Lesson 01 Demo 05

Inserting Documents Using MongoDB CRUD Operations

Objective: To insert documents in MongoDB using CRUD operations by creating a project and generating Getters and Setters

Tools required: Eclipse IDE

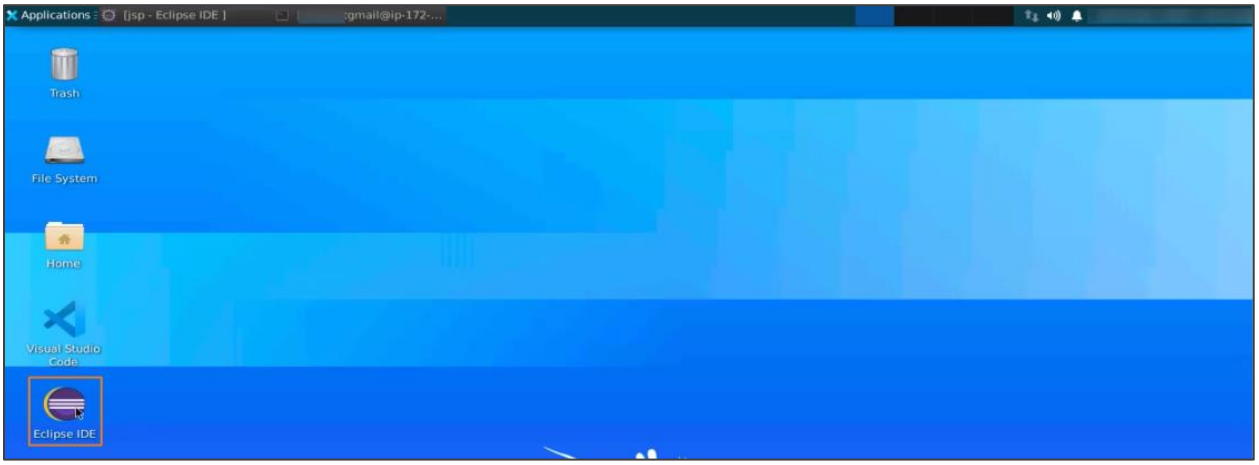
Prerequisites: None

Steps to be followed:

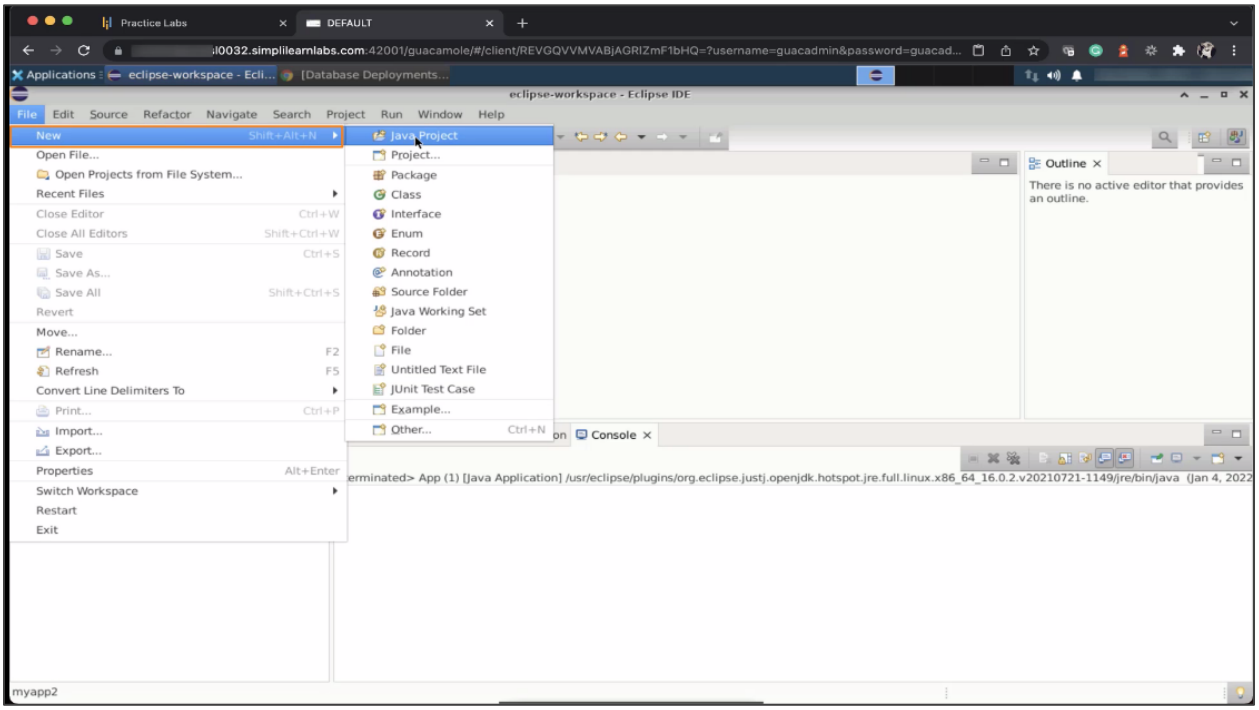
1. Create a Maven project
2. Add the dependencies and a print statement
3. Create a Customer and DBOperations class
4. Create a collection
5. Connect the database to the application
6. Create one new class
7. Generate constructor
8. Create a document
9. Generate Getters and Setters

Step 1: Create a Maven project

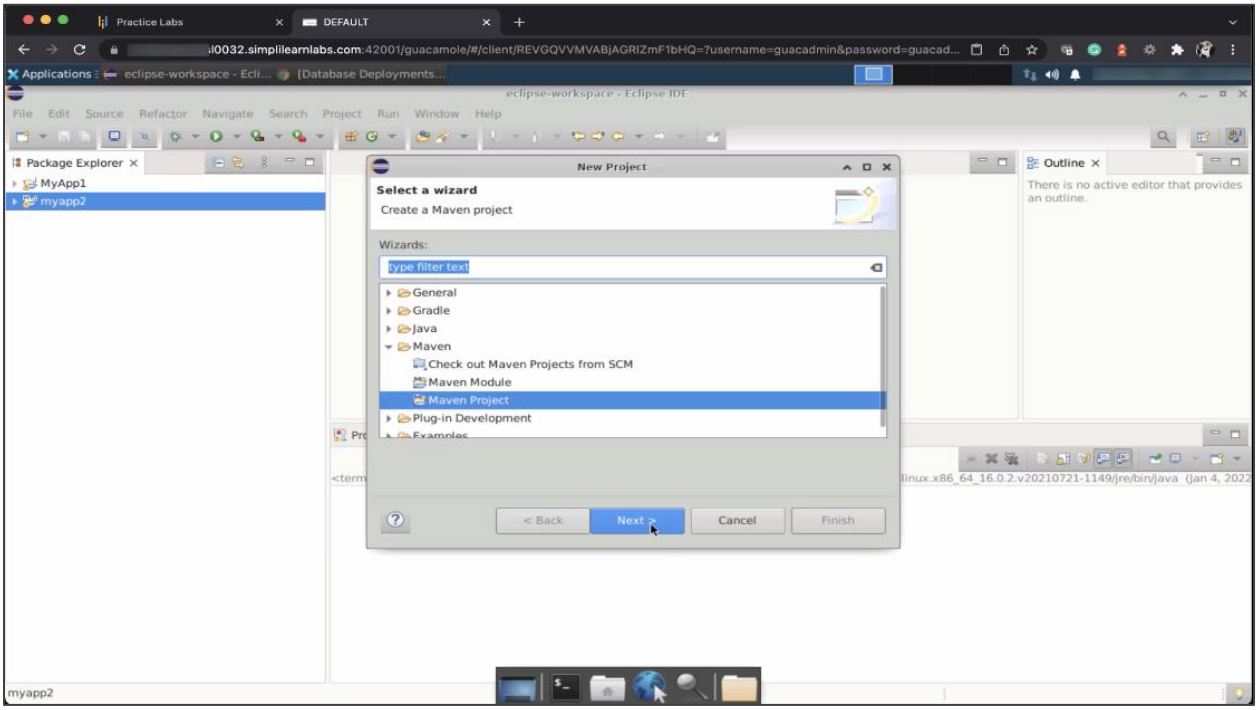
1.1 Open the **Eclipse IDE**



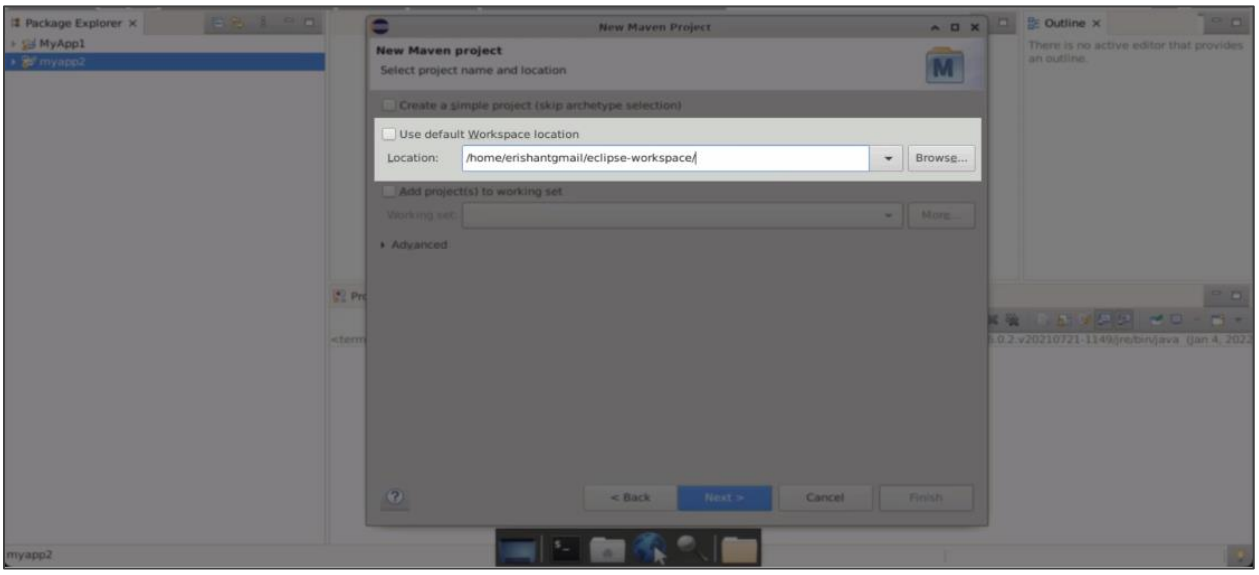
1.2 Create a new Maven project by clicking on **New** and selecting **Project**



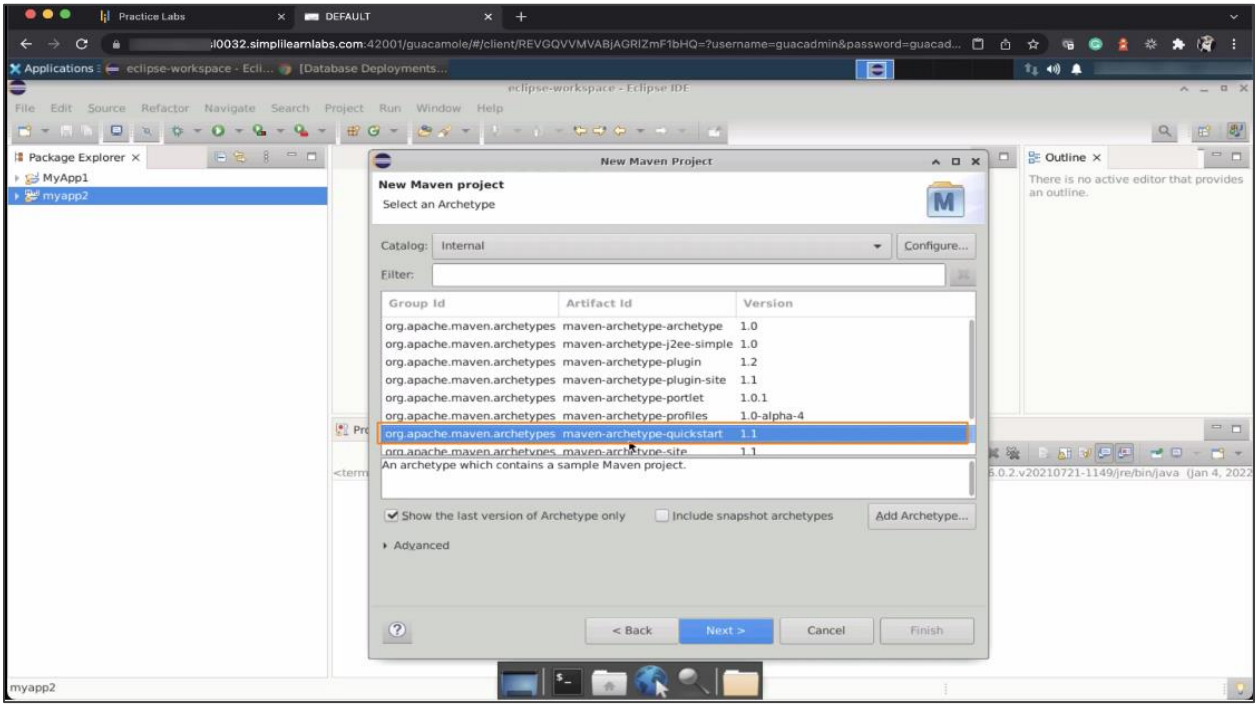
1.3 Select **Maven Project** and click on **Next**



