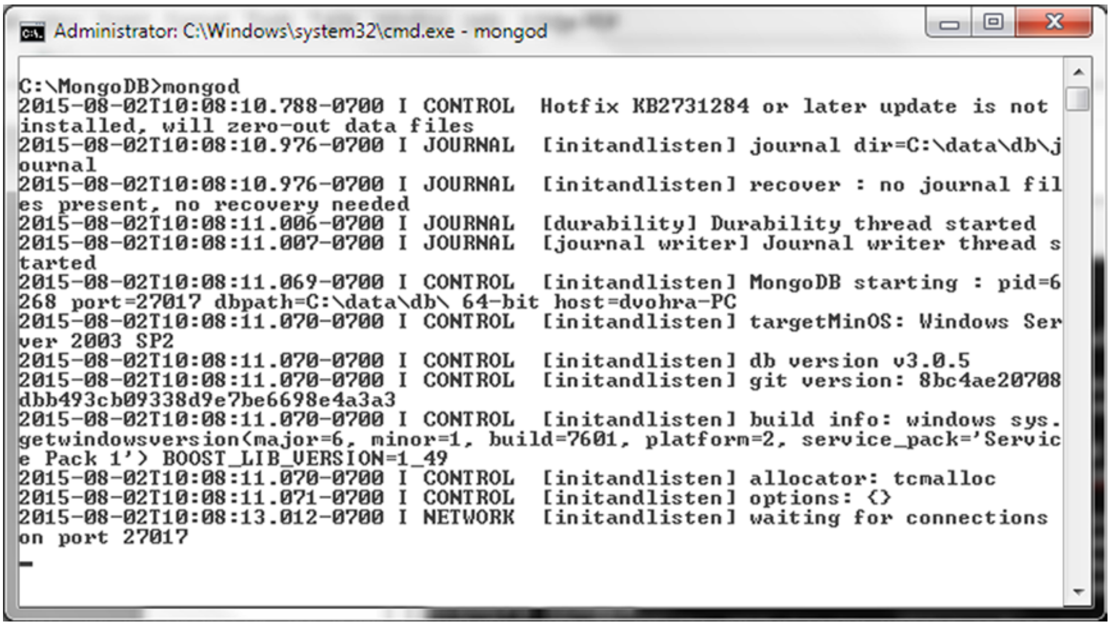
We need to download and install the following software.

1. MongoDB 3.0.5 binary distribution `mongodb-win32-x86_64-3.0.5-signed.msi` from `www.mongodb.org/downloads`. Stable Release (3.0.5) for Windows 64-bit is used in this chapter.
2. Eclipse IDE for Java EE Developers from `www.eclipse.org/downloads`.
3. Java 5 or later (Java 7 used) from `www.oracle.com/technetwork/java/javase/downloads/index.html`.

Double-click on the MongoDB binary distribution to install MongoDB. Add the `bin` directory (`C:\Program Files\MongoDB\Server\3.0\bin`) of the MongoDB installation to the `PATH` environment. Create a directory `C:\data\db` for the MongoDB data. Start the MongoDB server with the following command.

```
>mongod
```

The MongoDB server gets started as shown in Figure 1-1.



```
C:\MongoDB>mongod
2015-08-02T10:08:10.788-0700 I CONTROL  Hotfix KB2731284 or later update is not
installed, will zero-out data files
2015-08-02T10:08:10.976-0700 I JOURNAL  [initandlisten] journal dir=C:\data\db\j
ournal
2015-08-02T10:08:10.976-0700 I JOURNAL  [initandlisten] recover : no journal fil
es present, no recovery needed
2015-08-02T10:08:11.006-0700 I JOURNAL  [durability] Durability thread started
2015-08-02T10:08:11.007-0700 I JOURNAL  [journal writer] Journal writer thread s
tarted
2015-08-02T10:08:11.069-0700 I CONTROL  [initandlisten] MongoDB starting : pid=6
268 port=27017 dbpath=C:\data\db\ 64-bit host=dvohra-PC
2015-08-02T10:08:11.070-0700 I CONTROL  [initandlisten] targetMinOS: Windows Ser
ver 2003 SP2
2015-08-02T10:08:11.070-0700 I CONTROL  [initandlisten] db version v3.0.5
2015-08-02T10:08:11.070-0700 I CONTROL  [initandlisten] git version: 8bc4ae20708
dbb493cb09338d9e7be6698e4a3a3
2015-08-02T10:08:11.070-0700 I CONTROL  [initandlisten] build info: windows sys.
getwindowsversion(major=6, minor=1, build=7601, platform=2, service_pack='Servic
e Pack 1') BOOST_LIB_VERSION=1_49
2015-08-02T10:08:11.070-0700 I CONTROL  [initandlisten] allocator: tcmalloc
2015-08-02T10:08:11.071-0700 I CONTROL  [initandlisten] options: {}
2015-08-02T10:08:13.012-0700 I NETWORK  [initandlisten] waiting for connections
on port 27017
_
```

Figure 1-1. *Starting MongoDB*

Creating a Maven Project

To access MongoDB from a Java application we need to create a Maven project in Eclipse IDE and add the required dependencies.

1. Select File ➤ New ➤ Other. In the New window, select the Maven ➤ Maven Project wizard and click on Next as shown in Figure 1-2.

