**Title: Java with MongoDB - Libary systems**

**Module #1 : - Book entries**

**Instructions: step-by-step approach.**

**Prerequisites:**

1. **MongoDB Java Driver**: Make sure to add the MongoDB Java Driver dependency in your project. You can use Maven for that:

<dependency>

<groupId>org.mongodb</groupId>

<artifactId>mongodb-driver-sync</artifactId>

<version>4.5.0</version>

</dependency>

1. **MongoDB Setup**: Ensure you have MongoDB running on your system or use a cloud MongoDB service like Atlas.

**Step 1: Set up MongoDB Connection**

First, create a MongoDB connection. We'll use the MongoClient to interact with the database.

import com.mongodb.client.\*;

import org.bson.Document;

public class MongoDBConnection {

private static final String DB\_URI = "mongodb://localhost:27017"; // Adjust for your MongoDB setup

private static final String DB\_NAME = "library";

private static final String COLLECTION\_NAME = "books";

private MongoCollection<Document> booksCollection;

public MongoDBConnection() {

MongoClient mongoClient = MongoClients.create(DB\_URI);

MongoDatabase database = mongoClient.getDatabase(DB\_NAME);

this.booksCollection = database.getCollection(COLLECTION\_NAME);

}

public void insertBook(String title, String author, String genre, String publisher, int year) {

Document book = new Document("title", title)

.append("author", author)

.append("genre", genre)

.append("publisher", publisher)

.append("year", year);

booksCollection.insertOne(book);

}

}

**Step 2: Create the Java Form (Swing GUI)**

Now, create the form for input using Java Swing. The form will take book details such as title, author, genre, publisher, and year.

import javax.swing.\*;

import java.awt.\*;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

public class LibraryBookForm extends JFrame {

private JTextField titleField, authorField, genreField, publisherField, yearField;

private MongoDBConnection dbConnection;

public LibraryBookForm() {

dbConnection = new MongoDBConnection();

setTitle("Library Book Entry");

setSize(400, 300);

setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

setLocationRelativeTo(null);

// Create the form

JPanel panel = new JPanel();

panel.setLayout(new GridLayout(6, 2, 10, 10));

JLabel titleLabel = new JLabel("Book Title:");

titleField = new JTextField();

JLabel authorLabel = new JLabel("Author:");

authorField = new JTextField();

JLabel genreLabel = new JLabel("Genre:");

genreField = new JTextField();

JLabel publisherLabel = new JLabel("Publisher:");

publisherField = new JTextField();

JLabel yearLabel = new JLabel("Year:");

yearField = new JTextField();

JButton submitButton = new JButton("Submit");

// Add components to panel

panel.add(titleLabel);

panel.add(titleField);

panel.add(authorLabel);

panel.add(authorField);

panel.add(genreLabel);

panel.add(genreField);

panel.add(publisherLabel);

panel.add(publisherField);

panel.add(yearLabel);

panel.add(yearField);

panel.add(submitButton);

add(panel);

// Submit button action

submitButton.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

String title = titleField.getText();

String author = authorField.getText();

String genre = genreField.getText();

String publisher = publisherField.getText();

int year = Integer.parseInt(yearField.getText());

dbConnection.insertBook(title, author, genre, publisher, year);

JOptionPane.showMessageDialog(null, "Book added to the library database!");

// Clear the form after submission

titleField.setText("");

authorField.setText("");

genreField.setText("");

publisherField.setText("");

yearField.setText("");

}

});

}

public static void main(String[] args) {

SwingUtilities.invokeLater(new Runnable() {

@Override

public void run() {

new LibraryBookForm().setVisible(true);

}

});

}

}

**Explanation:**

1. **MongoDB Connection**: In MongoDBConnection, we connect to MongoDB and use MongoCollection<Document> to interact with the database and insert documents into the collection.
2. **Form Design**: The LibraryBookForm class is a simple Swing application with fields for title, author, genre, publisher, and year.
3. **ActionListener**: When the user clicks the submit button, the data from the form is collected and inserted into the MongoDB database.

**Step 3: Run the Application**

Once you have everything set up:

1. Make sure MongoDB is running on your machine or is accessible via a cloud service.
2. Run the LibraryBookForm Java application.

**Step 4: Test the Application**

1. Launch the GUI, fill in the details of a book, and hit "Submit".
2. After submission, the book will be added to your MongoDB database in the library collection.

This is a basic example of integrating a MongoDB database with a Java Swing form for library books entry. You can extend this with features like form validation, better error handling, or even listing all the books currently in the database.

**Transacting library systems:**

**Title: Borrowing a book**

**Module#2: library transactions**

Instruction: step by step approach

**Prerequisite: mongodb and java connectivity**

To connect to MongoDB using Java, you will use the official MongoDB Java Driver. You can establish the connection using a connection string, with or without a build tool like Maven or Gradle.

Here are the general steps to connect to a MongoDB server from a Java application.

Step 1: Add the MongoDB Java Driver

If you are using a build automation tool, you can add the driver as a dependency in your project's configuration file.