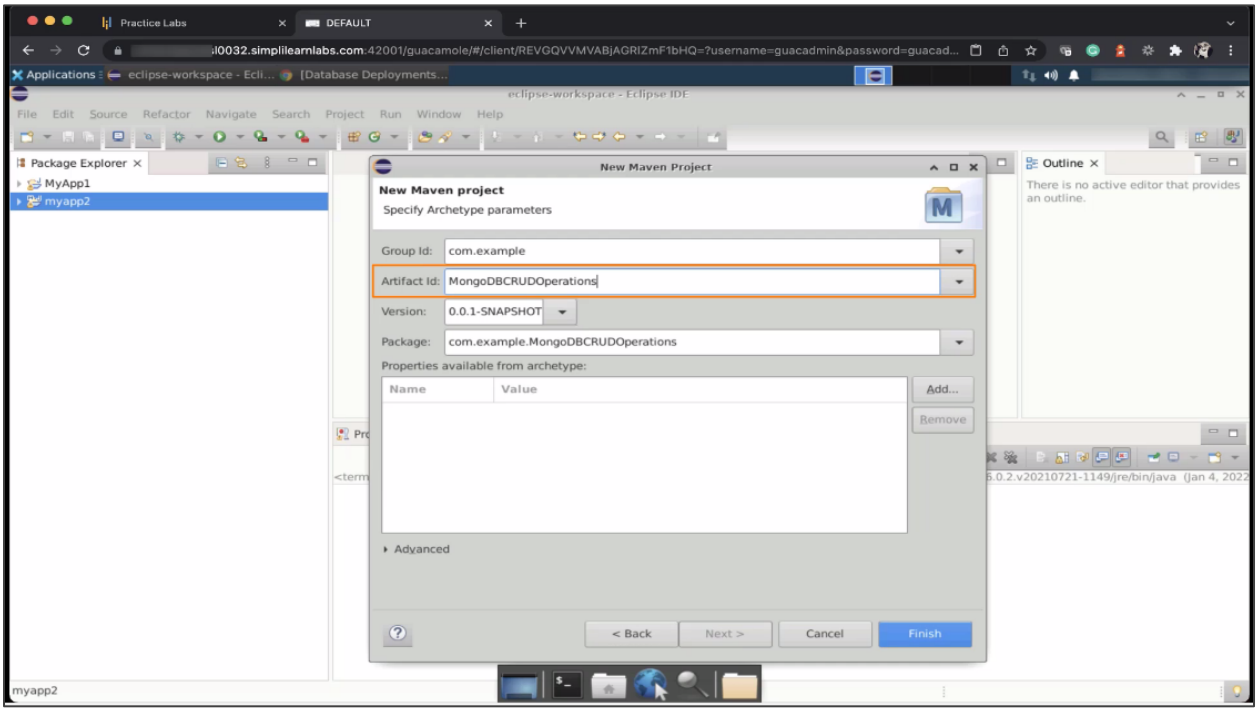
1.4 Choose the workspace location for the project



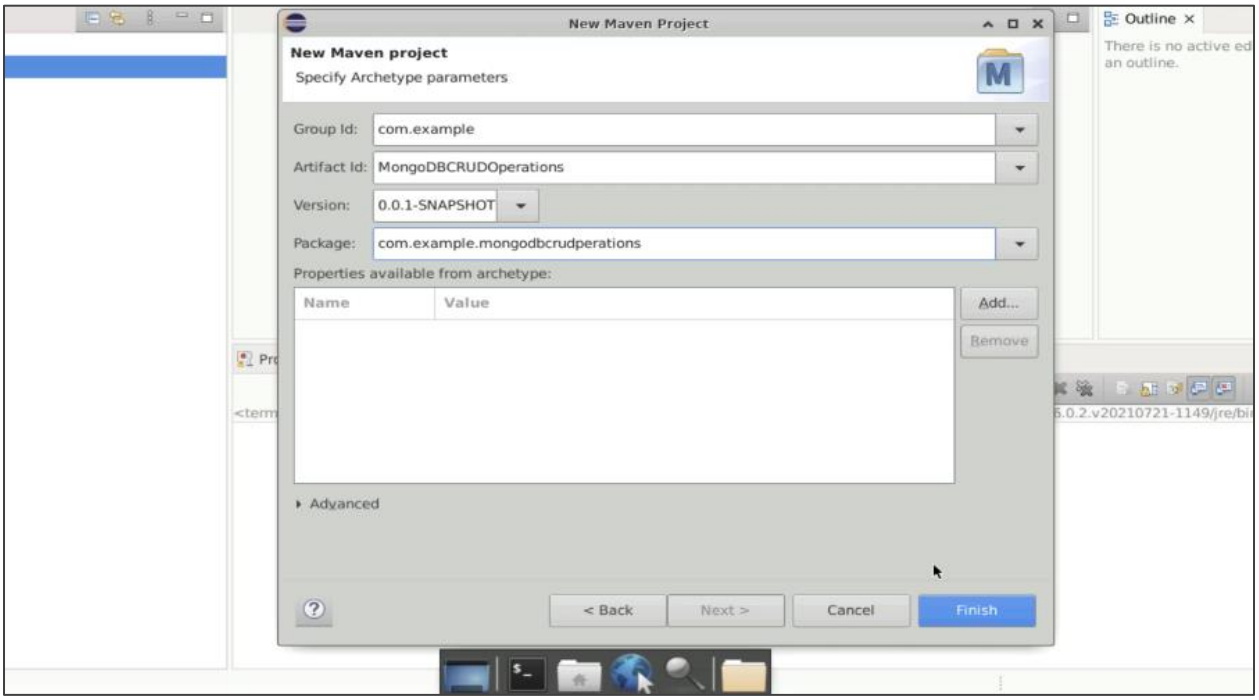
1.5 Select **maven-archetype-quickstart** as the Maven artifact and click **Next**



1.6 Enter **MongoDBCRUDOperations** as the Artifact Id

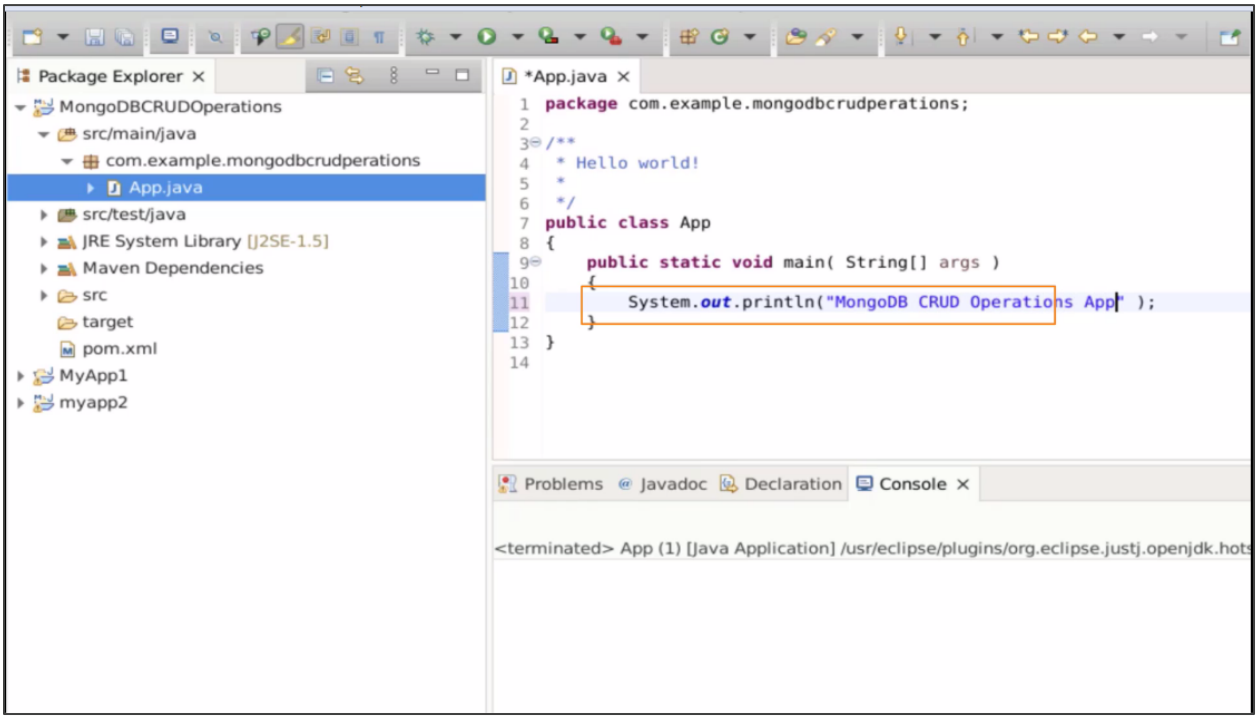


1.7 Enter **com.example** as the domain name and **com.example.mongdbocrudoperations** as the package name, and click on **Finish**

