Java Development with MongoDB

Brendan W. McAdams

Novus Partners, Inc.
http://novus.com

May 2010 / Mongo NYC

Outline

1. MongoDB + Java Basics
   - Setting up your Java environment
   - Connecting to MongoDB
2. Working with MongoDB
   - Collections + Documents
   - Inserting Documents to MongoDB
   - Querying MongoDB

Outline

1 MongoDB + Java Basics
- Setting up your Java environment
- Connecting to MongoDB

2 Working with MongoDB
- Collections + Documents
- Inserting Documents to MongoDB
- Querying MongoDB

Adding the MongoDB Driver To Your Project

Assuming you're using a dependency manager, make setup simple. . .

- Maven Dependency

```
<dependency>
<groupId>org.mongodb</groupId>
<artifactId>mongo-java-driver</artifactId>
<version>1.4</version>

</dependency>
```

- Ivy Dependency

```
<dependency org="org.mongodb" name="mongo-java-driver"

rev="1.4"/>
```

Adding the MongoDB Driver To Your Project

Assuming you're using a dependency manager, make setup simple...

- Maven Dependency

```
<dependency>
<groupId>org.mongodb</groupId>
<artifactId>mongo-java-driver</artifactId>
<version>1.4</version>

</dependency>
```

- Ivy Dependency

```
<dependency org="org.mongodb" name="mongo-java-driver"

rev="1.4"/>
```

Adding the MongoDB Driver To Your Project

Assuming you’re using a dependency manager, make setup simple…

- Maven Dependency

```
<dependency>
<groupId>org.mongodb</groupId>
<artifactId>mongo-java-driver</artifactId>
<version>1.4</version>

</dependency>
```

- Ivy Dependency

```
<dependency org="org.mongodb" name="mongo-java-driver"

rev="1.4"/>
```

Outline

1 MongoDB + Java Basics
- Setting up your Java environment
- Connecting to MongoDB

2 Working with MongoDB
- Collections + Documents
- Inserting Documents to MongoDB
- Querying MongoDB

Simple Connection

Getting connected to MongoDB is simple; Connections are pooled, so you only need one…

```
import com.mongodb.Mongo;

import com.mongodb.DB;
```

- `Mongo m = new Mongo();`
- `Mongo m = new Mongo(“localhost”);`
- `Mongo m = new Mongo(“localhost”, 27017);`
- Fetch a Database Handle (lazy)…

  `DB db = m.getDB(“javaDemo”);`
- If you need to authenticate…

  `boolean auth = db.authenticate(“login”, “password”);`

Simple Connection

Getting connected to MongoDB is simple; Connections are pooled, so you only need one…

```
import com.mongodb.Mongo;
```

```
import com.mongodb.DB;
```

- `Mongo m = new Mongo();`
- `Mongo m = new Mongo("localhost");`
- `Mongo m = new Mongo("localhost", 27017);`
- Fetch a Database Handle (lazy)…

  `DB db = m.getDB("javaDemo");`
- If you need to authenticate…

  `boolean auth = db.authenticate("login", "password");`

Simple Connection

Getting connected to MongoDB is simple; Connections are pooled, so you only need one…

```
import com.mongodb.Mongo;
import com.mongodb.DB;
```

- `Mongo m = new Mongo();`
- `Mongo m = new Mongo("localhost");`
- `Mongo m = new Mongo("localhost", 27017);`
- Fetch a Database Handle (lazy)…

  `DB db = m.getDB("javaDemo");`
- If you need to authenticate…

  `boolean auth = db.authenticate("login", "password");`

Simple Connection

Getting connected to MongoDB is simple; Connections are pooled, so you only need one...

```
import com.mongodb.Mongo;
import com.mongodb.DB;
```

- `Mongo m = new Mongo();`
- `Mongo m = new Mongo(“localhost”);`
- `Mongo m = new Mongo(“localhost”, 27017);`
- Fetch a Database Handle (lazy)...

  `DB db = m.getDB(“javaDemo”);`
- If you need to authenticate...