For a Maven project, add the following dependency to your pom.xml:

xml

<dependency>

<groupId>org.mongodb</groupId>

<artifactId>mongodb-driver-sync</artifactId>

<version>5.2.0</version>

</dependency>

Use code with caution.

*Note: Always check the MongoDB documentation for the latest version.*

For a Gradle project, add this dependency to your build.gradle file:

groovy

implementation 'org.mongodb:mongodb-driver-sync:5.2.0'

Use code with caution.

Step 2: Create the connection string

A connection string (or URI) contains all the information your driver needs to connect to a MongoDB deployment. This includes the hostname, port, authentication details, and other options.

* For a local MongoDB instance, the connection string is typically mongodb://localhost:27017.
* For a MongoDB Atlas cluster, you can find the connection string by logging into your Atlas account, navigating to the database section, and clicking the "Connect" button for your cluster.

The connection string will look similar to this, and you must replace the placeholders for your username, password, and database:  
mongodb+srv://<username>:<password>@cluster0.abcde.mongodb.net/?retryWrites=true&w=majority

Step 3: Write the Java connection code

The MongoClient class is the main entry point for interacting with MongoDB from a Java application. It is a thread-safe object that represents a connection pool, so you only need one instance of it.

Here is a basic example of connecting to a MongoDB cluster:

java

import com.mongodb.client.MongoClient;

import com.mongodb.client.MongoClients;

import com.mongodb.client.MongoDatabase;

import org.bson.Document;

public class MongoDBConnection {

public static void main(String[] args) {

*// Replace with your actual connection string*

String connectionString = "mongodb://localhost:27017";

try (MongoClient mongoClient = MongoClients.create(connectionString)) {

*// Get the database (MongoDB creates it if it doesn't exist)*

MongoDatabase database = mongoClient.getDatabase("myTestDB");

*// Get a collection*

*// MongoDB will also create the collection if it doesn't exist*

var collection = database.getCollection("myCollection");

*// Example: Insert a document*

Document document = new Document("name", "John Doe").append("age", 30);

collection.insertOne(document);

System.out.println("Document inserted successfully!");

} catch (Exception e) {

e.printStackTrace();

}

}

}

Use code with caution.

Step 4: Run the application

1. Start your MongoDB server. If you are using a local instance, make sure it is running.
2. Ensure you have a JDK installed (version 8 or later is recommended).
3. Compile and run your Java class. The program will connect to your MongoDB instance, create the specified database and collection, and insert a document.

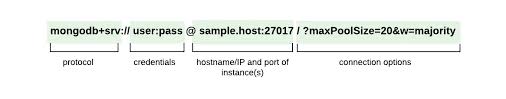
* Get Started with the Java Driver - Java Sync Driver v5.2 - MongoDB

Get Started with the Java Driver \* Introduction. \* Set up Your Project. \* Install the Java Development Kit (JDK) \* Create the Project. \* Add MongoDB as a Depend...

MongoDB

* Create a MongoClient - Java Sync Driver - MongoDB Docs

MongoClient. You can connect to and communicate with MongoDB using the MongoClient class. To create a MongoClientSettings object, use the MongoClientSettings. b...



book borrowing in java and mongodb

To create a book borrowing system using Java and MongoDB, you will need to

design your database schema, set up a connection, and implement methods for key library actions. MongoDB's document model allows for flexible and intuitive data structures, making it well-suited for this type of application.

Step 1: Design the MongoDB schema

You will need at least three collections to manage books, users, and borrowing transactions.

Books Collection (books)

Each document in this collection will represent a book.

json

{

"\_id": ObjectId("..."),

"title": "The Hitchhiker's Guide to the Galaxy",

"author": "Douglas Adams",

"isbn": "0345391802",

"status": "available" *// or "borrowed"*

}

**//note: Use code with caution.**

Users Collection (users)

This collection will store information about library members.

json

{

"\_id": ObjectId("..."),

"name": "Jane Doe",

"member\_id": "M-12345",

"borrowed\_books": [

*// This array can hold references to borrowed books*

*// It's a key feature of MongoDB's document model*

]

}

Use code with caution.

Borrowing Transactions Collection (transactions)

This collection will track the history of borrowed books.

json

{

"\_id": ObjectId("..."),

"book\_id": ObjectId("..."), *// Reference to the book*

"member\_id": ObjectId("..."), *// Reference to the user*

"borrowed\_date": ISODate("2025-09-01T10:00:00Z"),

"due\_date": ISODate("2025-09-15T10:00:00Z"),

"returned\_date": null, *// Will be updated on return*

"status": "active" *// or "returned" or "overdue"*

}

Use code with caution.

Step 2: Set up the Java project

1. Add the MongoDB Java Driver. If you're using Maven, include this dependency in your pom.xml:

xml

<dependency>

<groupId>org.mongodb</groupId>

<artifactId>mongodb-driver-sync</artifactId>

<version>5.2.0</version>

</dependency>

Use code with caution.

1. Establish a connection. Create a MongoClient instance to connect to your MongoDB deployment, as shown in the previous response.