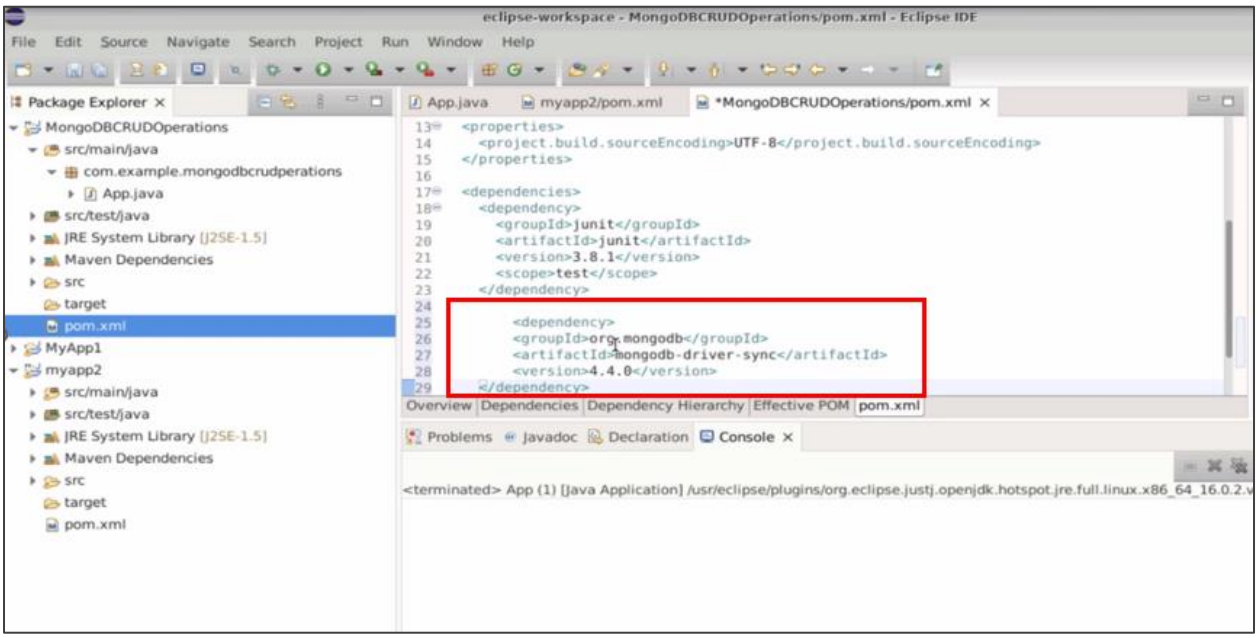


Step 2: Add the dependencies and a print statement

2.1 Open the **App.java** file and add a print statement as **MongoDB CRUD Operations App**

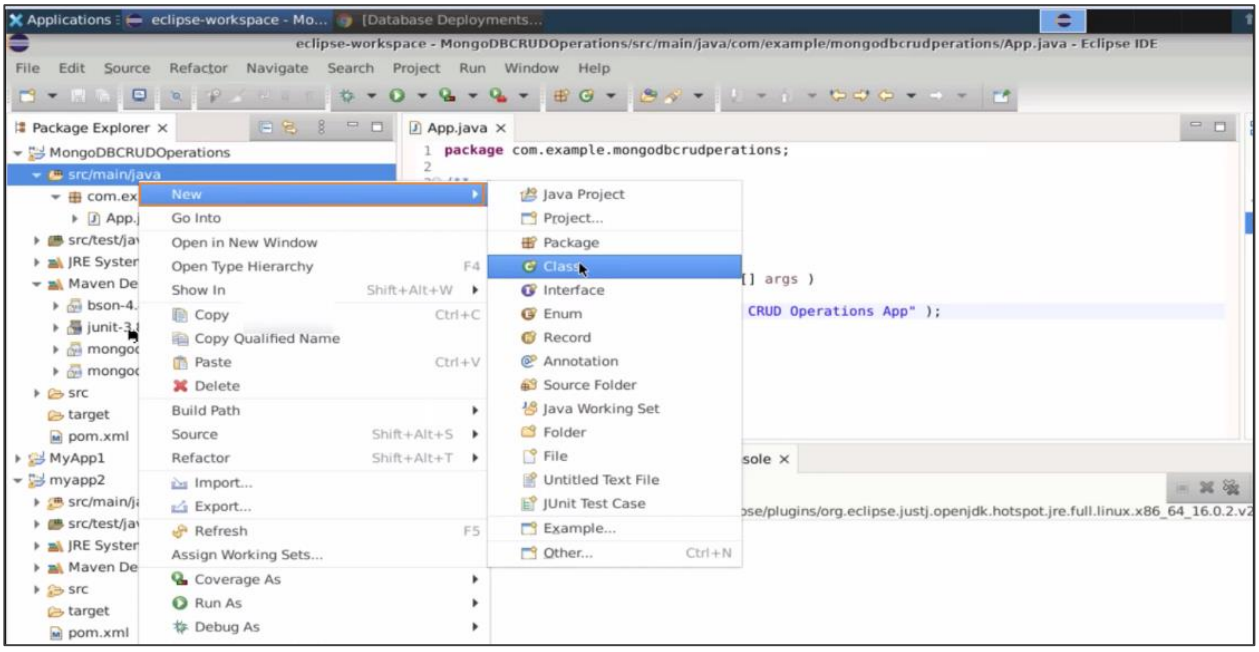


2.2 Next, open the **pom.xml** file and add a **mongodb-driver-sync dependency**

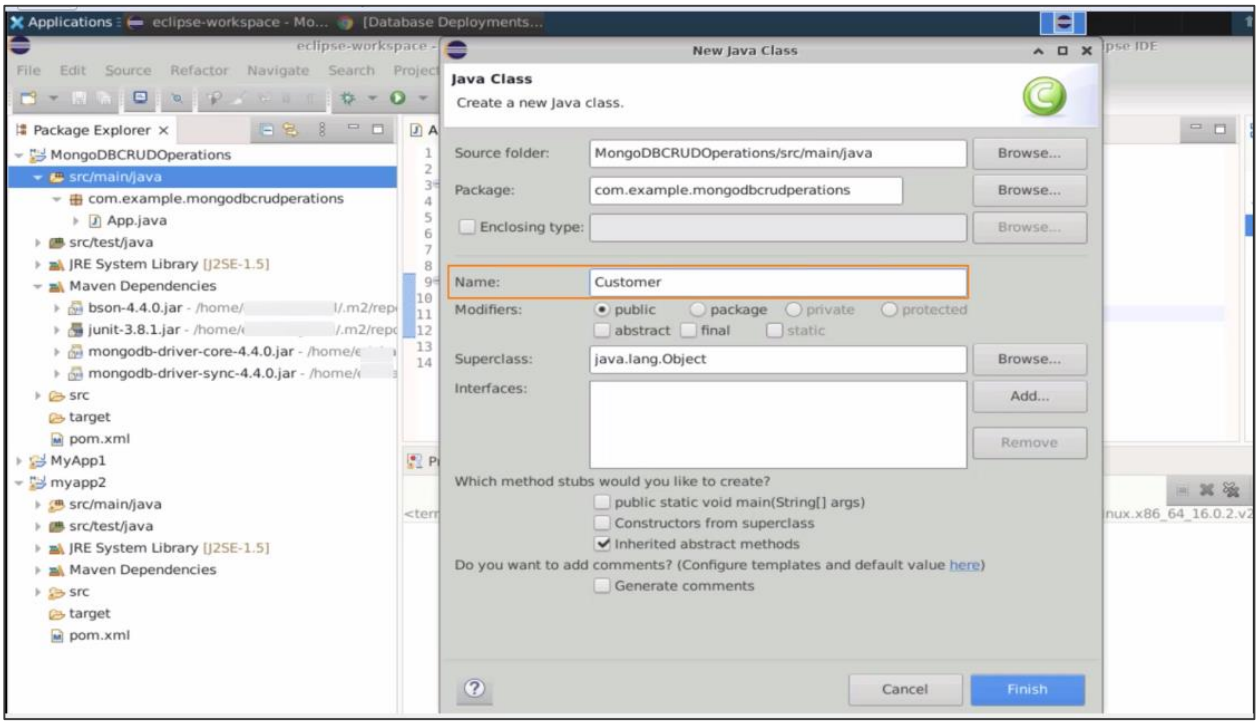


Step 3: Create a Customer and DBOperations class

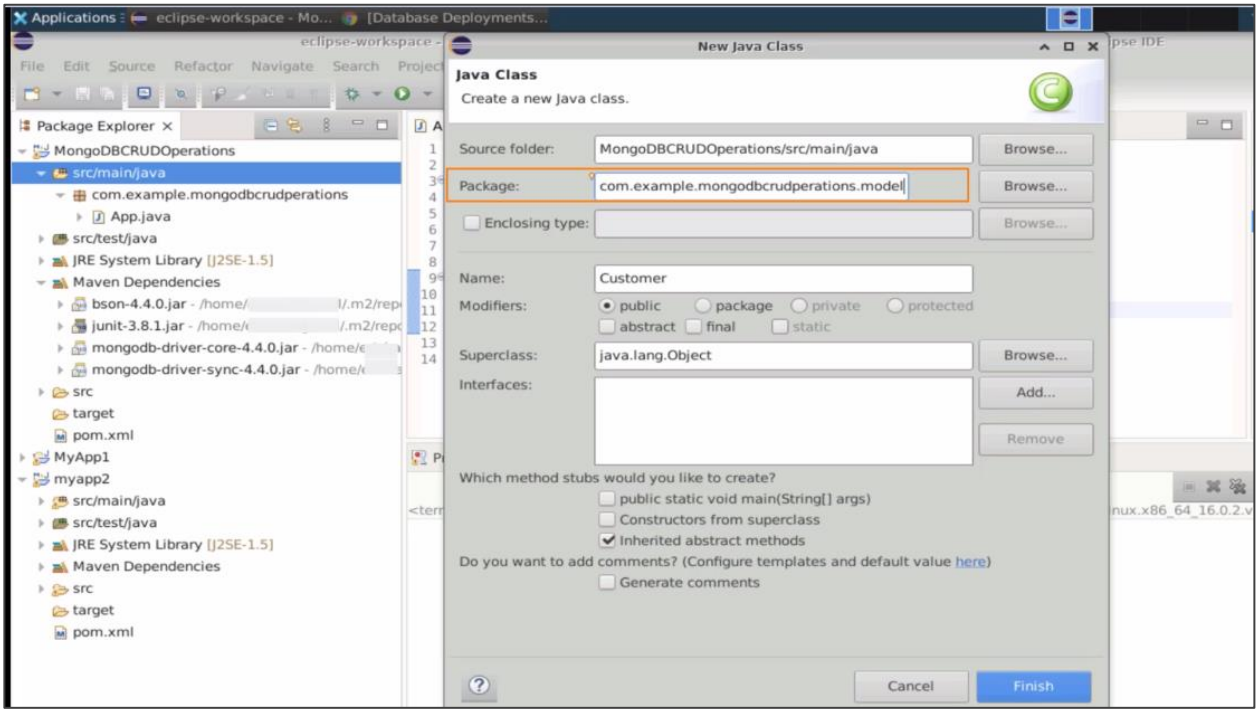
3.1 Create a new class by right-clicking on the project, selecting **New**, and then **Class**



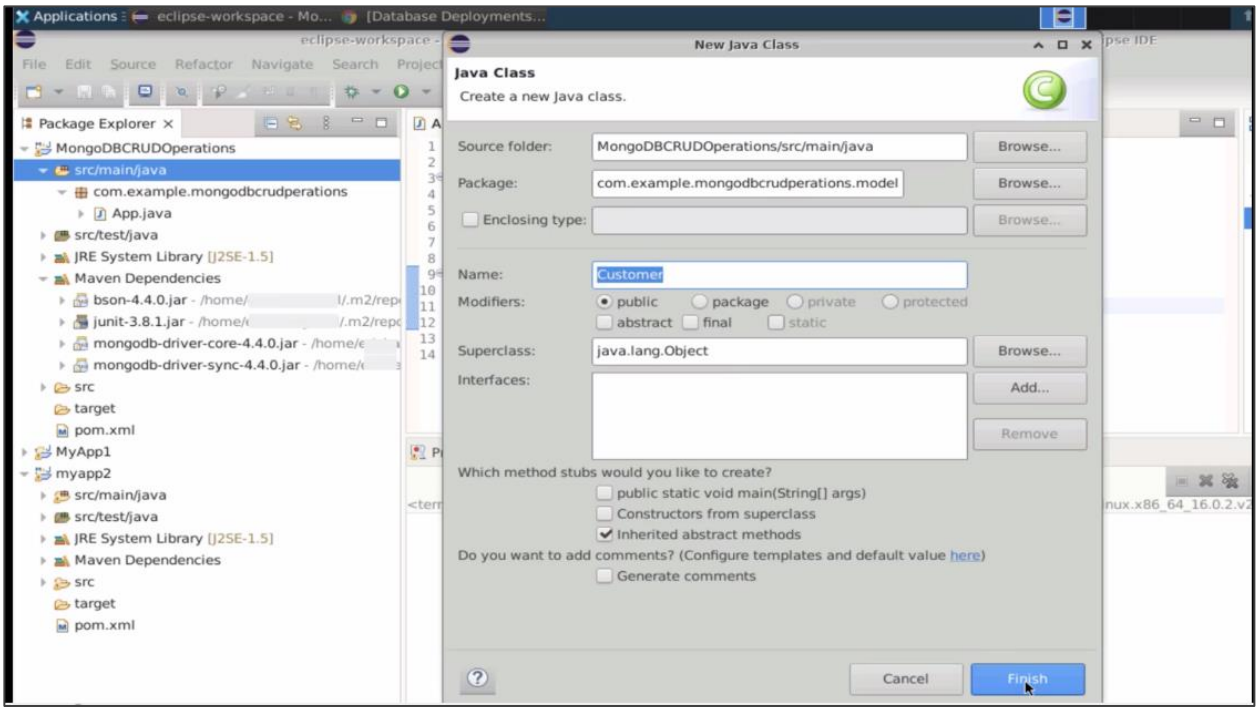
3.2 Name the class **Customer**



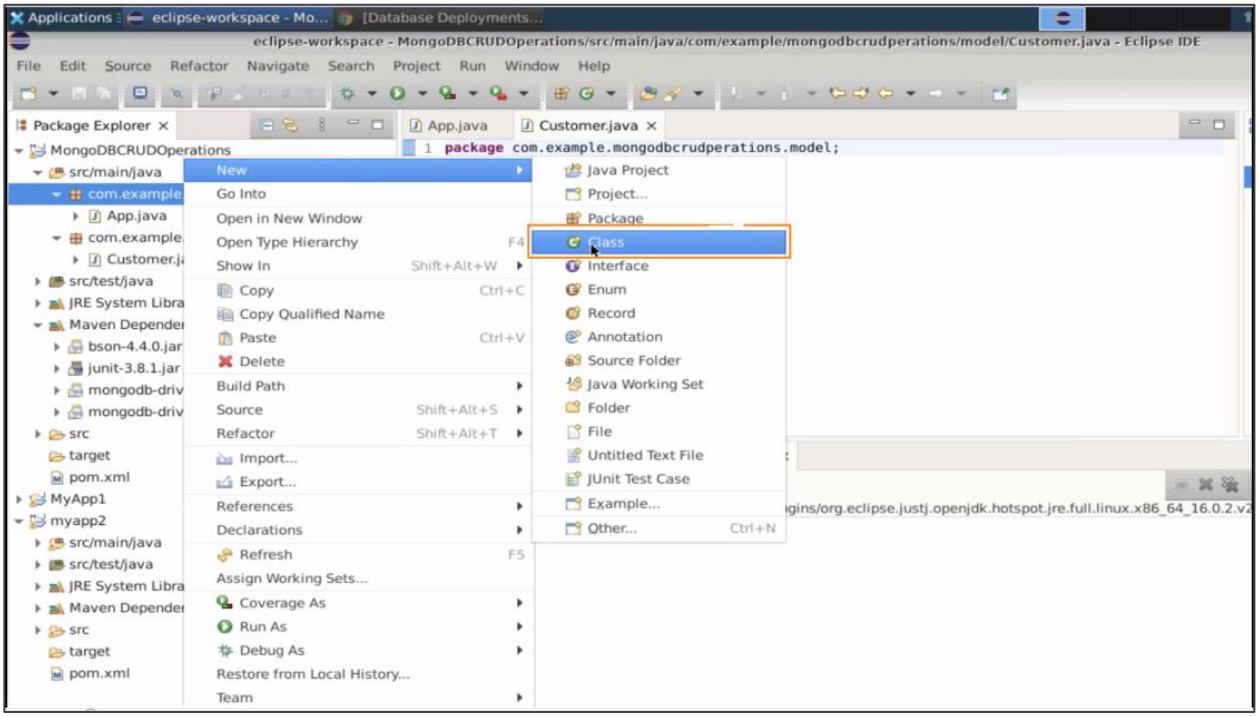
3.3 Then, add **.model** to the package name



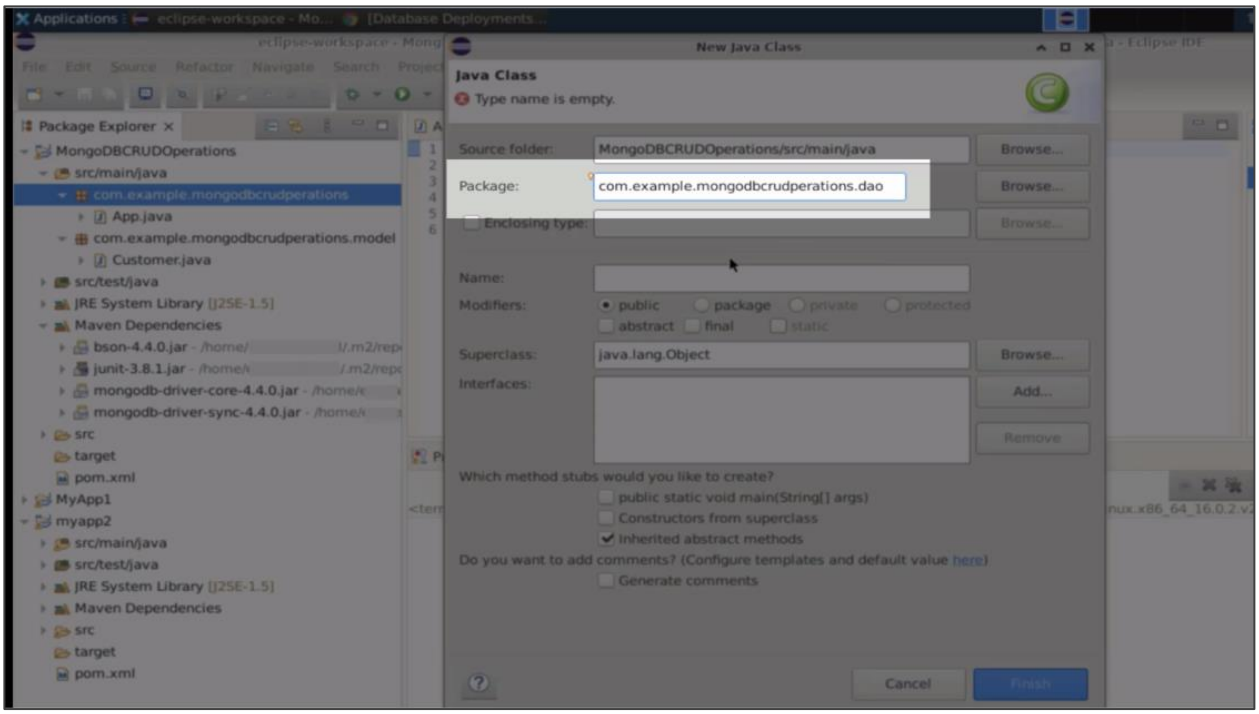
3.4 Click on **Finish**



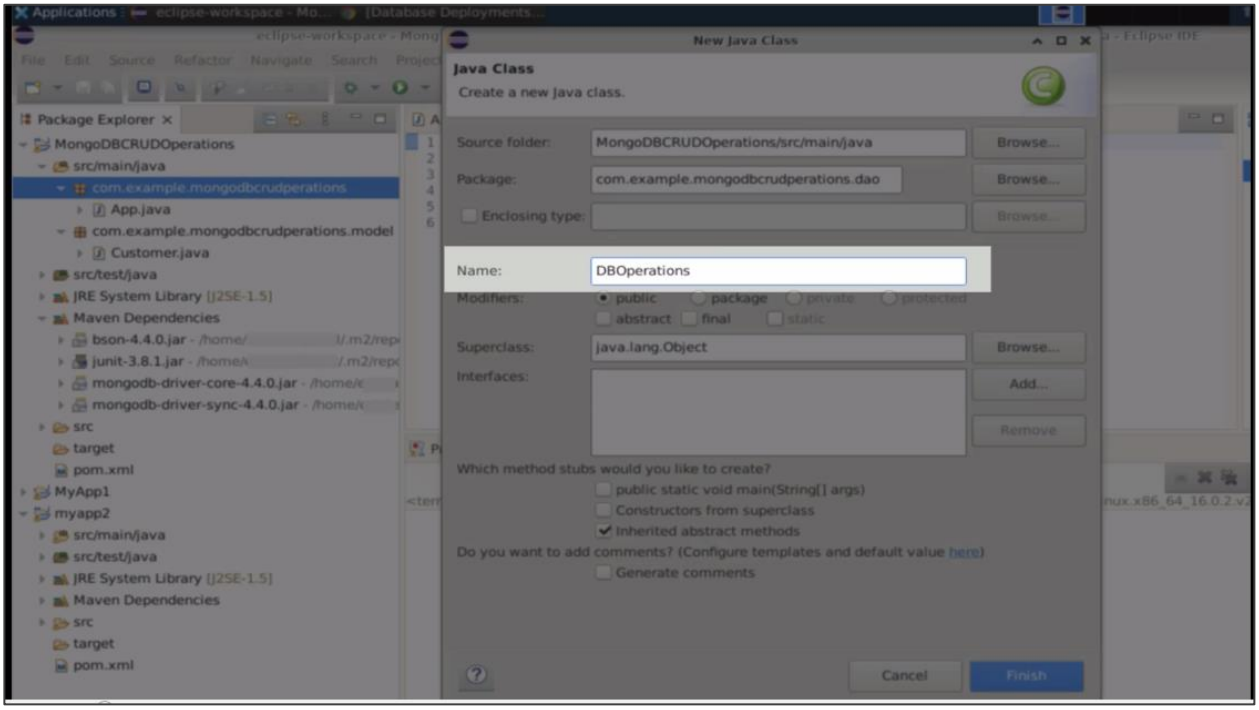
3.5 Create another class under the same package by clicking on **New** and then **Class**