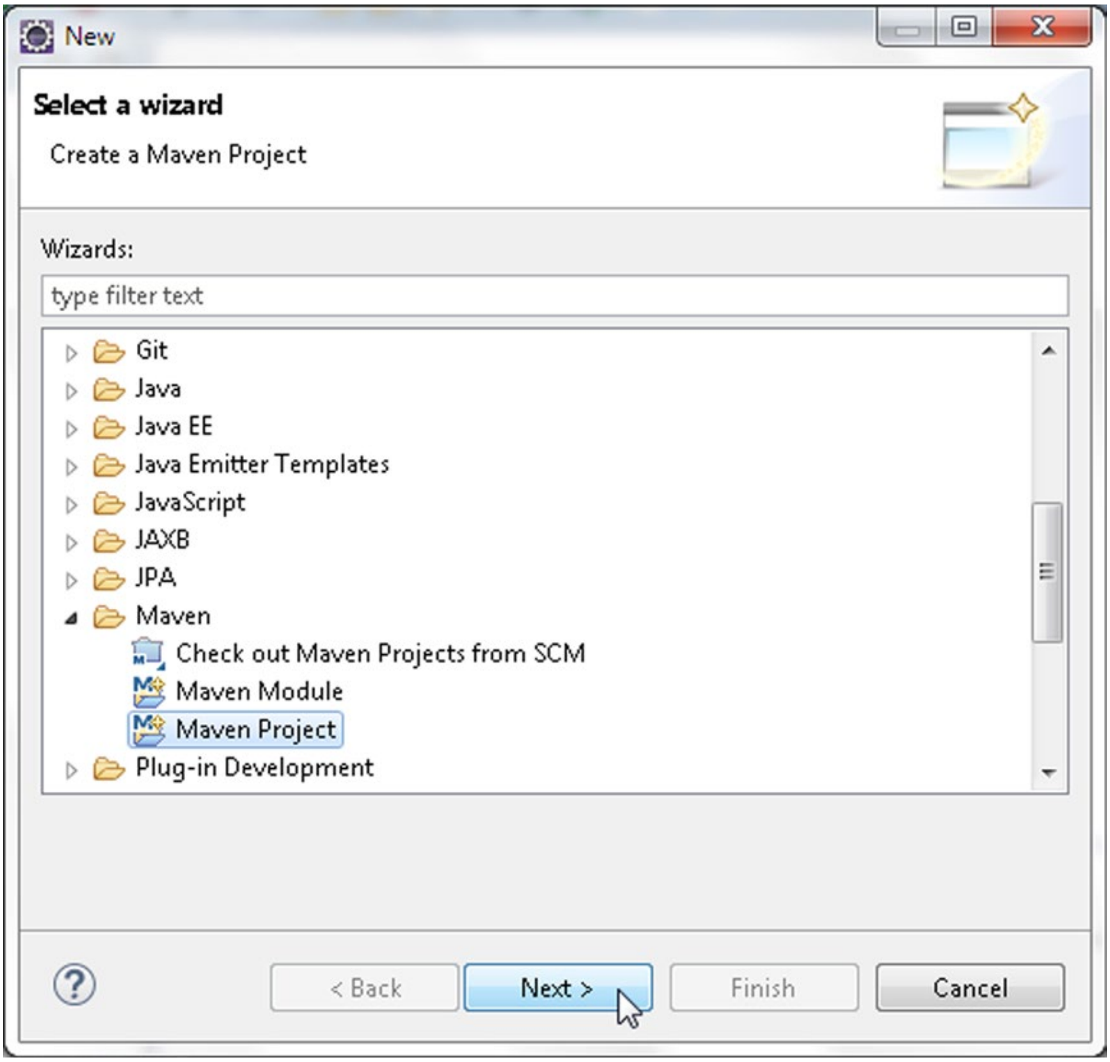
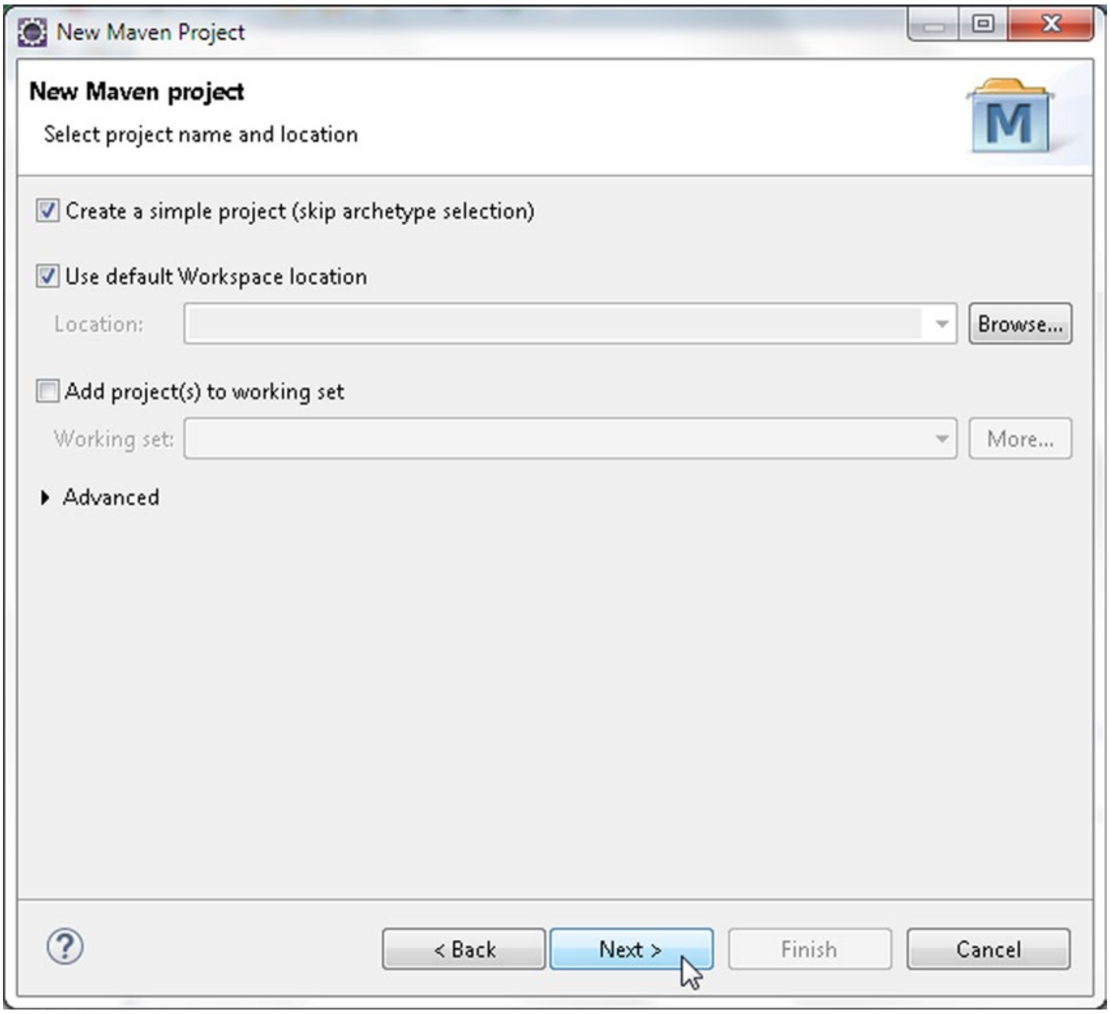


Figure 1-2. *Selecting Maven ➤ Maven Project*

2. In New Maven Project wizard select the check boxes "Create a simple project" and "Use default Workspace location" as shown in Figure 1-3. Click on Next.