  `boolean auth = db.authenticate(“login”, “password”);`

NOVUS™

Simple Connection

Getting connected to MongoDB is simple; Connections are pooled, so you only need one. . .

```
import com.mongodb.Mongo;

import com.mongodb.DB;
```

- `Mongo m = new Mongo();`
- `Mongo m = new Mongo("localhost");`
- `Mongo m = new Mongo("localhost", 27017);`
- Fetch a Database Handle (lazy). . .

  ```
  DB db = m.getDB("javaDemo");
  ```
- If you need to authenticate. . .

  ```
  boolean auth = db.authenticate("login", "password");
  ```

Simple Connection

Getting connected to MongoDB is simple; Connections are pooled, so you only need one…

```
import com.mongodb.Mongo;
import com.mongodb.DB;
```

- `Mongo m = new Mongo();`
- `Mongo m = new Mongo("localhost");`
- `Mongo m = new Mongo("localhost", 27017);`
- Fetch a Database Handle (lazy)…

  `DB db = m.getDB("javaDemo");`
- If you need to authenticate…

  `boolean auth = db.authenticate("login", "password");`

NOVUS™

Outline

1 MongoDB + Java Basics
- Setting up your Java environment
- Connecting to MongoDB

2 Working with MongoDB
- Collections + Documents
- Inserting Documents to MongoDB
- Querying MongoDB

Working with Collections

Collections are MongoDB “tables”…

- List all of the collections in a database…

```
Set<String> colls = db.getCollectionNames();
for (String s : colls) {
            System.out.println(s);
}
```

- Get a specific collection (lazy)…

```
Set<String> colls = db.getCollectionNames();
for (String s : colls) {
            System.out.println(s);
}
```

- Count the number of documents in a collection…

```
coll.getCount();
```

Working with Collections

Collections are MongoDB “tables”. . .

- List all of the collections in a database. . .

```
Set<String> colls = db.getCollectionNames();
for (String s :  colls) {
            System.out.println(s);
}
```

- Get a specific collection (lazy). . .

```
Set<String> colls = db.getCollectionNames();
for (String s :  colls) {
            System.out.println(s);
}
```

- Count the number of documents in a collection. . .

```
coll.getCount();
```

Working with Collections

Collections are MongoDB “tables”. . .

- List all of the collections in a database. . .

```
Set<String> colls = db.getCollectionNames();
for (String s : colls) {
            System.out.println(s);
}
```

- Get a specific collection (lazy). . .

```
Set<String> colls = db.getCollectionNames();
for (String s : colls) {
            System.out.println(s);
}
```

- Count the number of documents in a collection. . .

```
coll.getCount();
```

Working with Collections

Collections are MongoDB “tables”...

- List all of the collections in a database...

```
Set<String> colls = db.getCollectionNames();
for (String s :  colls) {
            System.out.println(s);
}
```

- Get a specific collection (lazy)...

```
Set<String> colls = db.getCollectionNames();
for (String s :  colls) {
            System.out.println(s);
}
```

- Count the number of documents in a collection...

```
coll.getCount();
```

MongoDB Documents
BSON

- “Documents” are MongoDB’s “rows”.
- MongoDB’s Internal Document representation is ’**BSON**’
  - **BSON** is a binary optimized flavor of **JSON**
  - Corrects **JSON**’s inefficiency in string encoding (Base64)
  - Supports extras including Regular Expressions, Byte Arrays, DateTimes & Timestamps, as well as datatypes for Javascript code blocks & functions.
  - Creative Commons licensed.
  - **BSON** implementation being split into its own package in most drivers.

NOVUS™

MongoDB Documents
BSON

- “Documents” are MongoDB’s “rows”.
- MongoDB’s Internal Document representation is ’**BSON**’
  - **BSON** is a binary optimized flavor of **JSON**
  - Corrects **JSON**’s inefficiency in string encoding (Base64)
  - Supports extras including Regular Expressions, Byte Arrays, DateTimes & Timestamps, as well as datatypes for Javascript code blocks & functions.
  - Creative Commons licensed.
  - **BSON** implementation being split into its own package in most drivers.

MongoDB Documents

BSON

- “Documents” are MongoDB’s “rows”.
- MongoDB’s Internal Document representation is ’**BSON**’
  - **BSON** is a binary optimized flavor of **JSON**
  - Corrects **JSON**’s inefficiency in string encoding (Base64)
  - Supports extras including Regular Expressions, Byte Arrays, DateTimes & Timestamps, as well as datatypes for Javascript code blocks & functions.
  - Creative Commons licensed.
  - **BSON** implementation being split into its own package in most drivers.