3.6 Name the package **dao**

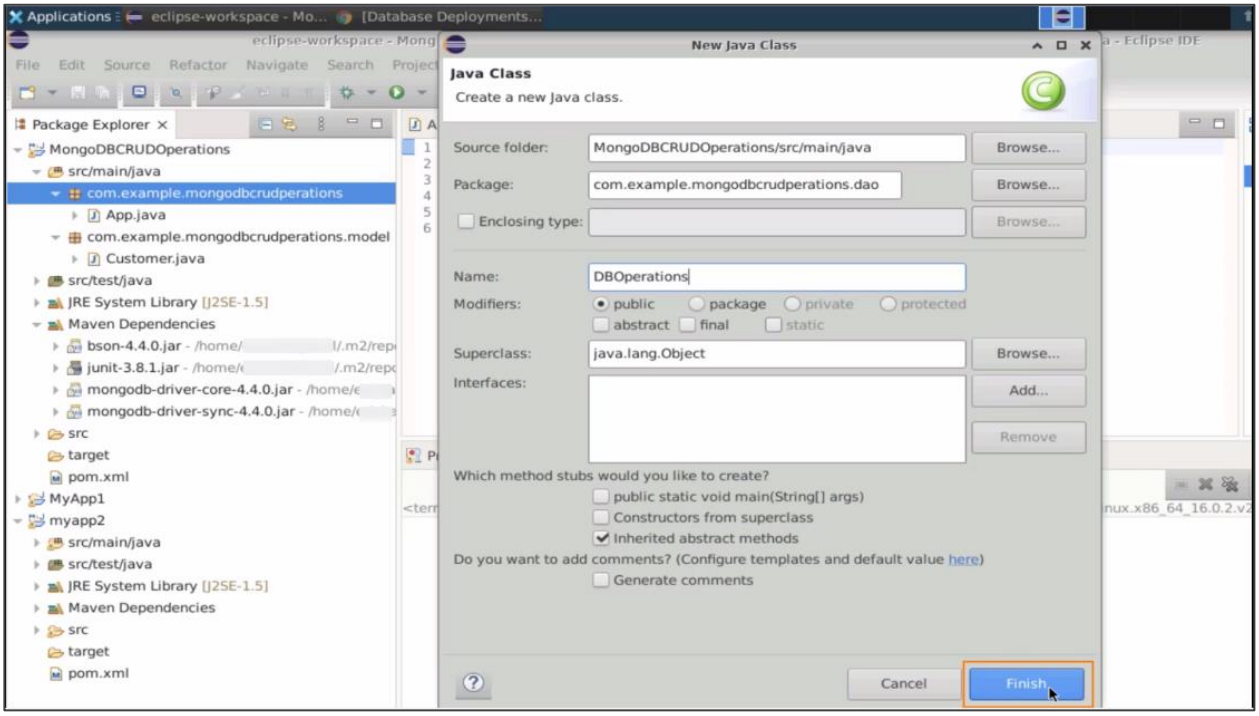


3.7 Name the class **DBOperations**



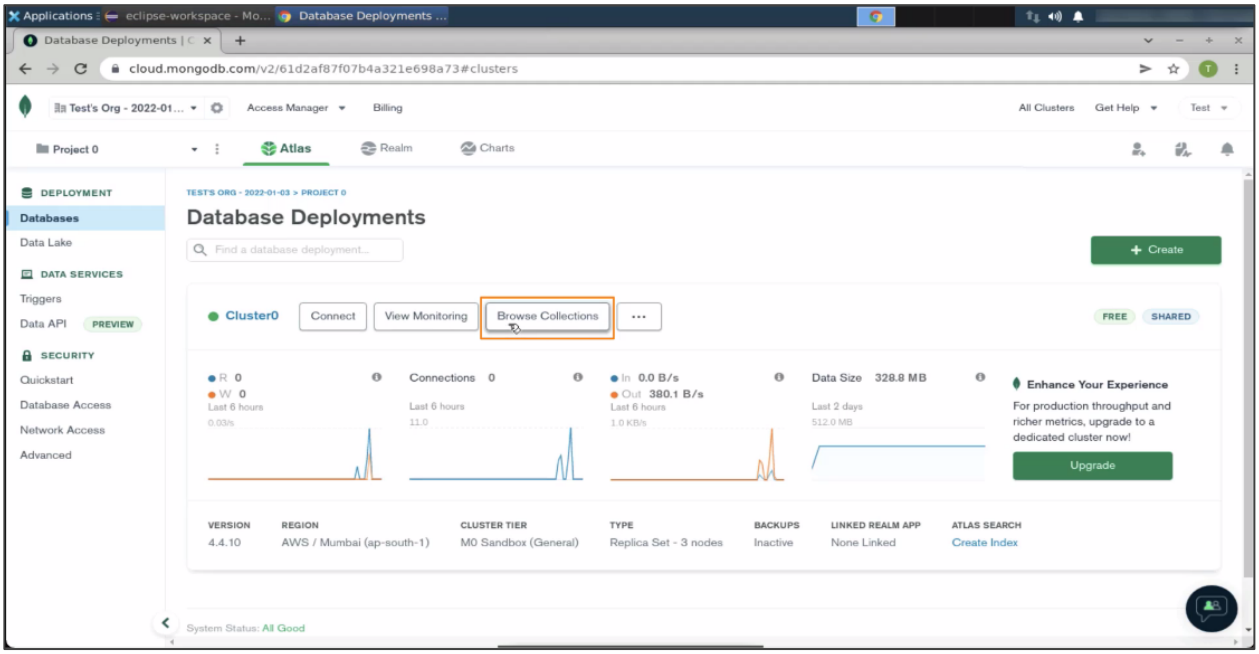
With this, you will be able to write the CRUD operations in MongoDB

3.8 Click on **Finish**

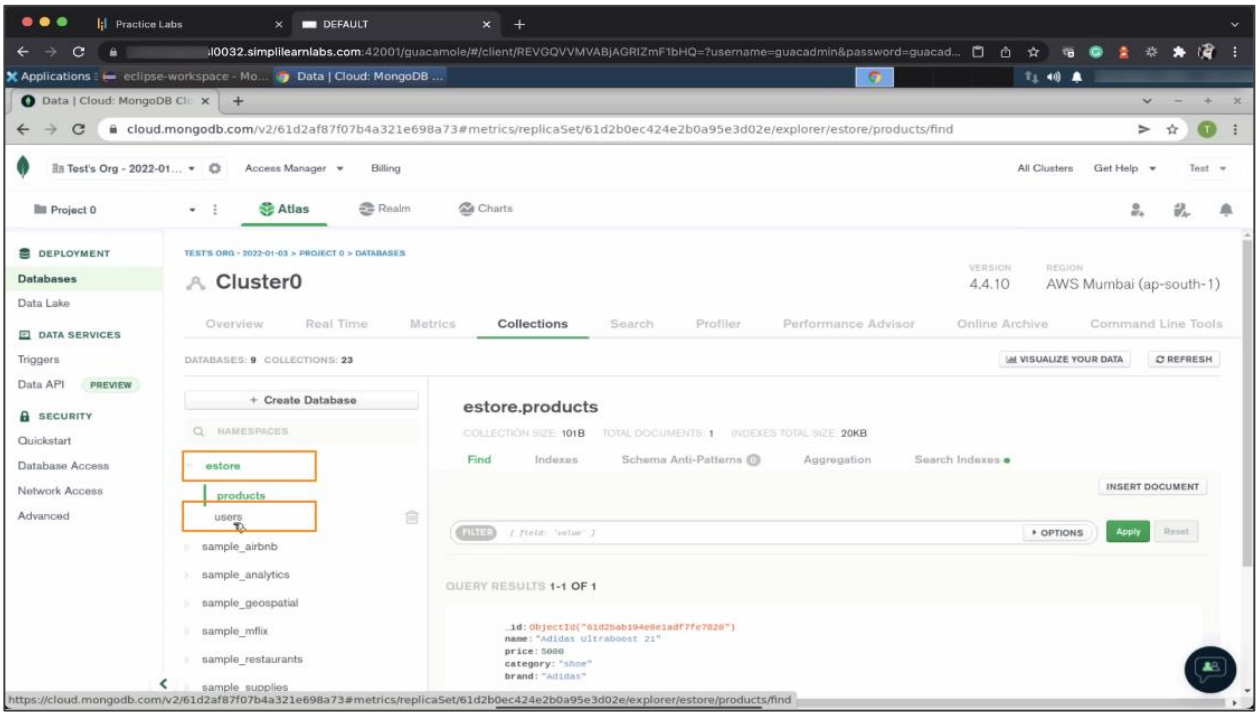


Step 4: Create a collection

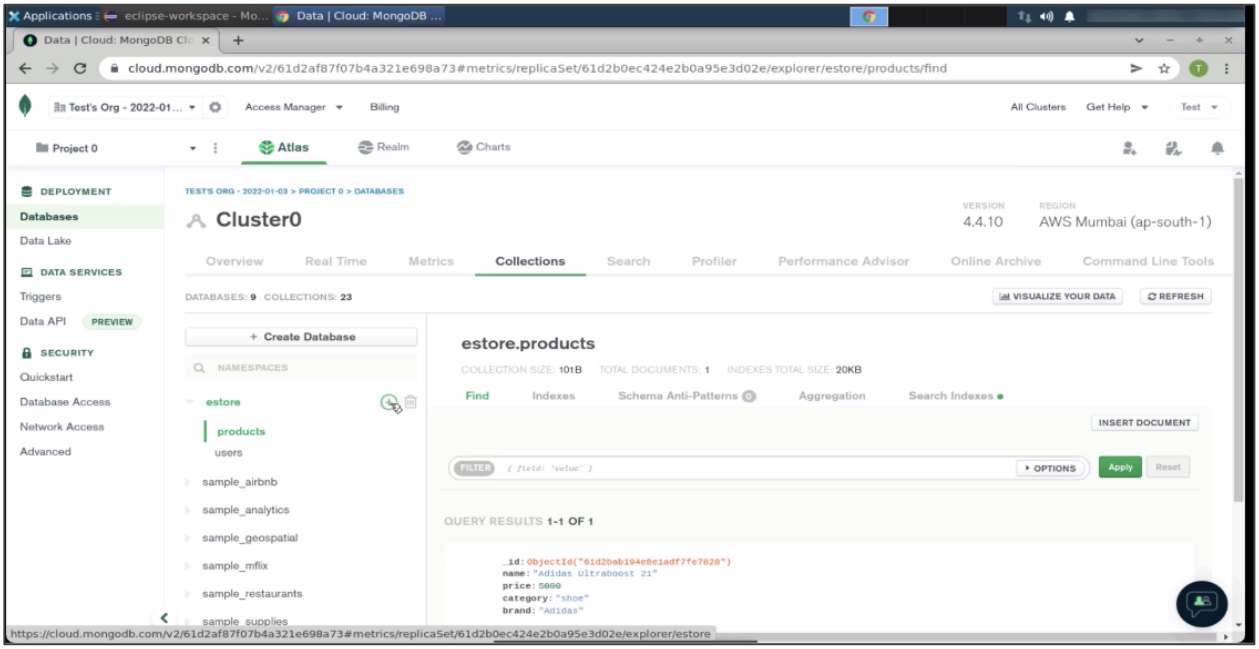
4.1 Inside MongoDB, click on **Browse Collections**