[✓] Create a simple project (skip archetype selection)

[✓] Use default Workspace location

Location:

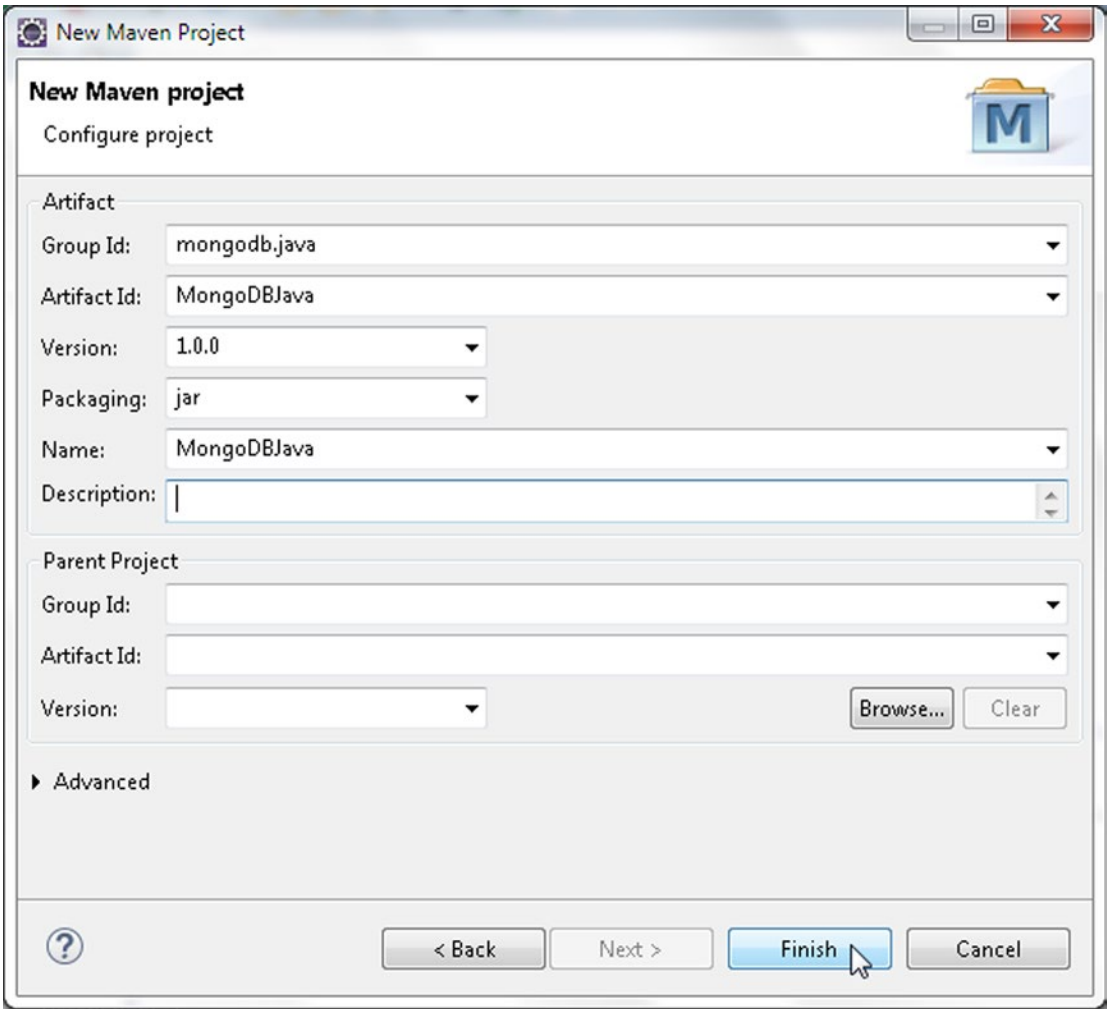
[ ] Add project(s) to working set

Working set:

Figure 1-3. New Maven Project wizard

3. Next, configure the project, specifying the following values as shown in Figure 1-4 and then clicking on Finish:

   - Group Id: `mongodb.java`
   - Artifact Id: `MongoDBJava`
   - Version: 1.0.0
   - Packaging: jar
   - Name: `MongoDBJava`



New Maven Project

New Maven project

Configure project

Artifact

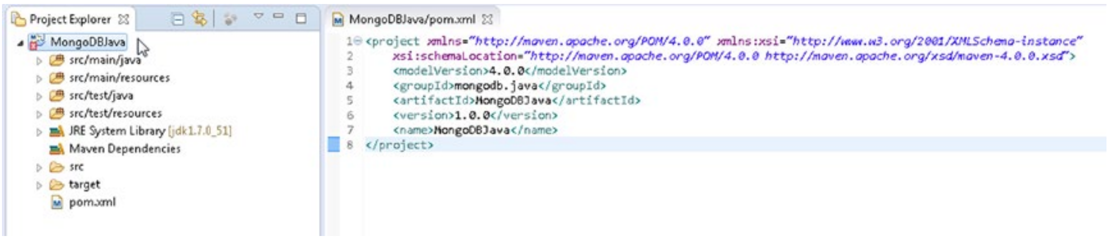
Group Id:	mongodb.java
Artifact Id:	MongoDBJava
Version:	1.0.0
Packaging:	jar
Name:	MongoDBJava
Description:	

Parent Project

Group Id:	
Artifact Id:	
Version:	

Figure 1-4. *Configuring the Maven Project*

A new Maven project gets created as shown in Figure 1-5.



```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>mongodb.java</groupId>
    <artifactId>MongoDBJava</artifactId>
    <version>1.0.0</version>
    <name>MongoDBJava</name>
</project>
```

Figure 1-5. *Maven Project MongoDBJava*

4. We need to add some Java classes to the Maven project to run CRUD operations on MongoDB server. Select File ➤ New ➤ Other, and in the New window, select Java ➤ Class as shown in Figure 1-6.