MongoDB Documents
BSON

- “Documents” are MongoDB’s “rows”.
- MongoDB’s Internal Document representation is ’**BSON**’
  - **BSON** is a binary optimized flavor of **JSON**
  - Corrects **JSON**’s inefficiency in string encoding (Base64)
  - Supports extras including Regular Expressions, Byte Arrays, DateTimes & Timestamps, as well as datatypes for Javascript code blocks & functions.
  - Creative Commons licensed.
  - **BSON** implementation being split into its own package in most drivers.

MongoDB Documents
BSON

- “Documents” are MongoDB’s “rows”.
- MongoDB’s Internal Document representation is ’**BSON**’
  - **BSON** is a binary optimized flavor of **JSON**
  - Corrects **JSON**’s inefficiency in string encoding (Base64)
  - Supports extras including Regular Expressions, Byte Arrays, DateTimes & Timestamps, as well as datatypes for Javascript code blocks & functions.
  - Creative Commons licensed.
  - **BSON** implementation being split into its own package in most drivers.

MongoDB Documents
BSON

- “Documents” are MongoDB’s “rows”.
- MongoDB’s Internal Document representation is ’**BSON**’
  - **BSON** is a binary optimized flavor of **JSON**
  - Corrects **JSON**’s inefficiency in string encoding (Base64)
  - Supports extras including Regular Expressions, Byte Arrays, DateTimes & Timestamps, as well as datatypes for Javascript code blocks & functions.
  - Creative Commons licensed.
  - **BSON** implementation being split into its own package in most drivers.

MongoDB Documents
BSON

- “Documents” are MongoDB’s “rows”.
- MongoDB’s Internal Document representation is ’**BSON**’
  - **BSON** is a binary optimized flavor of **JSON**
  - Corrects **JSON**’s inefficiency in string encoding (Base64)
  - Supports extras including Regular Expressions, Byte Arrays, DateTimes & Timestamps, as well as datatypes for Javascript code blocks & functions.
  - Creative Commons licensed.
  - **BSON** implementation being split into its own package in most drivers.

MongoDB Documents

From Java: BasicDBObject

- Java representation of **BSON** is the map-like **DBObject** (Java 2.0 driver has a new base class of **BSONObject** related to the **BSON** split-off)
- Easiest way to work with Mongo Documents is BasicDBObject…
  - BasicDBObject implements **java.util.LinkedHashMap<String, Object>**
  - Mutable object
  - Can take a **Map** as a constructor parameter
  - Example: `DBObject doc = new BasicDBObject(); doc.put("username", "bwmcadams"); doc.put("password", "MongoNYC");`
  - *toString* returns a JSON serialization.
- Use **BasicDBList** (implements **java.util.ArrayList<Object>**) to represent Arrays.

MongoDB Documents

From Java: BasicDBObject

- Java representation of **BSON** is the map-like **DBObject** (Java 2.0 driver has a new base class of **BSONObject** related to the **BSON** split-off)
- Easiest way to work with Mongo Documents is BasicDBObject. . .
  - **BasicDBObject** implements **java.util.LinkedHashMap<String, Object>**
  - Mutable object
  - Can take a **Map** as a constructor parameter
  - **Example:** `DBObject doc = new BasicDBObject(); doc.put("username", "bwmcadams");`
    `doc.put("password", "MongoNYC");`
  - *toString* returns a **JSON** serialization.
- Use **BasicDBList** (implements **java.util.ArrayList<Object>**) to represent Arrays.

MongoDB Documents

From Java: BasicDBObject

- Java representation of **BSON** is the map-like **DBObject** (Java 2.0 driver has a new base class of **BSONObject** related to the **BSON** split-off)
- Easiest way to work with Mongo Documents is BasicDBObject. . .
  - **BasicDBObject** implements **java.util.LinkedHashMap<String, Object>**
  - Mutable object
  - Can take a **Map** as a constructor parameter
  - Example: `DBObject doc = new BasicDBObject();`
    ```
    doc.put("username", "bwmcadams");
    doc.put("password", "MongoNYC");
    ```
  - *toString* returns a **JSON** serialization.
- Use **BasicDBList** (implements **java.util.ArrayList<Object>**) to represent Arrays.