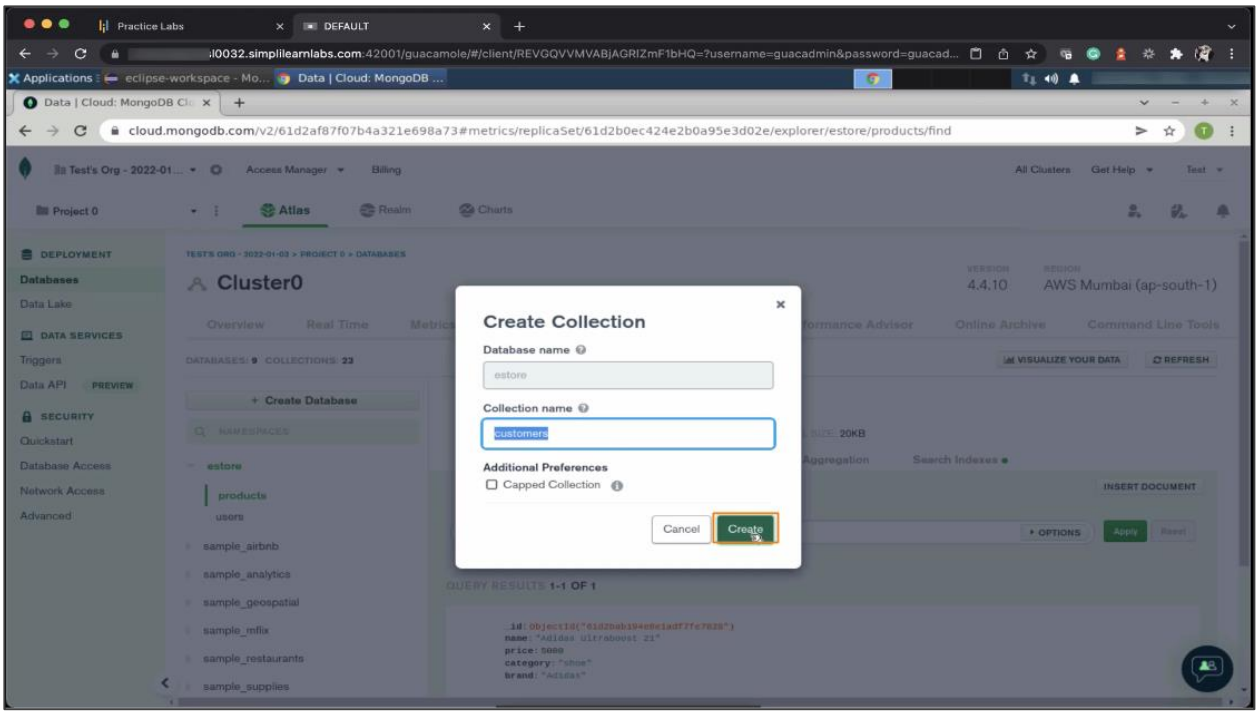
4.2 In **estore,** click on **users**



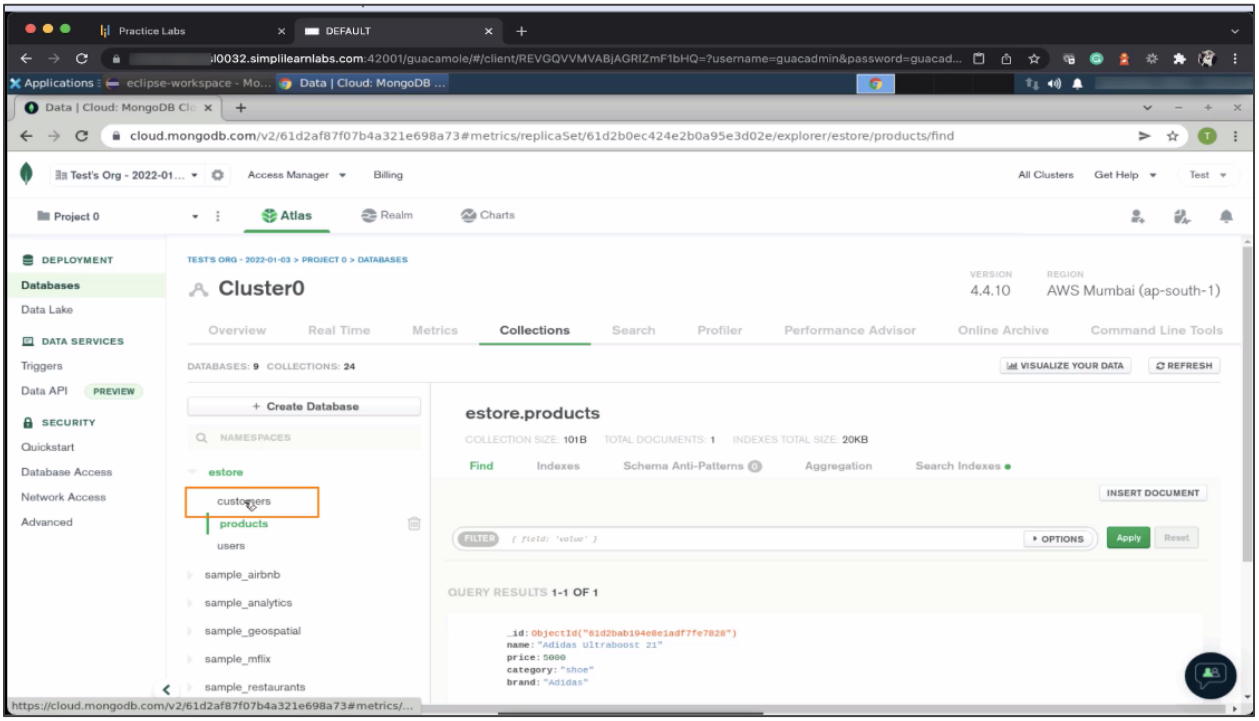
4.3 Create a new **estore** named **Collections**



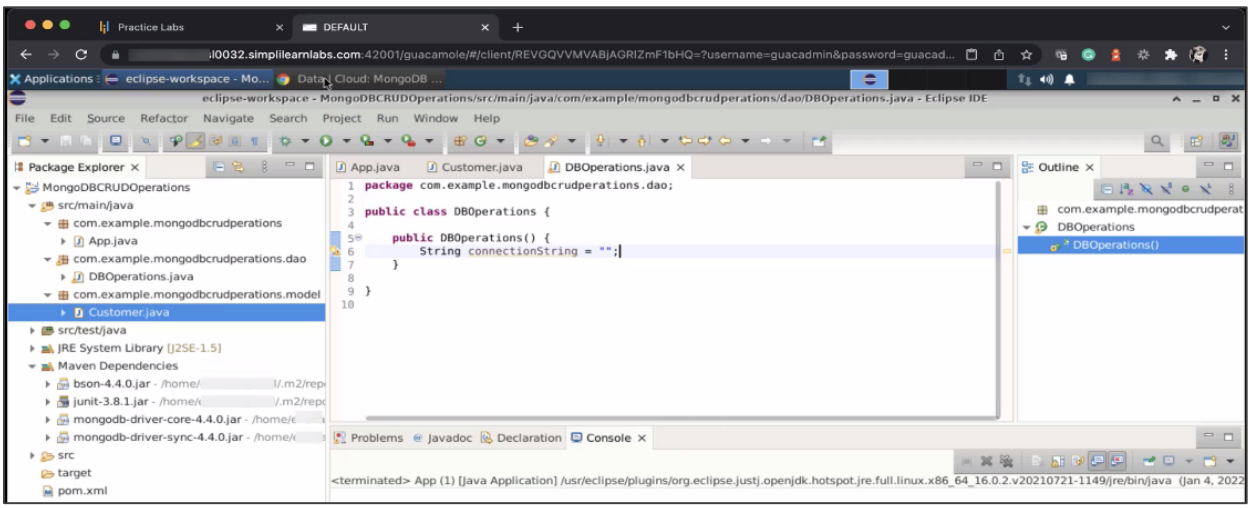
4.4 Name the collection and click on **Create**