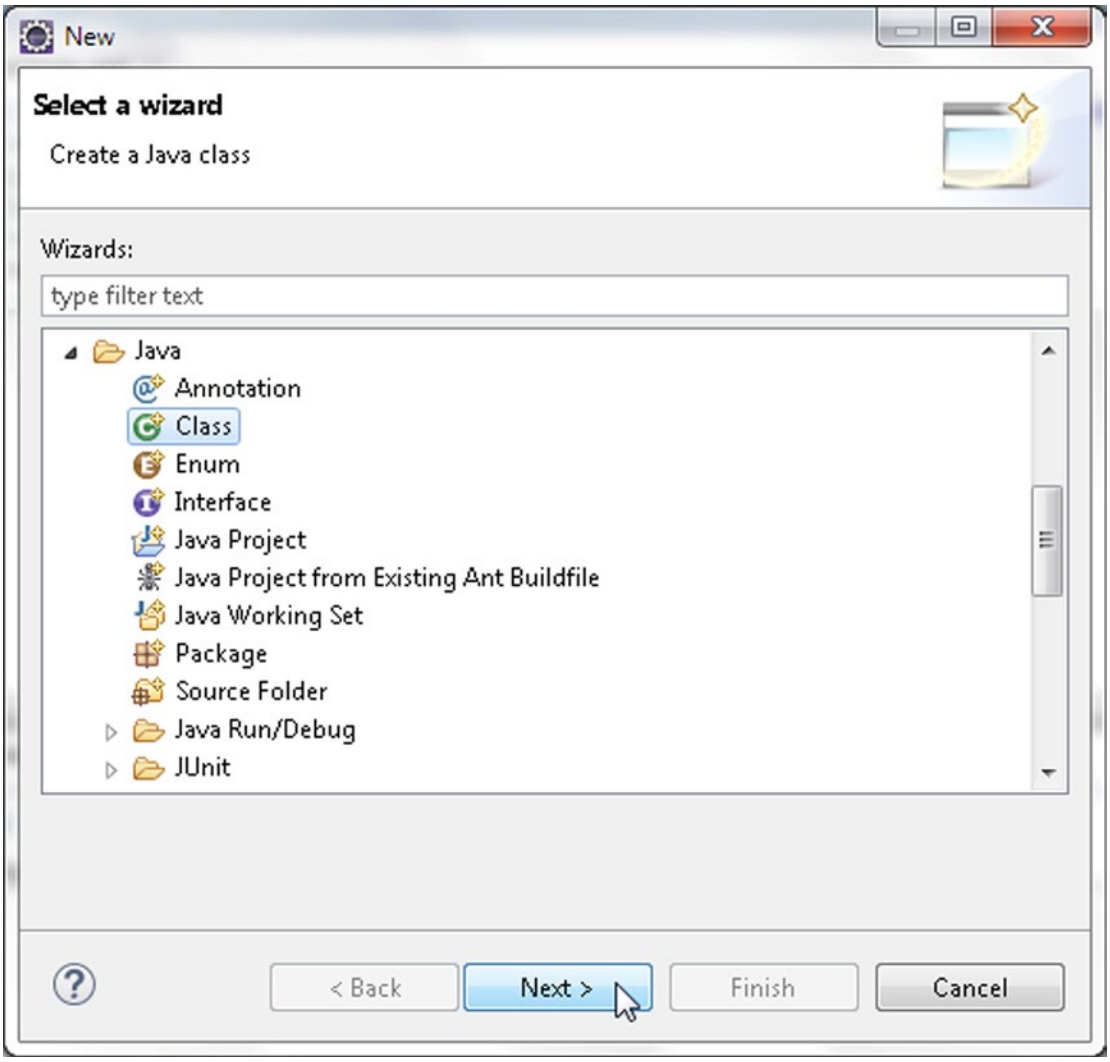
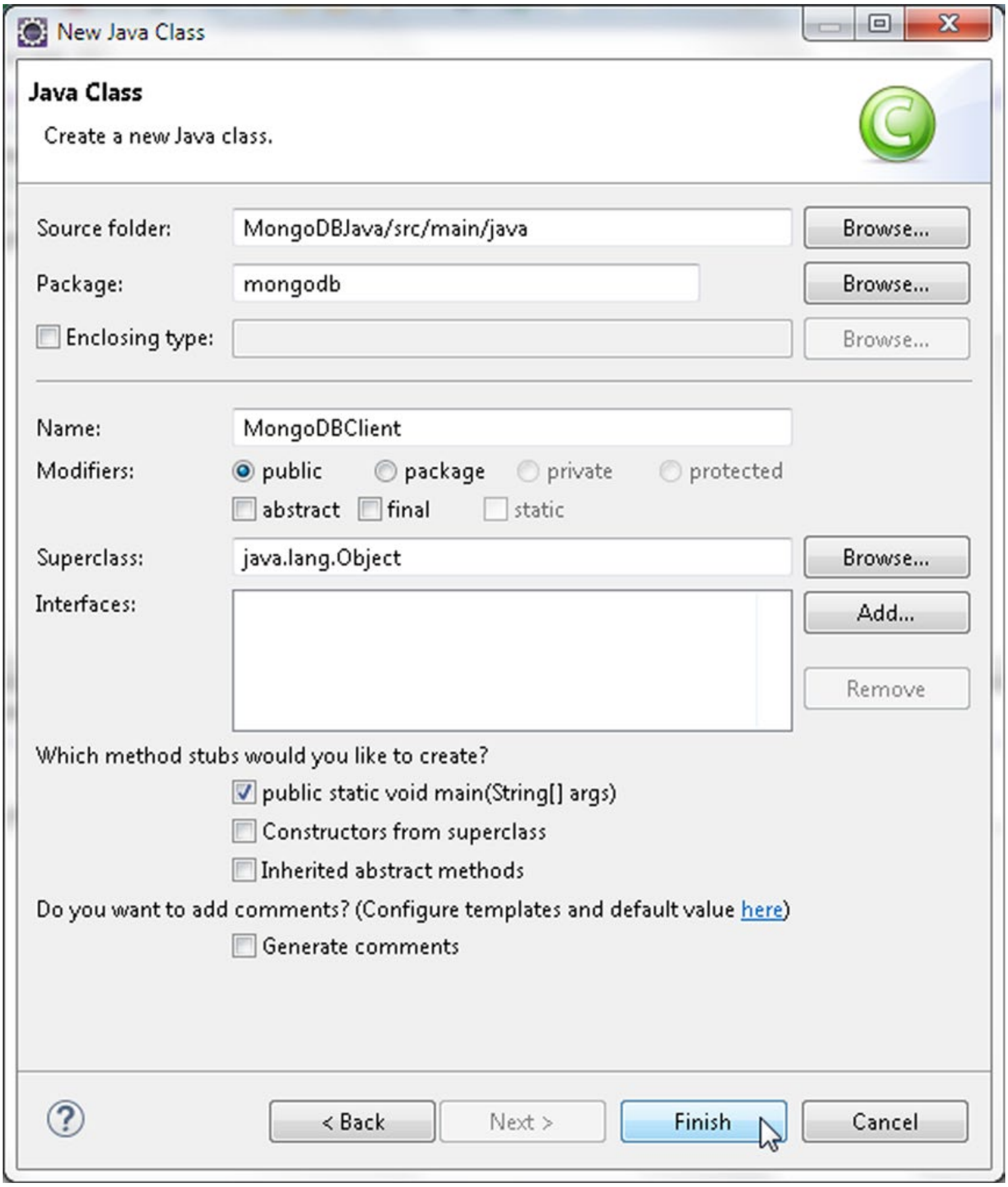