MongoDB Documents
From Java: BasicDBObject

- Java representation of **BSON** is the map-like **DBObject** (Java 2.0 driver has a new base class of **BSONObject** related to the **BSON** split-off)
- Easiest way to work with Mongo Documents is BasicDBObject. . .
  - **BasicDBObject** implements **java.util.LinkedHashMap<String, Object>**
  - Mutable object
  - Can take a **Map** as a constructor parameter
  - Example: `DBObject doc = new BasicDBObject();`
    `doc.put("username", "bwmcadams");`
    `doc.put("password", "MongoNYC");`
  - *toString* returns a **JSON** serialization.
- Use **BasicDBList** (implements **java.util.ArrayList<Object>**) to represent Arrays.

MongoDB Documents
From Java: BasicDBObject

- Java representation of **BSON** is the map-like **DBObject** (Java 2.0 driver has a new base class of **BSONObject** related to the **BSON** split-off)
- Easiest way to work with Mongo Documents is BasicDBObject. . .
  - **BasicDBObject** implements **java.util.LinkedHashMap<String, Object>**
  - Mutable object
  - Can take a **Map** as a constructor parameter
  - Example: `DBObject doc = new BasicDBObject();`
    `doc.put("username", "bwmcadams");`
    `doc.put("password", "MongoNYC");`
  - *toString* returns a **JSON** serialization.
- Use **BasicDBList** (implements **java.util.ArrayList<Object>**) to represent Arrays.

MongoDB Documents
From Java: BasicDBObject

- Java representation of **BSON** is the map-like **DBObject** (Java 2.0 driver has a new base class of **BSONObject** related to the **BSON** split-off)
- Easiest way to work with Mongo Documents is BasicDBObject. . .
  - **BasicDBObject** implements **java.util.LinkedHashMap<String, Object>**
  - Mutable object
  - Can take a **Map** as a constructor parameter
  - Example: `DBObject doc = new BasicDBObject();`
    `doc.put("username", "bwmcadams");`
    `doc.put("password", "MongoNYC");`
  - *toString* returns a **JSON** serialization.
- Use **BasicDBList** (implements **java.util.ArrayList<Object>**) to represent Arrays.

MongoDB Documents

From Java: BasicDBObject

- Java representation of **BSON** is the map-like **DBObject** (Java 2.0 driver has a new base class of **BSONObject** related to the **BSON** split-off)
- Easiest way to work with Mongo Documents is BasicDBObject…
    - **BasicDBObject** implements **java.util.LinkedHashMap<String, Object>**
    - Mutable object
    - Can take a **Map** as a constructor parameter
    - Example: `DBObject doc = new BasicDBObject();`

      ```
      doc.put("username", "bwmcadams");
      doc.put("password", "MongoNYC");
      ```
    - *toString* returns a **JSON** serialization.
- Use **BasicDBList** (implements **java.util.ArrayList<Object>**) to represent Arrays.

MongoDB Documents
From Java: BasicDBObject

- Java representation of **BSON** is the map-like **DBObject** (Java 2.0 driver has a new base class of **BSONObject** related to the **BSON** split-off)
- Easiest way to work with Mongo Documents is BasicDBObject. . .
    - **BasicDBObject** implements **java.util.LinkedHashMap<String, Object>**
    - Mutable object
    - Can take a **Map** as a constructor parameter
    - Example: `DBObject doc = new BasicDBObject();`
      ```
      doc.put("username", "bwmcadams");
      doc.put("password", "MongoNYC");
      ```
    - *toString* returns a **JSON** serialization.
- Use **BasicDBList** (implements **java.util.ArrayList<Object>**) to represent Arrays.

MongoDB Documents
From Java: BasicDBObject

- Java representation of **BSON** is the map-like **DBObject** (Java 2.0 driver has a new base class of **BSONObject** related to the **BSON** split-off)
- Easiest way to work with Mongo Documents is BasicDBObject. . .
  - **BasicDBObject** implements **java.util.LinkedHashMap<String, Object>**
  - Mutable object
  - Can take a **Map** as a constructor parameter
  - Example:
    ```
    DBObject doc = new BasicDBObject();
    doc.put("username", "bwmcadams");
    doc.put("password", "MongoNYC");
    ```
  - *toString* returns a **JSON** serialization.
- Use **BasicDBList** (implements **java.util.ArrayList<Object>**) to represent Arrays.