4.5 Now, click on the **customers** estore

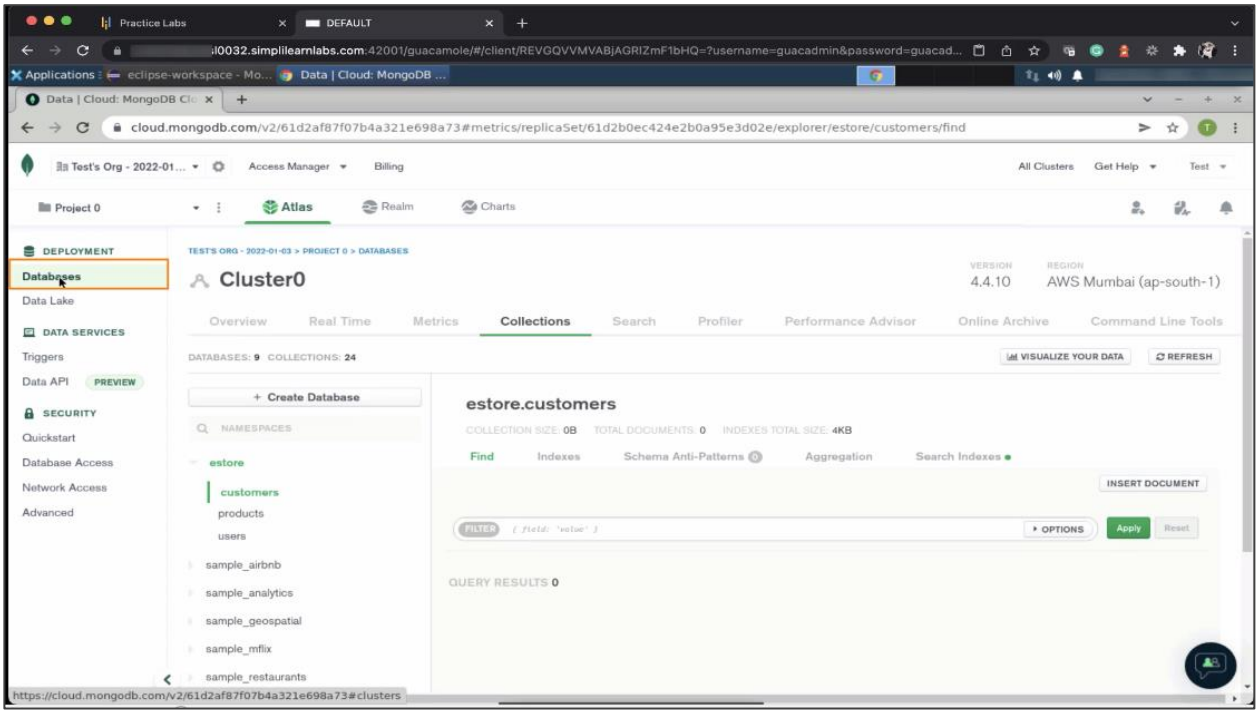


4.6 Go to the Eclipse workspace, and configure the connection with the MongoDB using the connection string

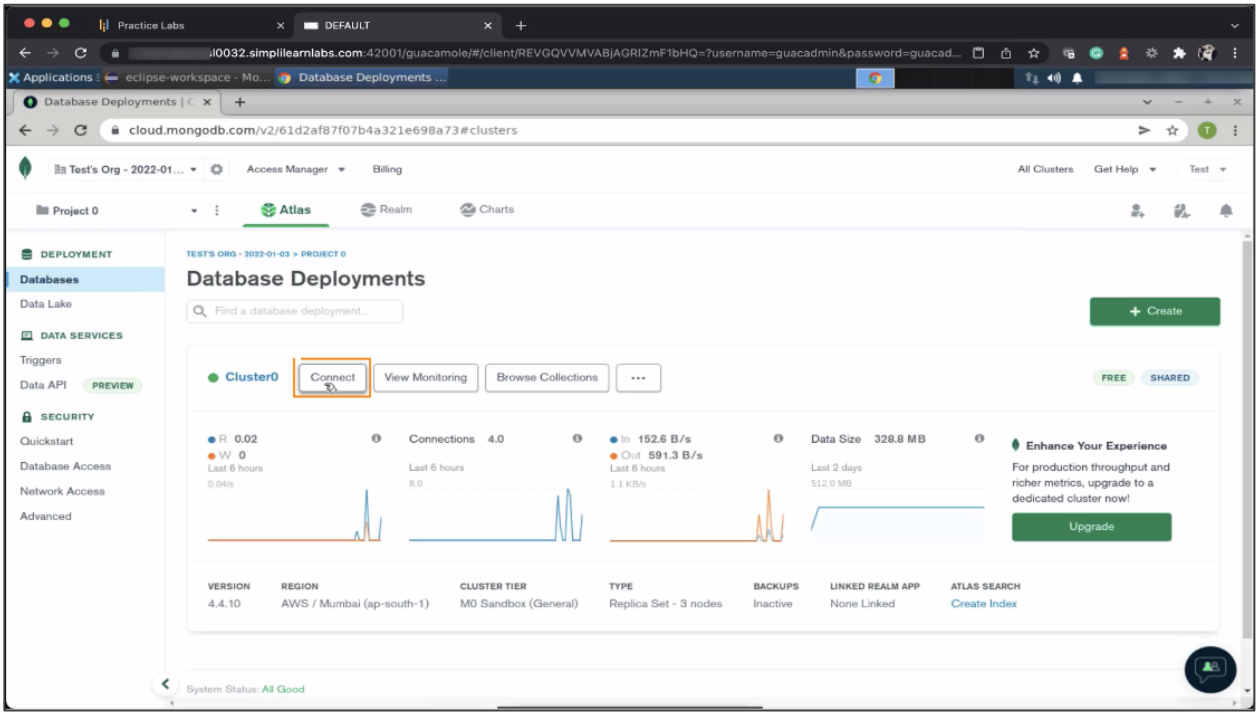


Step 5: Connect database to the application

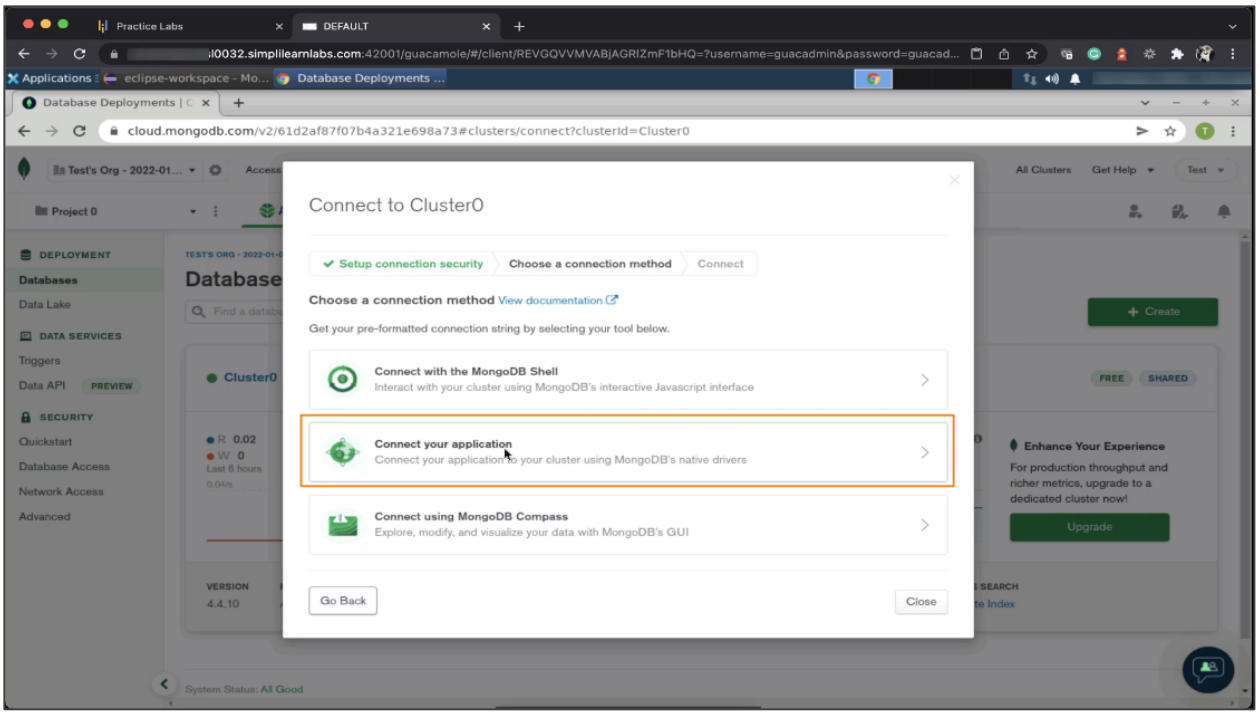
5.1 Navigate to **Databases**



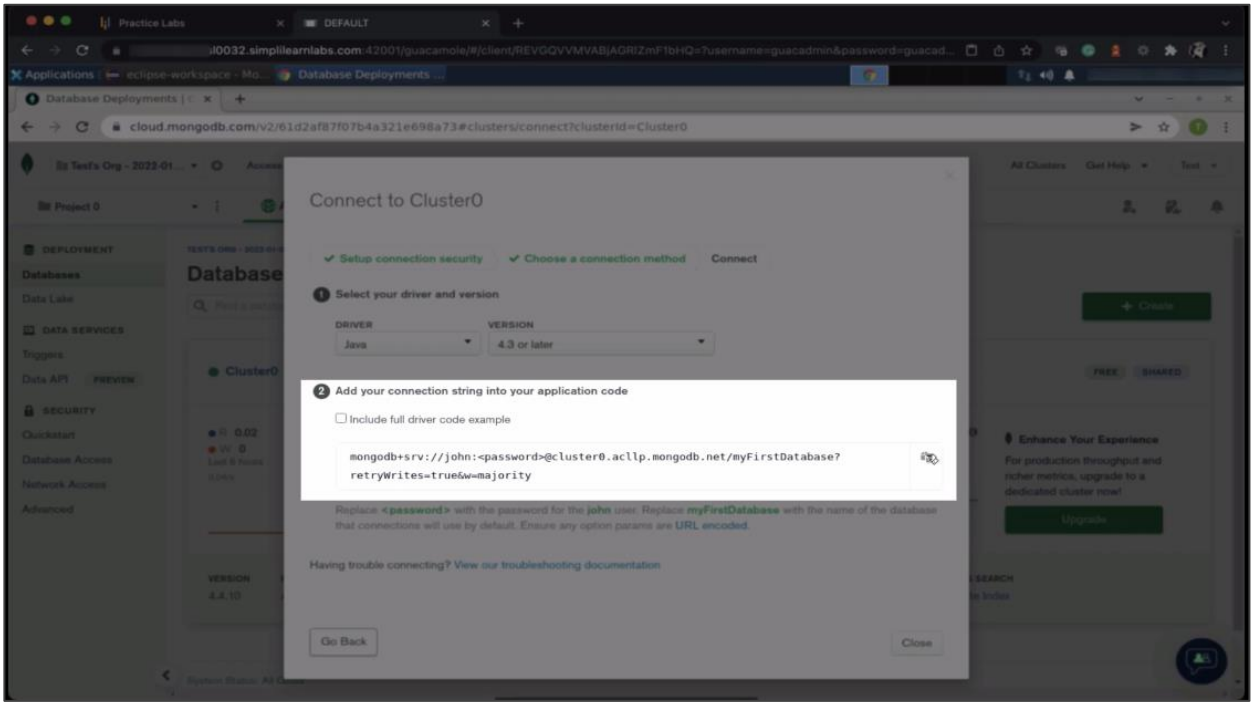
5.2 Then, click on **Connect**



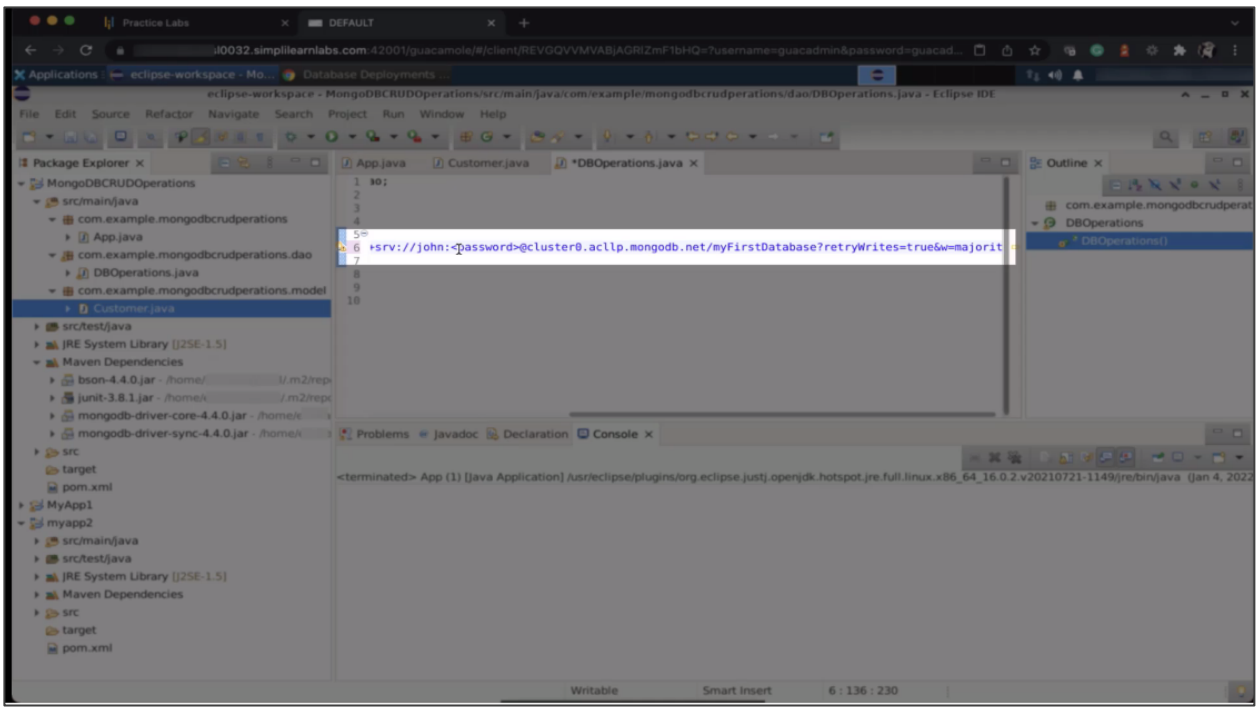
5.3 Select **Connect your application**



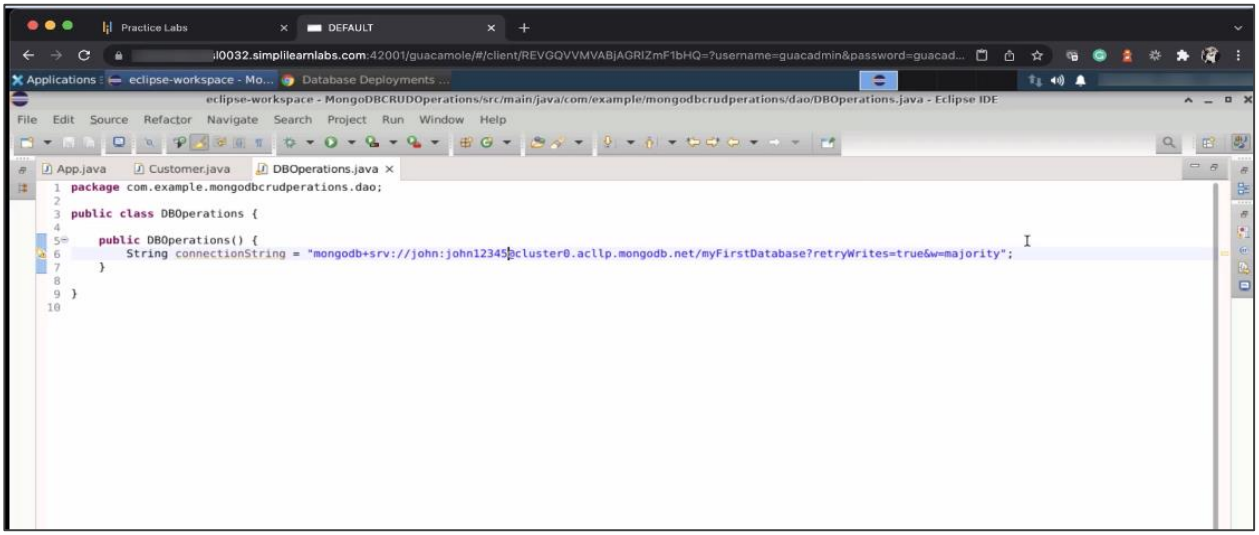
5.4 Copy the connection string



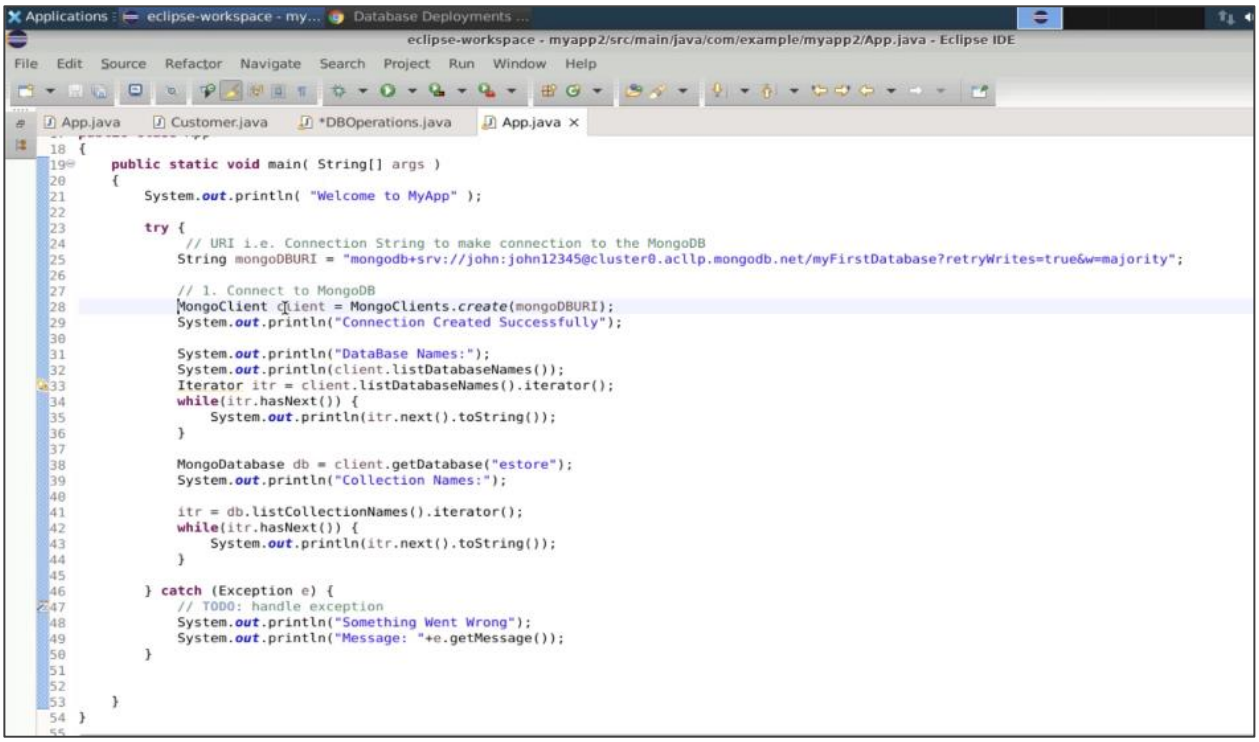
5.5 Paste the connection string in the code in the **Customer.java** file



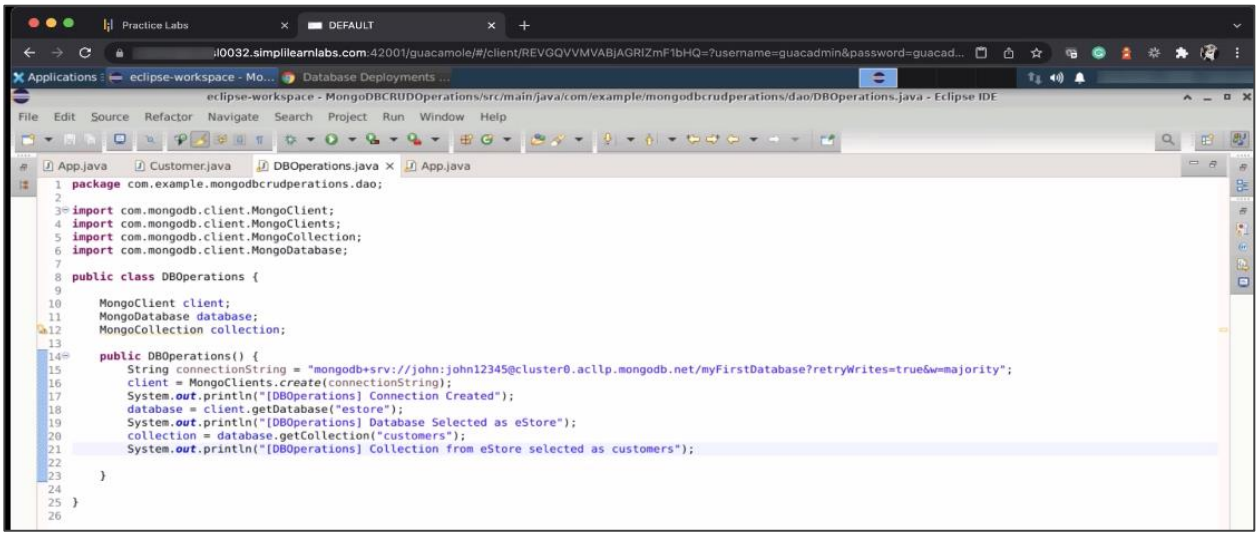
5.6 Change the password in the connection string as required, for example **john12345**



5.7 Go to the **App.java** file and you can see the MongoClient and MongoDatabase client. Use these in the **customer.java** file.