Figure 1-6. *Selecting Java ➤ Class*

5. In New Java Class wizard the Source folder is preselected as `MongoDBJava/src/main/java`. Specify a Package name (`mongodb`). Specify a class Name (for example, `MongoDBClient` for the application to connect to MongoDB and get data). Select the `public static void main (String[] args)` method stub to create the class and click on Finish as shown in Figure 1-7.


New Java Class

Java Class

Create a new Java class.

Source folder: MongoDBJava/src/main/java

Package: mongodb

[ ] Enclosing type:

Name: MongoDBClient

Modifiers: [✓ public] package private protected

[ ] abstract [ ] final [ ] static

Superclass: java.lang.Object

Interfaces:

Which method stubs would you like to create?

[✓] public static void main(String[] args)

[ ] Constructors from superclass

[ ] Inherited abstract methods

Do you want to add comments? (Configure templates and default value here)

[ ] Generate comments

Figure 1-7. *Creating a New Class*

6. The `MongoDBClient` class gets created in the `MongoDBJava` project. Similarly add the Java classes listed in Table 1-1 to the same package as the `MongoDBClient` class: the `mongodb` package.

***Table 1-1.** Java Classes*

Class	Description
`Catalog`	The model class to define properties and accessor methods for a record to be added to MongoDB server.
`CreateMongoDBDocument`	Class to create a MongoDB document.
`CreateMongoDBDocumentModel`	Class to create a MongoDB document using the model class catalog.
`UpdateDBDocument`	Class to update a document.
`DeleteDBDocument`	Class to delete a document.

The Java classes in the Maven project are shown in the Package Explorer in Figure 1-8.