MongoDB Documents
From Java: BasicDBObjectbuilder

For those who prefer immutability. . .

- **BasicDBObjectBuilder** follows the *Builder* pattern.
- *add()* your keys & values:
```
  BasicDBObjectBuilder builder = BasicDBObjectBuilder.start();
builder.add("username", "bwmcadams");
builder.add("password", "MongoNYC");

builder.add("presentation", "Java Development with MongoDB");
```
- A **BasicDBObjectBuilder** is not a **DBObject.**
- Call *get()* to return the built-up **DBObject**.
- *add()* returns itself so you can chain calls instead:
```
BasicDBOBjectBuilder.start().add("username", "bwmcadams").add("password",

"MongoNYC").add("presentation", "Java Development with MongoDB").get();
```

MongoDB Documents
From Java: BasicDBObjectbuilder

For those who prefer immutability…

- **BasicDBObjectBuilder** follows the *Builder* pattern.
- *add()* your keys & values:

```
  BasicDBObjectBuilder builder = BasicDBObjectBuilder.start();
builder.add("username", "bwmcadams");
builder.add("password", "MongoNYC");

builder.add("presentation", "Java Development with MongoDB");
```

- A **BasicDBObjectBuilder** is not a **DBObject**.
- Call *get()* to return the built-up **DBObject**.
- *add()* returns itself so you can chain calls instead:

```
BasicDBOBjectBuilder.start().add("username", "bwmcadams").add("password",

"MongoNYC").add("presentation", "Java Development with MongoDB").get();
```

NOVUS™

MongoDB Documents
From Java: BasicDBObjectbuilder

For those who prefer immutability…

- **BasicDBObjectBuilder** follows the *Builder* pattern.
- *add()* your keys & values:

```
 BasicDBObjectBuilder builder = BasicDBObjectBuilder.start();
builder.add("username", "bwmcadams");
builder.add("password", "MongoNYC");

builder.add("presentation", "Java Development with MongoDB");
```

- A **BasicDBObjectBuilder** is not a **DBObject**.
- Call *get()* to return the built-up **DBObject**.
- *add()* returns itself so you can chain calls instead:

```
BasicDBObjectBuilder.start().add("username", "bwmcadams").add("password",

"MongoNYC").add("presentation", "Java Development with MongoDB").get();
```

MongoDB Documents
From Java: BasicDBObjectbuilder

For those who prefer immutability. . .

- **BasicDBObjectBuilder** follows the *Builder* pattern.
- *add()* your keys & values:

```
  BasicDBObjectBuilder builder = BasicDBObjectBuilder.start();
builder.add("username", "bwmcadams");
builder.add("password", "MongoNYC");

builder.add("presentation", "Java Development with MongoDB");
```

- A **BasicDBObjectBuilder** is not a **DBObject.**
- Call *get()* to return the built-up **DBObject.**
- *add()* returns itself so you can chain calls instead:

```
BasicDBOBjectBuilder.start().add("username", "bwmcadams").add("password",

"MongoNYC").add("presentation", "Java Development with MongoDB").get();
```

MongoDB Documents
From Java: BasicDBObjectbuilder

For those who prefer immutability…

- **BasicDBObjectBuilder** follows the *Builder* pattern.
- *add()* your keys & values:

```
  BasicDBObjectBuilder builder = BasicDBObjectBuilder.start();
builder.add("username", "bwmcadams");
builder.add("password", "MongoNYC");

builder.add("presentation", "Java Development with MongoDB");
```

- A **BasicDBObjectBuilder** is not a **DBObject.**
- Call *get()* to return the built-up **DBObject.**
- *add()* returns itself so you can chain calls instead:

```
BasicDBOBjectBuilder.start().add("username", "bwmcadams").add("password",

"MongoNYC").add("presentation", "Java Development with MongoDB").get();
```