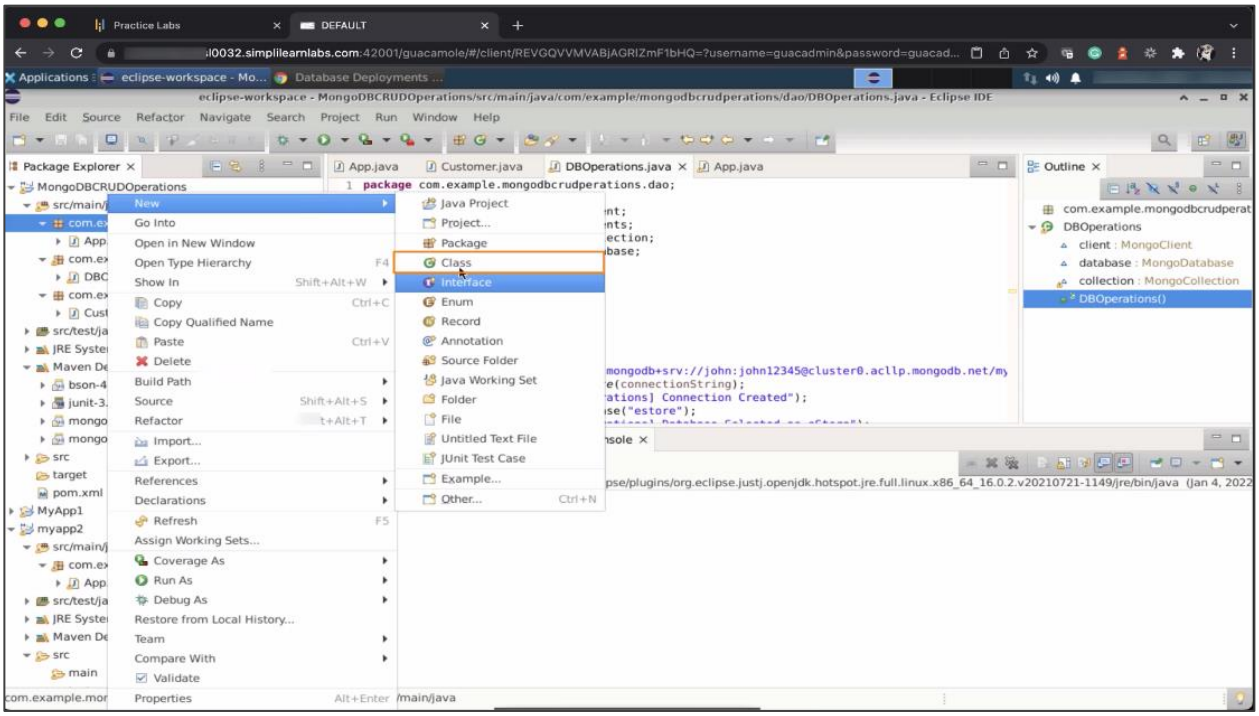


5.8 Go to the **DBOperations.java** file, define **MongoClient**, **MongoDatabase**, and **MongoCollection** APIs, and update the code as follows:

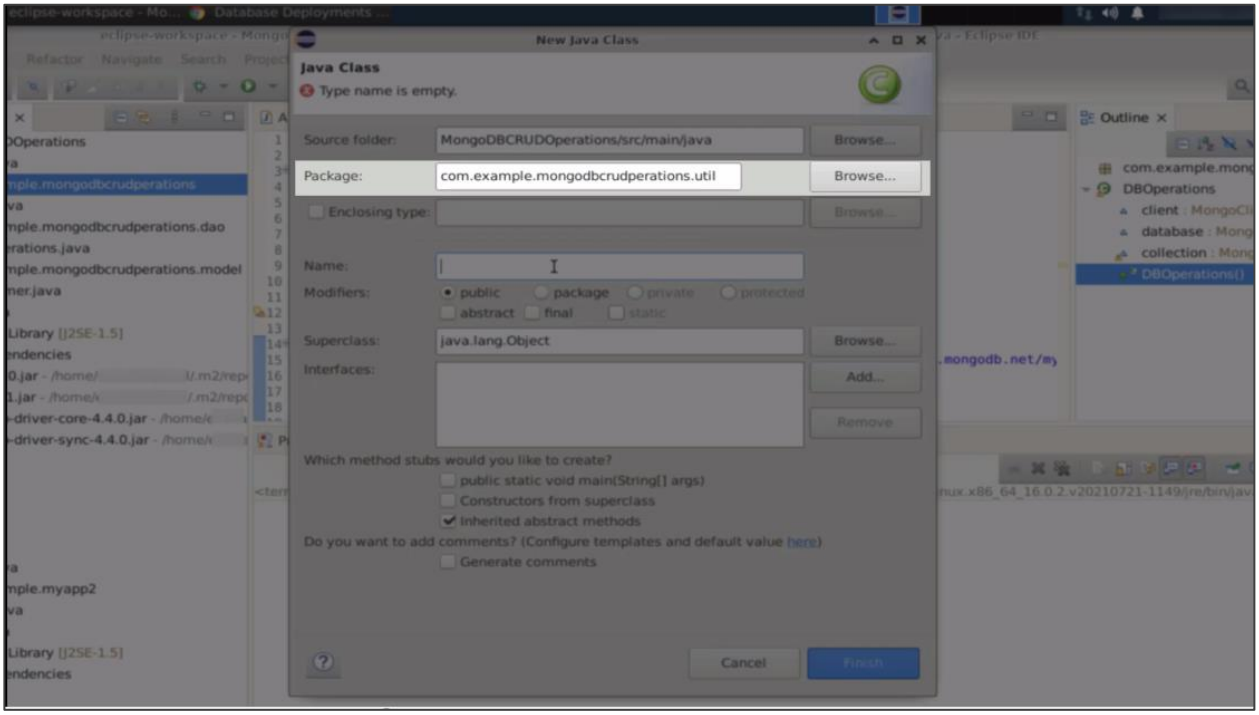


Step 6: Create a new class

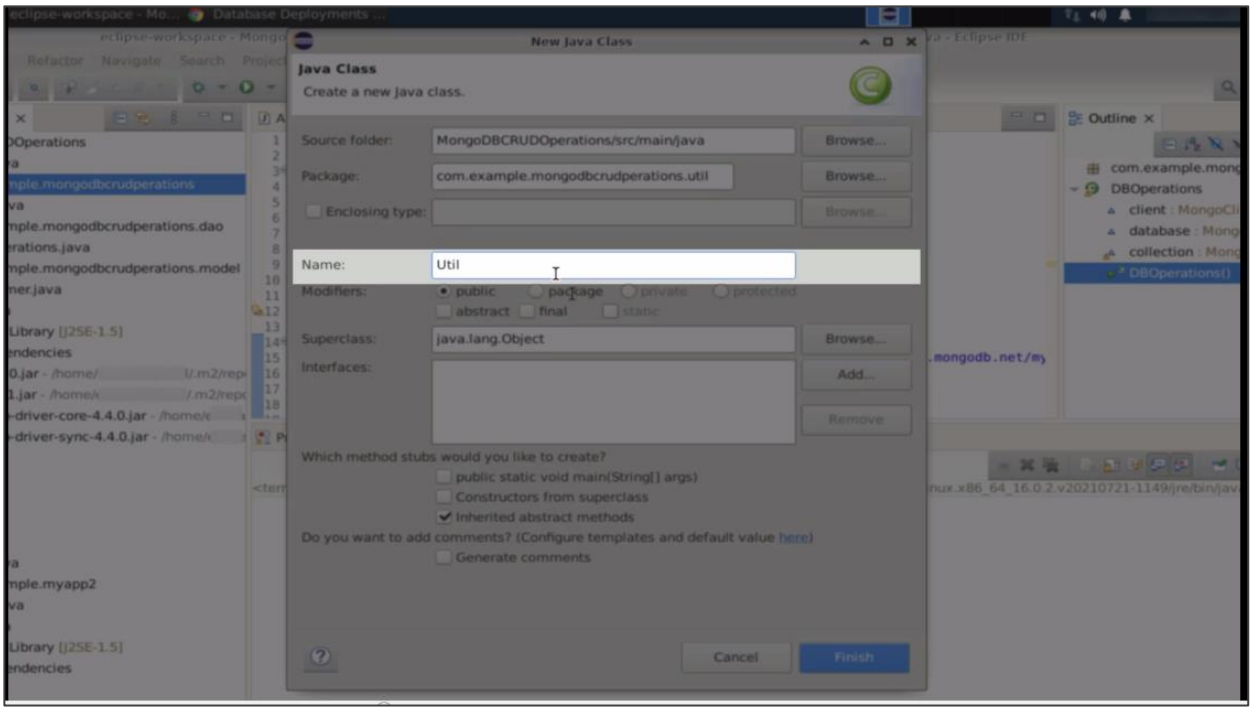
6.1 Create a new client by right-clicking on the file, selecting **New**, and then selecting **Class**