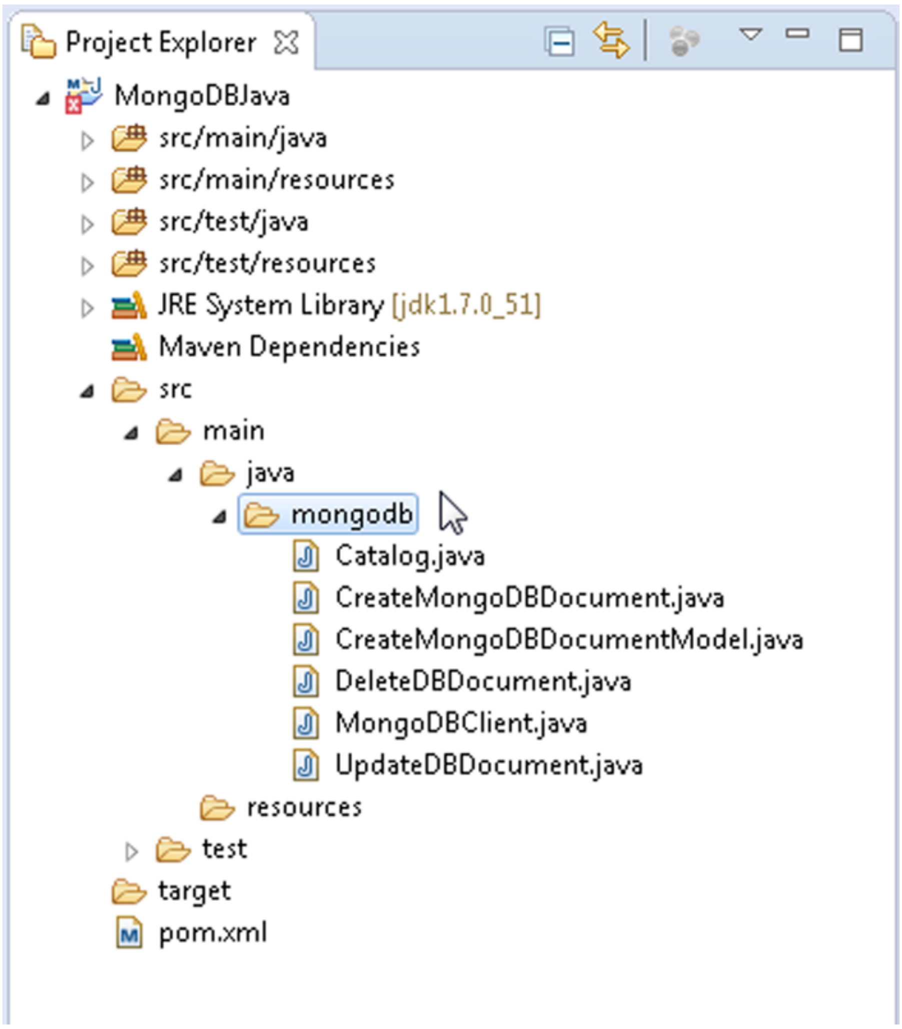
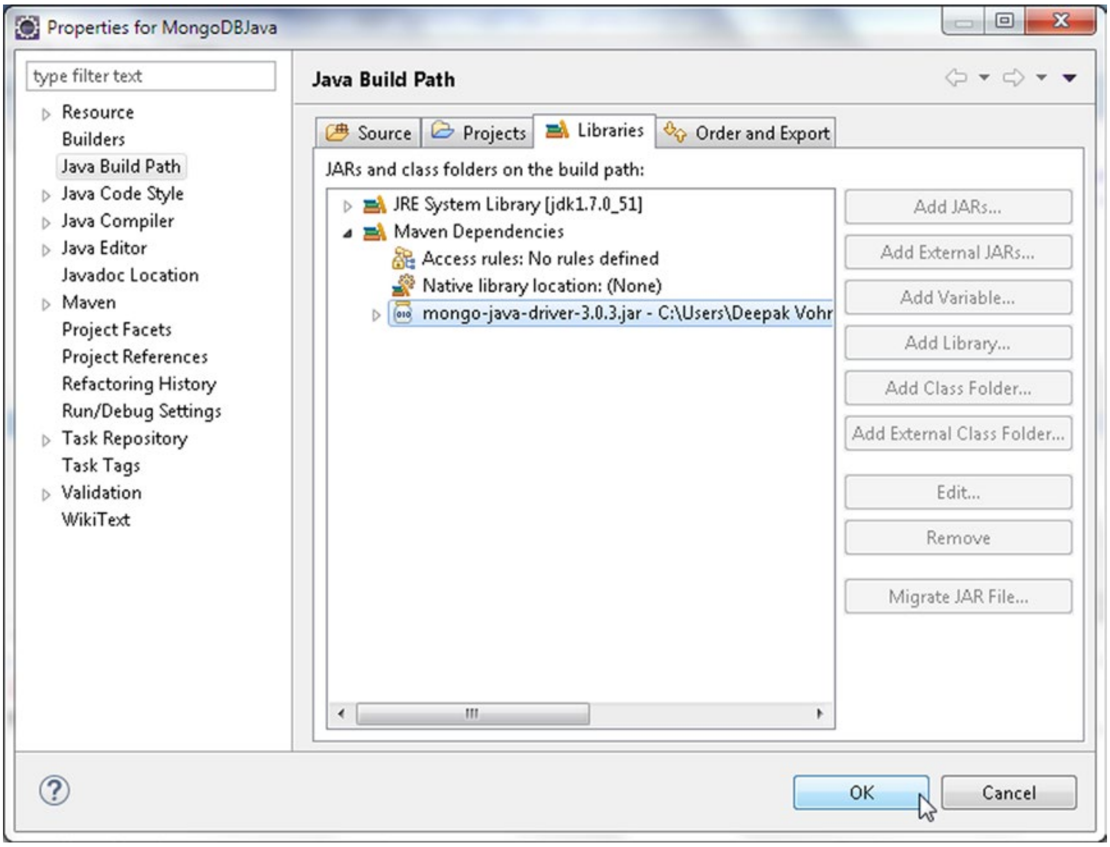


***Figure 1-8.** Java Classes*

7. Next, configure the Maven project by adding the required dependencies to the `pom.xml` configuration file. Add the Java MongoDB Driver dependency to the `pom.xml`. The `pom.xml` is listed below. Click on File ➤ Save All to save the `pom.xml`.

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
    http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>mongodb.java</groupId>
    <artifactId>MongoDBJava</artifactId>
    <version>1.0.0</version>
    <name>MongoDBJava</name>
    <dependencies>
        <dependency>
            <groupId>org.mongodb</groupId>
            <artifactId>mongo-java-driver</artifactId>
            <version>3.0.3</version>
        </dependency>
    </dependencies>
</project>
```

8. Right-click on the project node in Package Explorer and select Project Properties. In Properties select Java Build Path. Select the Libraries tab and click on Maven Dependencies to expand the node. The MongoDB Java driver should be listed in the Java Build Path as shown in Figure 1-9. Click on OK.

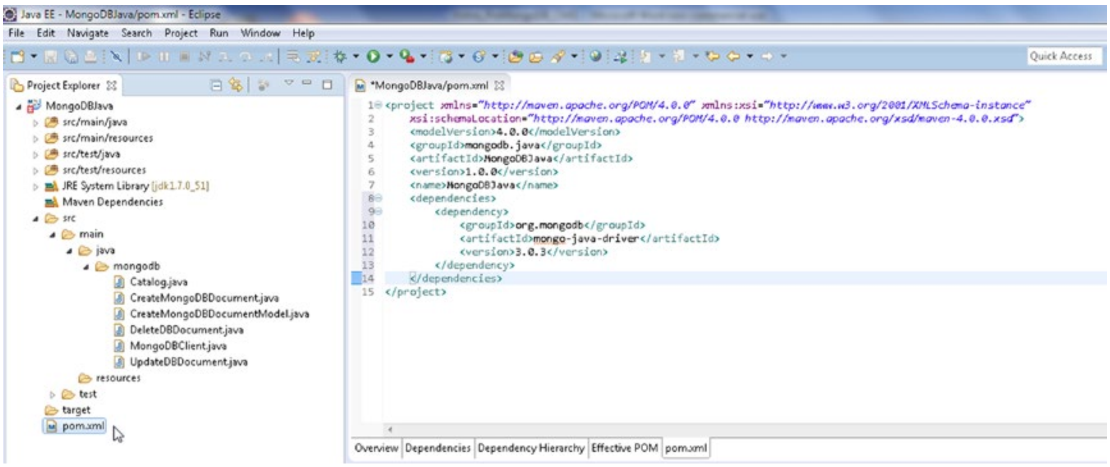


JARs and class folders on the build path:

- JRE System Library [jdk1.7.0_51]
- Maven Dependencies
  - Access rules: No rules defined
  - Native library location: (None)
  - mongo-java-driver-3.0.3.jar - C:\Users\Deepak Vohr

Figure 1-9. Mongo Java Driver in Java Build Path

The Maven project with the Mongo Java driver dependency configured in `pom.xml` is shown in Figure 1-10. The directory structure of the `MongoDBJava` Maven project is shown in the Package Explorer.