MongoDB Documents
Implementation by Extension: DBObject

- If you want to create your own concrete objects, extend & implement **DBObject**
  - Requires you implement a map-like interface including ability to get & set fields by key (even if you don't use them, required to deserialize)
  - Instances of **DBObject** can be saved directly to MongoDB.
- Feeling Fancy? Reflect instead…
  - Use **ReflectionDBObject** as a base class for your *Beans*.
  - **ReflectionDBObject** uses reflection to proxy your getters & setters and behave like a DBObject.
  - Downside: With Java's single inheritance you are stuck using this as your base class.
- Existing Object Model? ORM-Like Solutions (*Detailed later*)…
  - Daybreak
  - Morphia

MongoDB Documents
Implementation by Extension: DBObject

- If you want to create your own concrete objects, extend & implement **DBObject**
  - Requires you implement a map-like interface including ability to get & set fields by key (even if you don't use them, required to deserialize)
  - Instances of **DBObject** can be saved directly to MongoDB.
- Feeling Fancy? Reflect instead. . .
  - Use **ReflectionDBObject** as a base class for your *Beans*.
  - **ReflectionDBObject** uses reflection to proxy your getters & setters and behave like a DBObject.
  - Downside: With Java's single inheritance you are stuck using this as your base class.
- Existing Object Model? ORM-Like Solutions (*Detailed later*). . .
  - Daybreak
  - Morphia

NOVUS™

MongoDB Documents

Implementation by Extension: DBObject

- If you want to create your own concrete objects, extend & implement **DBObject**
  - Requires you implement a map-like interface including ability to get & set fields by key (even if you don't use them, required to deserialize)
  - Instances of **DBObject** can be saved directly to MongoDB.
- Feeling Fancy? Reflect instead. . .
  - Use **ReflectionDBObject** as a base class for your *Beans*.
  - **ReflectionDBObject** uses reflection to proxy your getters & setters and behave like a DBObject.
  - Downside: With Java's single inheritance you are stuck using this as your base class.
- Existing Object Model? ORM-Like Solutions (*Detailed later*). . .
  - Daybreak
  - Morphia

MongoDB Documents

Implementation by Extension: DBObject

- If you want to create your own concrete objects, extend & implement **DBObject**
  - Requires you implement a map-like interface including ability to get & set fields by key (even if you don't use them, required to deserialize)
  - Instances of **DBObject** can be saved directly to MongoDB.
- Feeling Fancy? Reflect instead...
  - Use **ReflectionDBObject** as a base class for your *Beans*.
  - **ReflectionDBObject** uses reflection to proxy your getters & setters and behave like a DBObject.
  - Downside: With Java's single inheritance you are stuck using this as your base class.
- Existing Object Model? ORM-Like Solutions (*Detailed later*)...
  - Daybreak
  - Morphia

MongoDB Documents
Implementation by Extension: DBObject

- If you want to create your own concrete objects, extend & implement **DBObject**
  - Requires you implement a map-like interface including ability to get & set fields by key (even if you don't use them, required to deserialize)
  - Instances of **DBObject** can be saved directly to MongoDB.
- Feeling Fancy? Reflect instead. . .
  - Use **ReflectionDBObject** as a base class for your *Beans*.
  - **ReflectionDBObject** uses reflection to proxy your getters & setters and behave like a DBObject.
  - Downside: With Java's single inheritance you are stuck using this as your base class.
- Existing Object Model? ORM-Like Solutions (*Detailed later*). . .
  - Daybreak
  - Morphia

MongoDB Documents
Implementation by Extension: DBObject

- If you want to create your own concrete objects, extend & implement **DBObject**
  - Requires you implement a map-like interface including ability to get & set fields by key (even if you don't use them, required to deserialize)
  - Instances of **DBObject** can be saved directly to MongoDB.
- Feeling Fancy? Reflect instead. . .
  - Use **ReflectionDBObject** as a base class for your *Beans*.
  - **ReflectionDBObject** uses reflection to proxy your getters & setters and behave like a DBObject.
  - Downside: With Java's single inheritance you are stuck using this as your base class.
- Existing Object Model? ORM-Like Solutions (*Detailed later*). . .
  - Daybreak
  - Morphia