Step 3: Implement core borrowing logic in Java

Here are Java methods to handle the main operations for a book borrowing system.

1. Borrow a book

This method finds an available book by its title, updates its status to "borrowed," and creates a new transaction document.

java

import com.mongodb.client.MongoClient;

import com.mongodb.client.MongoClients;

import com.mongodb.client.MongoCollection;

import com.mongodb.client.MongoDatabase;

import com.mongodb.client.result.UpdateResult;

import org.bson.Document;

import org.bson.types.ObjectId;

import java.time.Instant;

import static com.mongodb.client.model.Filters.and;

import static com.mongodb.client.model.Filters.eq;

import static com.mongodb.client.model.Updates.set;

public class LibraryService {

private final MongoCollection<Document> booksCollection;

private final MongoCollection<Document> transactionsCollection;

public LibraryService(MongoClient client) {

MongoDatabase database = client.getDatabase("libraryDB");

this.booksCollection = database.getCollection("books");

this.transactionsCollection = database.getCollection("transactions");

}

public void borrowBook(String bookTitle, ObjectId memberId) {

*// 1. Find an available book*

Document book = booksCollection.find(and(eq("title", bookTitle), eq("status", "available"))).first();

if (book == null) {

System.out.println("Book '" + bookTitle + "' is not available for borrowing.");

return;

}

*// 2. Update the book's status to 'borrowed'*

ObjectId bookId = book.getObjectId("\_id");

UpdateResult updateResult = booksCollection.updateOne(

eq("\_id", bookId),

set("status", "borrowed")

);

if (updateResult.getModifiedCount() > 0) {

*// 3. Create a new transaction document*

Document transaction = new Document("book\_id", bookId)

.append("member\_id", memberId)

.append("borrowed\_date", Instant.now())

.append("due\_date", Instant.now().plus(14, java.time.temporal.ChronoUnit.DAYS))

.append("status", "active");

transactionsCollection.insertOne(transaction);

System.out.println("Book '" + bookTitle + "' has been successfully borrowed.");

} else {

System.out.println("Could not process borrowing for '" + bookTitle + "'.");

}

}

}

Use code with caution.

2. Return a book

This method finds the active transaction for a given book, updates its status to "returned," and changes the book's status back to "available."

java

public void returnBook(String bookTitle, ObjectId memberId) {

*// 1. Find the book and the active transaction*

Document book = booksCollection.find(eq("title", bookTitle)).first();

if (book == null) {

System.out.println("Book not found in library catalog.");

return;

}

Document transaction = transactionsCollection.find(and(

eq("book\_id", book.getObjectId("\_id")),

eq("member\_id", memberId),

eq("status", "active")

)).first();

if (transaction == null) {

System.out.println("No active borrowing transaction found for this book and member.");

return;

}

*// 2. Update the transaction to 'returned' and set the return date*

UpdateResult transactionUpdateResult = transactionsCollection.updateOne(

eq("\_id", transaction.getObjectId("\_id")),

and(set("status", "returned"), set("returned\_date", Instant.now()))

);

*// 3. Update the book's status back to 'available'*

UpdateResult bookUpdateResult = booksCollection.updateOne(

eq("\_id", book.getObjectId("\_id")),

set("status", "available")

);

if (transactionUpdateResult.getModifiedCount() > 0 && bookUpdateResult.getModifiedCount() > 0) {

System.out.println("Book '" + bookTitle + "' has been successfully returned.");

} else {

System.out.println("An error occurred during the return process.");

}

}

Use code with caution.

4. Running the application

You can add these methods to a main class to create a working example.

java

public class LibraryApp {

public static void main(String[] args) {

try (MongoClient mongoClient = MongoClients.create("mongodb://localhost:27017")) {

LibraryService library = new LibraryService(mongoClient);

*// --- Setup (for testing) ---*

MongoDatabase database = mongoClient.getDatabase("libraryDB");

MongoCollection<Document> booksCollection = database.getCollection("books");

MongoCollection<Document> usersCollection = database.getCollection("users");

*// Insert a sample book*

Document sampleBook = new Document("title", "The Hobbit")

.append("author", "J.R.R. Tolkien")

.append("isbn", "9780261103347")

.append("status", "available");

booksCollection.insertOne(sampleBook);

System.out.println("Inserted sample book.");

*// Insert a sample user*

ObjectId memberId = new ObjectId();

Document sampleUser = new Document("\_id", memberId)

.append("name", "Alice")

.append("member\_id", "M-54321");

usersCollection.insertOne(sampleUser);

System.out.println("Inserted sample user.");

*// --- End Setup ---*

*// --- Run borrowing logic ---*

*// Borrow the book*

library.borrowBook("The Hobbit", memberId);

*// Try to borrow it again (should fail)*

library.borrowBook("The Hobbit", memberId);

*// Return the book*

library.returnBook("The Hobbit", memberId);

*// Borrow it again (should succeed)*

library.borrowBook("The Hobbit", memberId);

} catch (Exception e) {

e.printStackTrace();

}

}

}

**//Note: Use code with caution.**