```
MongoDBJava
  src/main/java
  src/main/resources
  src/test/java
  src/test/resources
  JRE System Library [jdk1.7.0_51]
  Maven Dependencies
  src
    main
      java
        mongodb
          Catalog.java
          CreateMongoDBDocument.java
          CreateMongoDBDocumentModel.java
          DeleteDBDocument.java
          MongoDBClient.java
          UpdateDBDocument.java
      resources
    test
  target
  pom.xml
```

```xml
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>mongodb.java</groupId>
    <artifactId>MongoDBJava</artifactId>
    <version>1.0.0</version>
    <name>MongoDBJava</name>
    <dependencies>
        <dependency>
            <groupId>org.mongodb</groupId>
            <artifactId>mongo-java-driver</artifactId>
            <version>3.0.3</version>
        </dependency>
    </dependencies>
</project>
```

Figure 1-10. Maven Project with Mongo Java Driver Dependency Configured

Creating a BSON Document

In this section we shall add a BSON (Binary JSON) document to MongoDB server. We shall use the `CreateMongoDBDocument.java` application in Eclipse IDE. A MongoDB document is represented with the `org.bson.Document` class. MongoDB stores data in collections. The main packages for MongoDB classes in the MongoDB Java driver are `com.mongodb` and `com.mongodb.client`. A MongoDB client to connect to MongoDB server is represented with the `com.mongodb.MongoClient` class. A `MongoClient` object provides connection pooling, and only one instance is required for the entire instance. The `MongoClient` class provides several constructors, some of which are listed in Table 1-2.

Table 1-2. *MongoClient Class Constructors*

Constructor	Description
`MongoClient()`	Creates an instance based on a single MongoDB node for localhost and default port 27017.
`MongoClient(String host)`	Creates an instance based on a single MongoDB node with host specified as a host[:port] String literal.
`MongoClient(String host, int port)`	Creates an instance based on a single MongoDB node using the specified host and port.
`MongoClient(List<ServerAddress> seeds)`	Creates an instance using a List of MongoDB servers to select from. The server with the lowest ping time is selected. If the lowest ping time server is down, the next in the list is selected.
`MongoClient(List<ServerAddress> seeds, List<MongoCredential> credentialsList)`	Same as the previous version except a List of credentials are provided to authenticate connections to the server/s.
`MongoClient(List<ServerAddress> seeds, List<MongoCredential> credentialsList, MongoClientOptions options)`	Same as the previous version except that Mongo client options are also provided.

1. Create a `MongoClient` instance using the `MongoClient(List<ServerAddress> seeds)` constructor. Supply “localhost” or the IPv4 address of the host and port as 27017.

```
MongoClient mongoClient = new MongoClient
(Arrays.asList(new ServerAddress("localhost", 27017)));
```

2. When creating many `MongoClient` instances, all resource usage limits apply per `MongoClient` instance. To close an instance you need to call `MongoClient.close()` to clean up resources. A logical database in MongoDB is represented with the `com.mongodb.client.MongoDatabase` interface. Obtain a `com.mongodb.client.MongoDatabase` instance for the “local” database, which is a default MongoDB database instance, using the `getDatabase(String databaseName)` method in `MongoClient` class.