MongoDB Documents
Implementation by Extension: DBObject

- If you want to create your own concrete objects, extend & implement **DBObject**
  - Requires you implement a map-like interface including ability to get & set fields by key (even if you don't use them, required to deserialize)
  - Instances of **DBObject** can be saved directly to MongoDB.
- Feeling Fancy? Reflect instead. . .
  - Use **ReflectionDBObject** as a base class for your *Beans*.
  - **ReflectionDBObject** uses reflection to proxy your getters & setters and behave like a DBObject.
  - Downside: With Java's single inheritance you are stuck using this as your base class.
- Existing Object Model? ORM-Like Solutions (*Detailed later*). . .
  - Daybreak
  - Morphia

MongoDB Documents
Implementation by Extension: DBObject

- If you want to create your own concrete objects, extend & implement **DBObject**
  - Requires you implement a map-like interface including ability to get & set fields by key (even if you don't use them, required to deserialize)
  - Instances of **DBObject** can be saved directly to MongoDB.
- Feeling Fancy? Reflect instead. . .
  - Use **ReflectionDBObject** as a base class for your *Beans*.
  - **ReflectionDBObject** uses reflection to proxy your getters & setters and behave like a DBObject.
  - Downside: With Java's single inheritance you are stuck using this as your base class.
- Existing Object Model? ORM-Like Solutions (*Detailed later*). . .
  - Daybreak
  - Morphia

MongoDB Documents

Implementation by Extension: DBObject

- If you want to create your own concrete objects, extend & implement **DBObject**
  - Requires you implement a map-like interface including ability to get & set fields by key (even if you don't use them, required to deserialize)
  - Instances of **DBObject** can be saved directly to MongoDB.
- Feeling Fancy? Reflect instead. . .
  - Use **ReflectionDBObject** as a base class for your *Beans*.
  - **ReflectionDBObject** uses reflection to proxy your getters & setters and behave like a DBObject.
  - Downside: With Java's single inheritance you are stuck using this as your base class.
- Existing Object Model? ORM-Like Solutions (*Detailed later*). . .
  - Daybreak
  - Morphia

MongoDB Documents
Implementation by Extension: DBObject

- If you want to create your own concrete objects, extend & implement **DBObject**
  - Requires you implement a map-like interface including ability to get & set fields by key (even if you don't use them, required to deserialize)
  - Instances of **DBObject** can be saved directly to MongoDB.
- Feeling Fancy? Reflect instead. . .
  - Use **ReflectionDBObject** as a base class for your *Beans*.
  - **ReflectionDBObject** uses reflection to proxy your getters & setters and behave like a DBObject.
  - Downside: With Java's single inheritance you are stuck using this as your base class.
- Existing Object Model? ORM-Like Solutions (*Detailed later*). . .
  - Daybreak
  - Morphia

Outline

1 MongoDB + Java Basics
- Setting up your Java environment
- Connecting to MongoDB

2 Working with MongoDB
- Collections + Documents
- Inserting Documents to MongoDB
- Querying MongoDB

Inserting Documents

- One at a time:

```
DBObject doc = BasicDBObjectBuilder.start().add("username",
"bwmcadams").add("password", "MongoNYC").add("presentation", "Java Development
with MongoDB").get();
// DBCollection coll
coll.insert(doc);
```

- Got multiple documents? Call *insert()* in a loop, or pass **DBObject[]** or **List<DBObject>**
- Three ways to store your documents:
  - INSERT (*insert()*) always attempts to add a new row.
  - SAVE (*save()*) only attempts to insert unless *_id* is defined. Otherwise, it will attempt to update the identified document.
  - UPDATE (*update()*) Allows you to pass a query to filter by and the fields to change. Boolean option "multi" specifies if multiple documents should be updated. Boolean "upsert" specifies that the object should be inserted if it doesn't exist (e.g. query doesn't match).

Inserting Documents

- One at a time:

```
DBObject doc = BasicDBObjectBuilder.start().add("username",
"bwmcadams").add("password", "MongoNYC").add("presentation", "Java Development
with MongoDB").get();
// DBCollection coll
coll.insert(doc);
```

- Got multiple documents? Call *insert()* in a loop, or pass **DBObject[]** or **List<DBObject>**
- Three ways to store your documents:
  - INSERT (*insert()*) always attempts to add a new row.
  - SAVE (*save()*) only attempts to insert unless *_id* is defined. Otherwise, it will attempt to update the identified document.
  - UPDATE (*update()*) Allows you to pass a query to filter by and the fields to change. Boolean option "multi" specifies if multiple documents should be updated. Boolean "upsert" specifies that the object should be inserted if it doesn't exist (e.g. query doesn't match).

