## 1. MongoDB Java Driver Download

If you have maven project, just add below dependency to include MongoDB java driver into your application.

```
<dependency>
	<groupId>org.mongodb</groupId>
	<artifactId>mongo-java-driver</artifactId>
	<version>2.12.3</version>
</dependency>
```

If you have a standalone project, you can download MongoDB Java Driver from this link and include it in your project build path.

Now let's go through some basic usage of MongoDB java driver and then we will look into **MongoDB Java Example** program for CRUD operations.

## 2. Creating MongoDB Java Connection

**MongoClient** is the interface between our java program and MongoDB server. `MongoClient` is used to create connection, connect to database, retrieve collection names and create/read/update/delete database, collections, document etc.
One of the MongoDB java driver feature I like most is that it's thread safe, so we can create an instance of `MongoClient` once and reuse it. Even if multiple thread accesses it simultaneously, a connection is returned from the internal connection pool maintained by it.

For every request to the database (find, insert etc) the Java thread will obtain a connection from the pool, execute the operation, and release the connection. This means the connection (socket) used may be different each time.

Below are some of the common methods to connect to a MongoDB server.

```
MongoClient mongoClient = new MongoClient(); //connects to
default host and port i.e 127.0.0.1:27017
// or
MongoClient mongoClient = new MongoClient( "localhost" );
//connects to default port i.e 27017
// or
MongoClient mongoClient = new MongoClient( "localhost" ,
27017 ); // should use this always
```

```
// or, to connect to a replica set, with auto-discovery of
the primary
MongoClient mongoClient = new MongoClient(Arrays.asList(new
ServerAddress("localhost", 27017),
                                        new
ServerAddress("localhost", 27018),
                                        new
ServerAddress("localhost", 27019)));
```

### 3. Connection to MongoDB Database

Once we get the connection to MongoDB server, next step is to create the connection to the database, as shown below. Note that if database is not present, MongoDB will create it for you.

```
MongoClient mongo = new MongoClient("localhost", 27017);
DB db = mongo.getDB("journaldev");
```

MongoClient provide a useful method to get all the database names, as shown below.

```
MongoClient mongo = new MongoClient("localhost", 27017);
List<String> dbs = mongo.getDatabaseNames();
System.out.println(dbs); // [journaldev, local, admin]
```

We can have user-password based authentication for databases, in that case we need to provide authorization credentials like below.

```
MongoCredential journaldevAuth =
MongoCredential.createPlainCredential("pankaj", "journaldev",
"pankaj123".toCharArray());
MongoCredential testAuth =
MongoCredential.createPlainCredential("pankaj", "test",
"pankaj123".toCharArray());
List<MongoCredential> auths = new
ArrayList<MongoCredential>();
auths.add(journaldevAuth);
auths.add(testAuth);

ServerAddress serverAddress = new ServerAddress("localhost",
27017);
MongoClient mongo = new MongoClient(serverAddress, auths);
```

If you are using older versions, you need to provide authentication details after getting the database object like below.

```
MongoClient mongo = new MongoClient("localhost", 27017);
DB db = mongo.getDB("journaldev");
boolean auth = db.authenticate("pankaj",
"pankaj123".toCharArray());
```

You can easily figure out flaws in the earlier approach, the authentication should be done at early stage because we can't recover from it.

We can drop a database either by using `MongoClient` `dropDatabase(String db)` method or by `DB` `dropDatabase()` method. Since we are dropping the database, i prefer to use MongoClient method.

## 4. MongoDB and Collections

Every database can have zero or multiple collections, they are like tables in relational database servers except that you don't have specific format of data. Think of it like a generic list vs list of Strings in terms of java programming language.

We can get all the collections names using below code.

```
MongoClient mongo = new MongoClient("localhost", 27017);
DB db = mongo.getDB("journaldev");

Set<String> collections = db.getCollectionNames();
System.out.println(collections); // [datas, names,
system.indexes, users]
```

We can get a specific collection by providing it's name, as shown below.

```
DB db = mongo.getDB("journaldev");
DBCollection col = db.getCollection("users");
```

Again if the collection doesn't exist, MongoDB will create it for you. All the data in MongoDB goes into some collection, so at this point we are ready to perform insert/update/delete operations.

We can use `DBCollection` `drop()` method to drop a collection from the database.

## 5. MongoDB Java Example

Even though we can work on any valid JSON document in MongoDB collection, in real life we have POJO classes that are mapped with these data. So I will create a java bean and use it for my examples.