```
MongoDatabase db = mongoClient.getDatabase("local");
```

3. Some of the Mongo client API has been modified in version 3.0.x. For example, a database instance is represented with the `MongoDatabase` in 3.0.x instead of `com.mongodb.DB`. The `getDB(String dbName)` method, which returns a `DB` instance, in `MongoClient` is deprecated. A database collection in 3.0.x is represented with `com.mongodb.client.MongoCollection<TDocument>` instead of `com.mongodb.DBCollection`. Get all collections from the database instance using the `listCollectionNames()` method in `MongoDatabase`.

   ```
   MongoIterable<String> colls = db.listCollectionNames();
   ```

4. The `listCollectionNames()` method returns a `MongoIterable<String>` of collections. Iterate over the collection to output the collection names.

   ```
   for (String s : colls) {
   System.out.println(s);
   }
   ```

5. Next, create a new `MongoCollection<Document>`instance using the `getCollection(String collectionName)` method in `MongoDatabase`. Create a collection of `Document` instances called `catalog`. A collection gets created implicitly when the `getCollection(String)` method is invoked.

   ```
   MongoCollection<Document> coll = db.getCollection("catalog");
   ```

A MongoDB specific BSON object is represented with the `org.bson.Document` class, which implements the `Map` interface among others. The `Document` class provides the following constructors listed in Table 1-3 to create a new instance.

Table 1-3. *Document Class Constructors*

Constructor	Description
`Document()`	Creates an empty `Document` instance.
`Document(Map<String,Object> map)`	Creates a `Document` instance initialized with a `Map`.
`Document(String key, Object value)`	Creates a `Document` instance initialized with a key/value pair.

The `Document` class provides some other utility methods, some of which are in Table 1-4.

Table 1-4. *Document Class Utility Methods*

Method	Description
`append(String key, Object value)`	Appends a key/value pair to a `Document` object and returns a new instance.
`toString()`	Returns a String representation of the object.

6. Create a Document instance using the Document(String key, Object value) constructor and use the append(String key, Object value) method to append key/value pairs. The append() method may be invoked multiple times in sequence to add multiple key/value pairs. Add key/value pairs for the journal, publisher, edition, title, and author fields.

   ```
   Document catalog = new Document("journal", "Oracle Magazine")
   .append("publisher", "Oracle Publishing")
   .append("edition", "November December 2013")
   .append("title", "Engineering as a Service").append("author",
   "David A. Kelly");
   ```

7. The MongoCollection<TDocument> interface provides insertOne(TDocument document) method to add a document(s) to a collection. Add the catalog Document to the MongoCollection<TDocument> instance for the catalog collection.

   ```
   coll.insertOne(catalog);
   ```

8. The MongoCollection<TDocument> interface provides overloaded find() method to find a Document instance. Next, obtain the document added using the find() method. Furthermore, the find() method returns an iterable collection from which we obtain the first document using the first() method.

   ```
   Document dbObj = coll.find().first();
   ```

9. Output the Document object found as such and also by iterating over the Set<E> obtained from the Document using the keySet() method. The keySet() method returns a Set<String>. Create an Iterator from the Set<String> using the iterator() method. While the Iterator has elements as determined by the hasNext() method, obtain the elements using the next() method. Each element is a key in the Document fetched. Obtain the value for the key using the get(String key) method in Document.

   ```
   System.out.println(dbObj);
   Set<String> set = dbObj.keySet();
   Iterator iter = set.iterator();
   while(iter.hasNext()){
   Object obj=     iter.next();
   System.out.println(obj);
   System.out.println(dbObj.get(obj.toString()));
   }
   ```

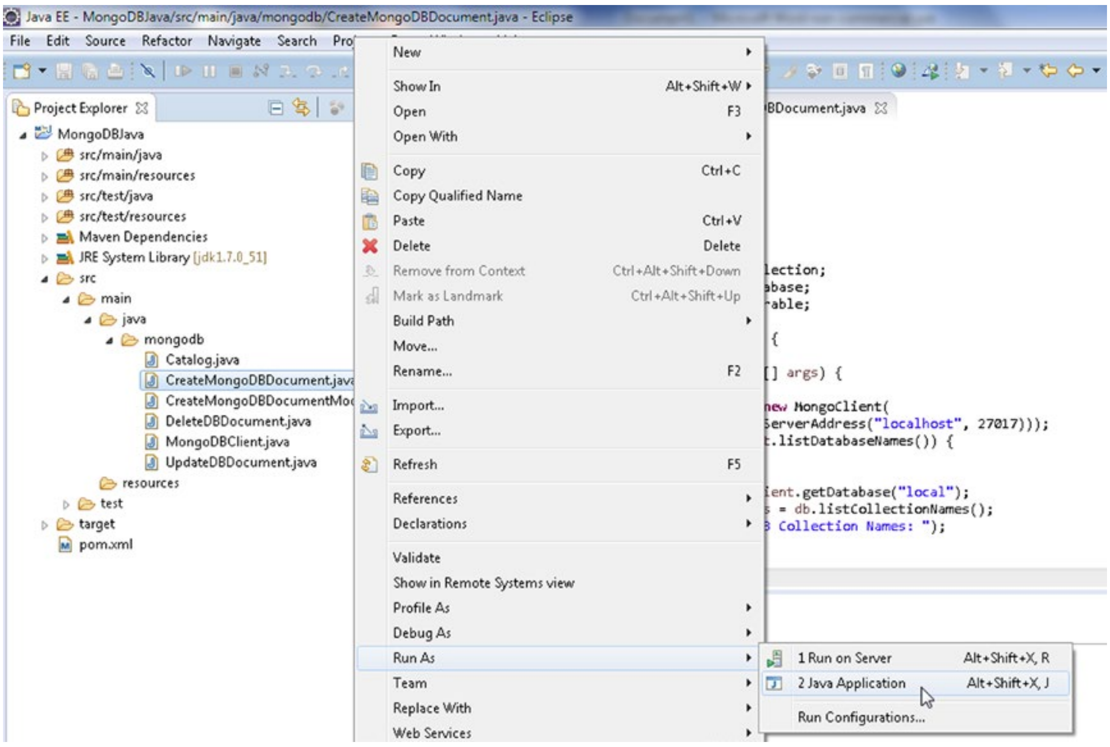
10. Close the MongoClient instance.

   ```
   mongoClient.close();
   ```

The `CreateMongoDBDocument` class is listed below.

```java
package mongodb;

import java.util.Arrays;
import java.util.Iterator;
import java.util.Set;
import org.bson.Document;
import com.mongodb.MongoClient;
import com.mongodb.ServerAddress;
import com.mongodb.client.MongoCollection;
import com.mongodb.client.MongoDatabase;
import com.mongodb.client.MongoIterable;

public class CreateMongoDBDocument {

    public static void main(String[] args) {

        MongoClient mongoClient = new MongoClient(
                Arrays.asList(new ServerAddress("localhost", 27017)));
        for (String s : mongoClient.listDatabaseNames()) {
            System.out.println(s);
        }
        MongoDatabase db = mongoClient.getDatabase("local");
        MongoIterable<String> colls = db.listCollectionNames();
        System.out.println("MongoDB Collection Names: ");
        for (String s : colls) {
            System.out.println(s);
        }
        MongoCollection<Document> coll = db.getCollection("catalog");
        Document catalog = new Document("journal", "Oracle Magazine")
                .append("publisher", "Oracle Publishing")
                .append("edition", "November December 2013")
                .append("title", "Engineering as a Service")
                .append("author", "David A. Kelly");
        coll.insertOne(catalog);
        Document dbObj = coll.find().first();
        System.out.println(dbObj);
        Set<String> set = catalog.keySet();
        Iterator<String> iter = set.iterator();
        while (iter.hasNext()) {
            Object obj = iter.next();
            System.out.println(obj);
            System.out.println(dbObj.get(obj.toString()));
        }
        mongoClient.close();
    }
}
```

11. To run the `CreateMongoDBDocument` application, right-click on the `CreateMongoDBDocument.java` file in Package Explorer and select Run As ➤ Java Application as shown in Figure 1-11.



```
lection;
abase;
able;

{
[] args) {

new MongoClient(
ServerAddress("localhost", 27017)));
.listDatabaseNames()) {

ient.getDatabase("local");
 = db.listCollectionNames();
 Collection Names: ");
```

Figure 1-11. Running CreateMongoDBDocument.java Application

A new BSON document gets stored in a new collection `catalog` in MongoDB database. The document stored is also output as such and as key/value pairs as shown in Figure 1-12.