Inserting Documents

- One at a time:

```
DBObject doc = BasicDBObjectBuilder.start().add("username",
"bwmcadams").add("password", "MongoNYC").add("presentation", "Java Development
with MongoDB").get();
// DBCollection coll
coll.insert(doc);
```

- Got multiple documents? Call *insert()* in a loop, or pass **DBObject[]** or **List<DBObject>**
- Three ways to store your documents:
  - INSERT (*insert()*) always attempts to add a new row.
  - SAVE (*save()*) only attempts to insert unless *_id* is defined. Otherwise, it will attempt to update the identified document.
  - UPDATE (*update()*) Allows you to pass a query to filter by and the fields to change. Boolean option "multi" specifies if multiple documents should be updated. Boolean "upsert" specifies that the object should be inserted if it doesn't exist (e.g. query doesn't match).

Inserting Documents

- One at a time:
```
DBObject doc = BasicDBOBjectBuilder.start().add("username",
"bwmcadams").add("password", "MongoNYC").add("presentation", "Java Development
with MongoDB").get();
// DBCollection coll
coll.insert(doc);
```
- Got multiple documents? Call *insert()* in a loop, or pass **DBObject[]** or **List<DBObject>**
- Three ways to store your documents:
  - INSERT (*insert()*) always attempts to add a new row.
  - SAVE (*save()*) only attempts to insert unless *_id* is defined. Otherwise, it will attempt to update the identified document.
  - UPDATE (*update()*) Allows you to pass a query to filter by and the fields to change. Boolean option "multi" specifies if multiple documents should be updated. Boolean "upsert" specifies that the object should be inserted if it doesn't exist (e.g. query doesn't match).

Inserting Documents

- One at a time:
```
DBObject doc = BasicDBObjectBuilder.start().add("username",
"bwmcadams").add("password", "MongoNYC").add("presentation", "Java Development
with MongoDB").get();
// DBCollection coll
coll.insert(doc);
```
- Got multiple documents? Call *insert()* in a loop, or pass **DBObject[]** or **List<DBObject>**
- Three ways to store your documents:
  - INSERT (*insert()*) always attempts to add a new row.
  - SAVE (*save()*) only attempts to insert unless *_id* is defined. Otherwise, it will attempt to update the identified document.
  - UPDATE (*update()*) Allows you to pass a query to filter by and the fields to change. Boolean option "multi" specifies if multiple documents should be updated. Boolean "upsert" specifies that the object should be inserted if it doesn't exist (e.g. query doesn't match).

Inserting Documents

- One at a time:
```
DBObject doc = BasicDBObjectBuilder.start().add("username",
"bwmcadams").add("password", "MongoNYC").add("presentation", "Java Development
with MongoDB").get();
// DBCollection coll
coll.insert(doc);
```
- Got multiple documents? Call *insert()* in a loop, or pass **DBObject[]** or **List<DBObject>**
- Three ways to store your documents:
  - INSERT (*insert()*) always attempts to add a new row.
  - SAVE (*save()*) only attempts to insert unless *_id* is defined. Otherwise, it will attempt to update the identified document.
  - UPDATE (*update()*) Allows you to pass a query to filter by and the fields to change. Boolean option "multi" specifies if multiple documents should be updated. Boolean "upsert" specifies that the object should be inserted if it doesn't exist (e.g. query doesn't match).

Outline

1 MongoDB + Java Basics
- Setting up your Java environment
- Connecting to MongoDB

2 Working with MongoDB
- Collections + Documents
- Inserting Documents to MongoDB
- Querying MongoDB

NOVUS™

MongoDB Querying
Implementation by Extension: DBObject

- Find a single row with *findOne()*. Takes the first row returned.
- Getting a cursor of all documents (*find()* with no query):

```
DBCursor cur = coll.find();
while (cur.hasNext()) {
          System.out.println(cur.next());
}
```

MongoDB Querying
Implementation by Extension: DBObject

- Find a single row with *findOne()*. Takes the first row returned.
- Getting a cursor of all documents (*find()* with no query):

```
DBCursor cur = coll.find();
while (cur.hasNext()) {
         System.out.println(cur.next());
}
```