User.java

```
package com.journaldev.mongodb.model;

public class User {

	private int id;
	private String name;
	private String role;
	private boolean isEmployee;

	public int getId() {
		return id;
	}
	public void setId(int id) {
		this.id = id;
	}
	public String getName() {
		return name;
	}
	public void setName(String name) {
		this.name = name;
	}
	public String getRole() {
		return role;
	}
	public void setRole(String role) {
		this.role = role;
	}
	public boolean isEmployee() {
		return isEmployee;
	}
	public void setEmployee(boolean isEmployee) {
		this.isEmployee = isEmployee;
	}
}
```

Here is the complete MongoDB java example program showing all the CRUD operations one by one.

MongoDBExample.java

```
package com.journaldev.mongodb.main;

import java.net.UnknownHostException;
```

```
import com.journaldev.mongodb.model.User;
import com.mongodb.BasicDBObjectBuilder;
import com.mongodb.DB;
import com.mongodb.DBCollection;
import com.mongodb.DBCursor;
import com.mongodb.DBObject;
import com.mongodb.MongoClient;
import com.mongodb.WriteResult;

public class MongoDBExample {

	public static void main(String[] args) throws
UnknownHostException {

		User user = createUser();
		DBObject doc = createDBObject(user);

		MongoClient mongo = new MongoClient("localhost",
27017);
		DB db = mongo.getDB("journaldev");

		DBCollection col = db.getCollection("users");

		//create user
		WriteResult result = col.insert(doc);
		System.out.println(result.getUpsertedId());
		System.out.println(result.getN());
		System.out.println(result.isUpdateOfExisting());
		System.out.println(result.getLastConcern());

		//read example
		DBObject query =
BasicDBObjectBuilder.start().add("_id", user.getId()).get();
		DBCursor cursor = col.find(query);
		while(cursor.hasNext()){
			System.out.println(cursor.next());
		}

		//update example
		user.setName("Pankaj Kumar");
		doc = createDBObject(user);
		result = col.update(query, doc);
		System.out.println(result.getUpsertedId());
		System.out.println(result.getN());
		System.out.println(result.isUpdateOfExisting());
		System.out.println(result.getLastConcern());

		//delete example
		result = col.remove(query);
		System.out.println(result.getUpsertedId());
		System.out.println(result.getN());
```

```
            System.out.println(result.isUpdateOfExisting());
            System.out.println(result.getLastConcern());

            //close resources
            mongo.close();
      }

      private static DBObject createDBObject(User user) {
            BasicDBObjectBuilder docBuilder =
BasicDBObjectBuilder.start();

            docBuilder.append("_id", user.getId());
            docBuilder.append("name", user.getName());
            docBuilder.append("role", user.getRole());
            docBuilder.append("isEmployee",
user.isEmployee());
            return docBuilder.get();
      }

      private static User createUser() {
            User u = new User();
            u.setId(2);
            u.setName("Pankaj");
            u.setEmployee(true);
            u.setRole("CEO");
            return u;
      }


}
```

A sample execution results in following output.

```
null
0
false
WriteConcern { "getlasterror" : 1} / (Continue on error?
false)
{ "_id" : 2 , "name" : "Pankaj" , "role" : "CEO" ,
"isEmployee" : true}
null
1
true
WriteConcern { "getlasterror" : 1} / (Continue on error?
false)
null
1
false
```

```
WriteConcern { "getlasterror" : 1} / (Continue on error?
false)
```

Notice that I am saving User id with **_id** name, this is a reserved key for the primary key of any record in the collection. If we don't provide one, MongoDB will create one for us. It's like sequencer or auto increment column in relational database tables.

Since I am deleting the created record, further execution won't cause any issues. But if there are duplicate record, then we will get below errors.

```
Exception in thread "main"
com.mongodb.MongoException$DuplicateKey: { "serverUsed" :
"localhost:27017" , "ok" : 1 , "n" : 0 ,
 "err" : "insertDocument :: caused by :: 11000 E11000
duplicate key error index: journaldev.users.$_id_  dup key:
{ : 1 }" ,
"code" : 11000}
      at
com.mongodb.CommandResult.getWriteException(CommandResult.jav
a:88)
      at
com.mongodb.CommandResult.getException(CommandResult.java:79)
      at
com.mongodb.DBCollectionImpl.translateBulkWriteException(DBCo
llectionImpl.java:314)
      at
com.mongodb.DBCollectionImpl.insert(DBCollectionImpl.java:189
)
      at
com.mongodb.DBCollectionImpl.insert(DBCollectionImpl.java:165
)
      at
com.mongodb.DBCollection.insert(DBCollection.java:93)
      at
com.mongodb.DBCollection.insert(DBCollection.java:78)
      at
com.mongodb.DBCollection.insert(DBCollection.java:120)
      at
com.journaldev.mongodb.main.MongoDBExample.main(MongoDBExampl
e.java:27)
```