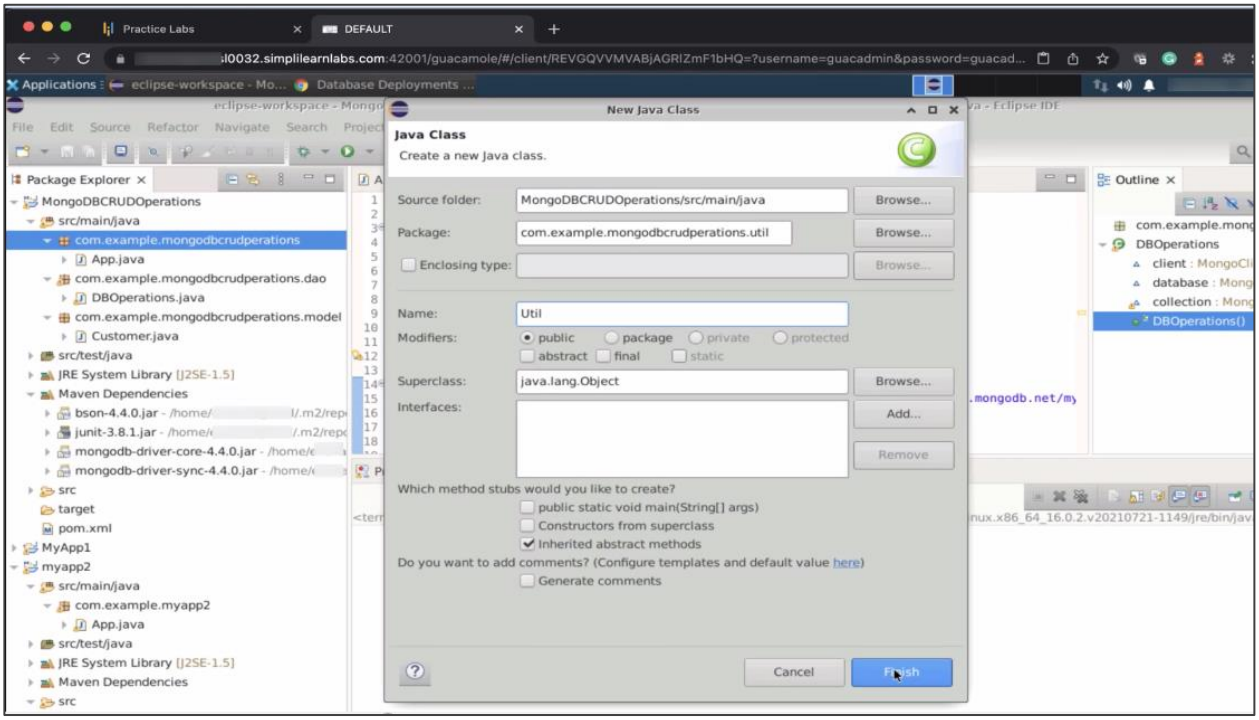
6.2 Add **.util** to the package



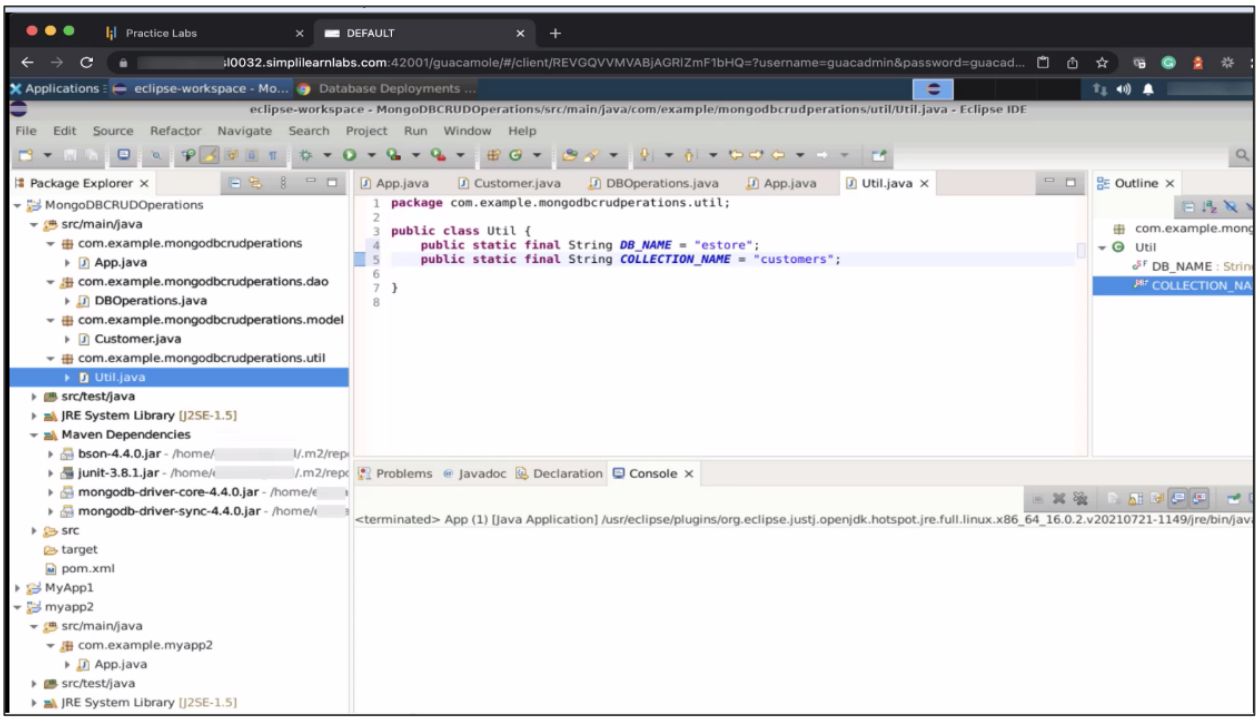
6.3 Name the class **Util**



6.4 Click on **Finish**



6.5 In the **Util.java** class, define string for **DB_NAME** and **COLLECTION_NAME**



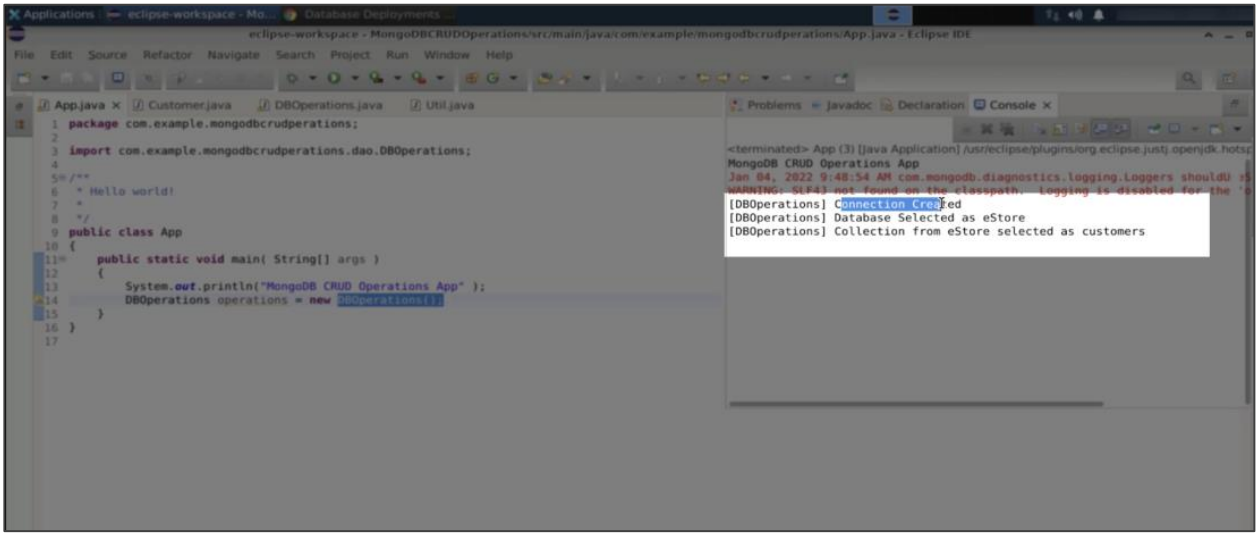
6.6 In **DBOperations.java**, replace the **estore** with the **Util** class and the collection with **Util.COLLECTION_NAME**

```
package com.example.mongodbcrudperations.dao;

import com.example.mongodbcrudperations.util.Util;
import com.mongodb.client.MongoClient;
import com.mongodb.client.MongoClients;
import com.mongodb.client.MongoCollection;
import com.mongodb.client.MongoDatabase;

public class DBOperations {

    MongoClient client;
    MongoDatabase database;
    MongoCollection collection;

    public DBOperations() {
        String connectionString = "mongodb+srv://john:john12345@cluster0.acllp.mongodb.net/myFirstDatabase?retryWrites=true&w=majority";
        client = MongoClients.create(connectionString);
        System.out.println("[DBOperations] Connection Created");
        database = client.getDatabase(Util.DB_NAME);
        System.out.println("[DBOperations] Database Selected as eStore");
        collection = database.getCollection(Util.COLLECTION_NAME);
        System.out.println("[DBOperations] Collection from eStore selected as customers");

    }

}
```

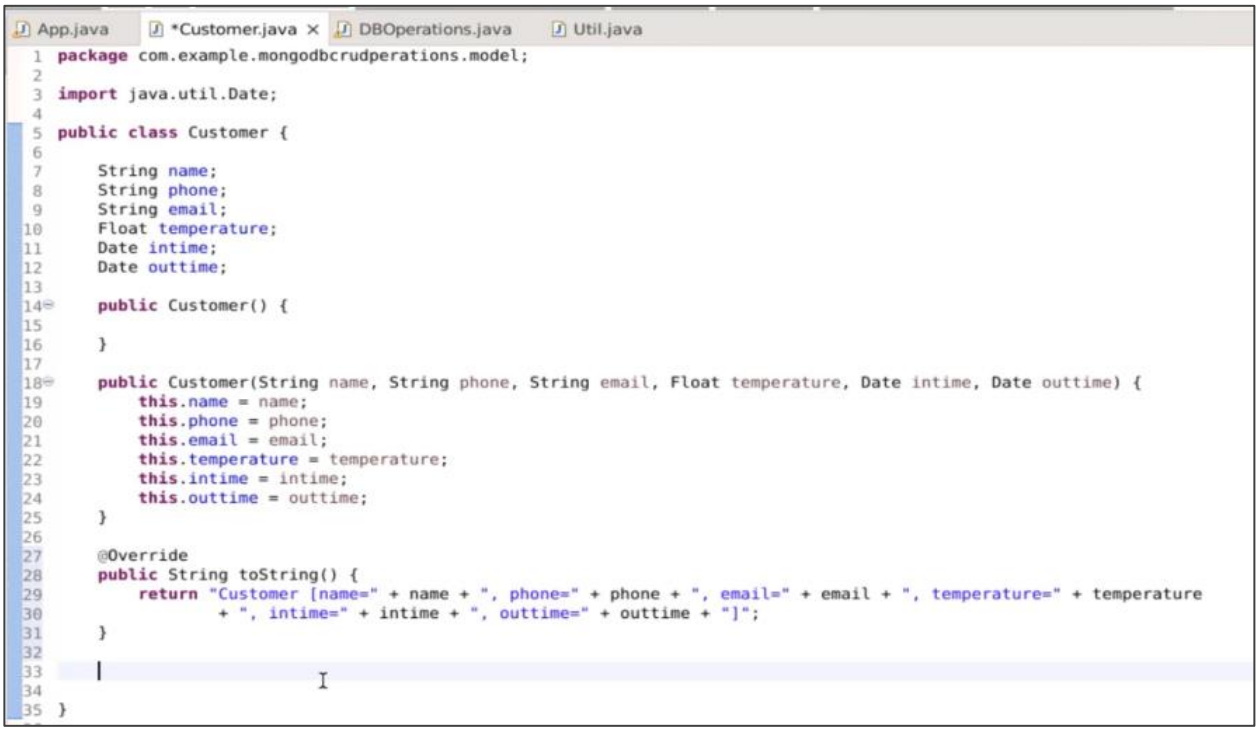
6.7 To test the operations app, navigate to the **App.java** file and write the object of **DBOperations** class. This will automatically create a constructor in the DBOperations.

```
package com.example.mongodbcrudperations;

import com.example.mongodbcrudperations.dao.DBOperations;

/**
 * Hello world!
 *
 */
public class App
{
    public static void main( String[] args )
    {
        System.out.println("MongoDB CRUD Operations App" );
        DBOperations operations = new DBOperations();
    }
}
```

6.8 Run the app and you can see that the connection is created

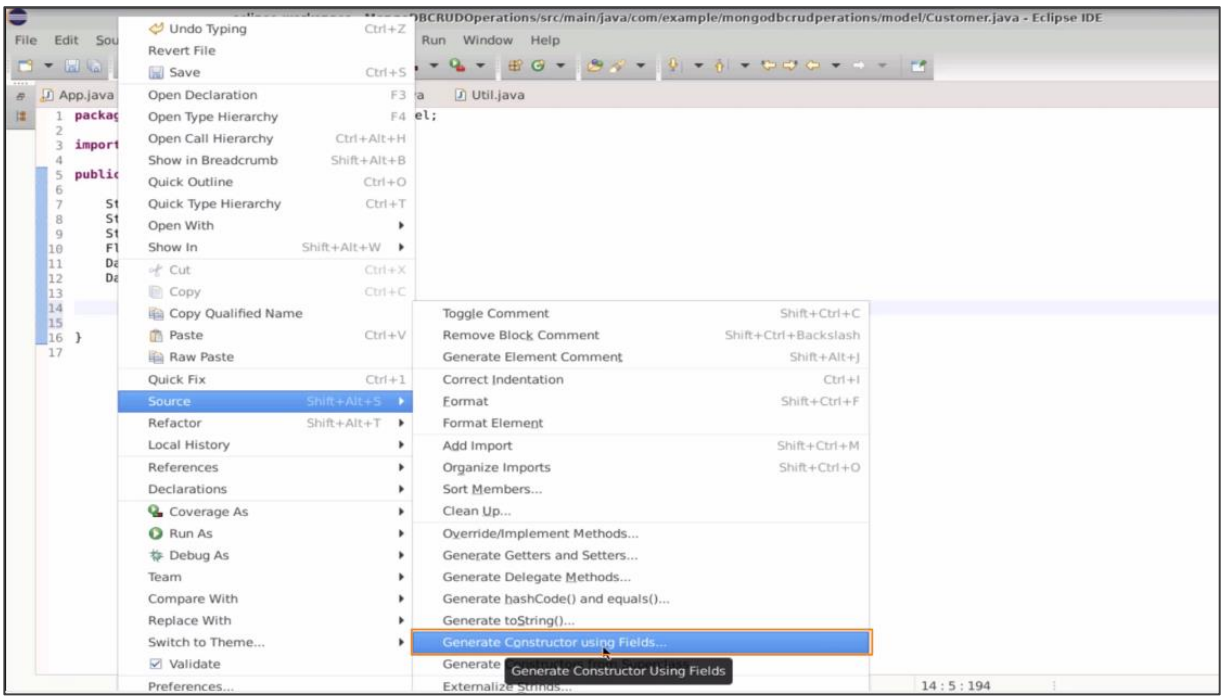


6.9 Go to the **Customer.java** file and define the customer attributes inside the **Customer** class. Generate the customer string.

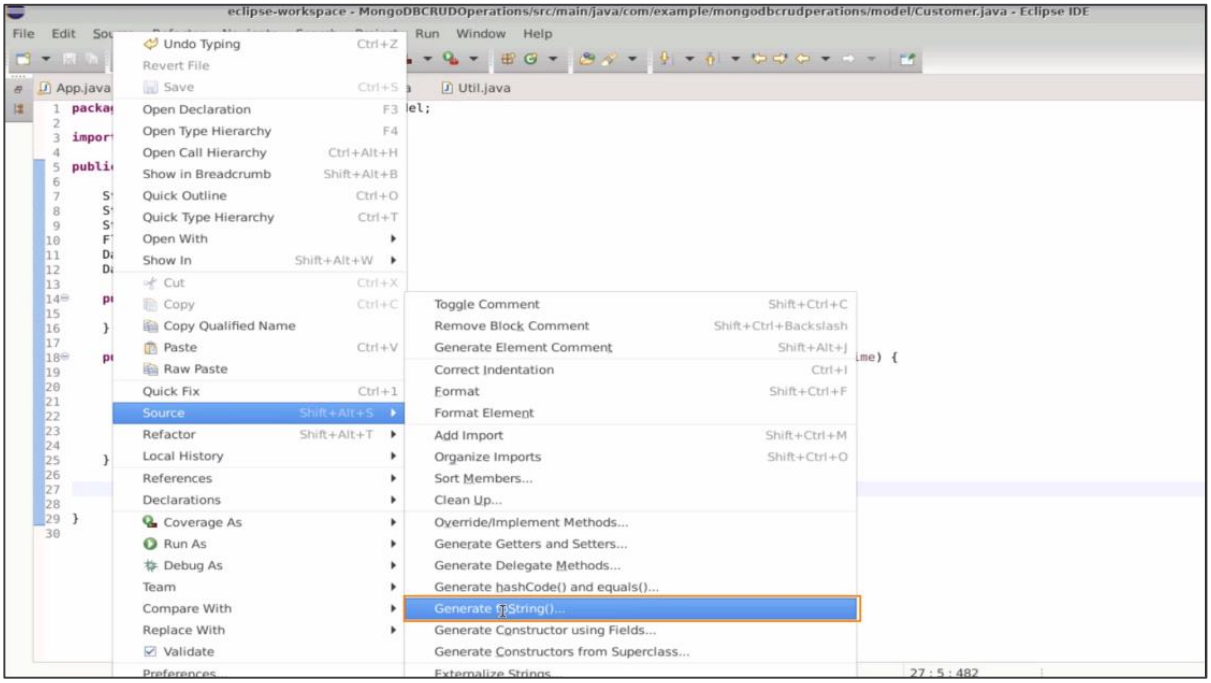


Step 7: Generate the constructor

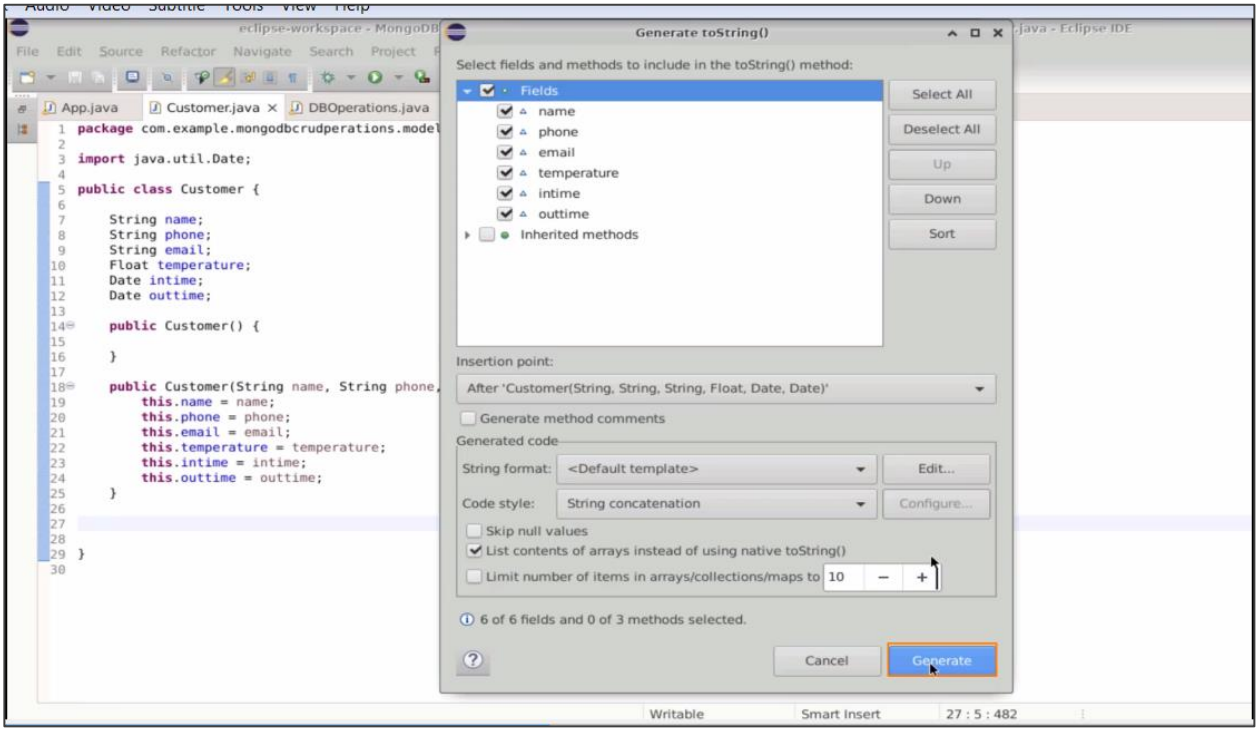
7.1 Right-click and select **Source**, then select **Generate Constructor using Fields**



7.2 Go to **Source** and click on **Generate toString()**



7.3 Click on **Generate**



7.4 Go back to the **App.java** file, define the **customer** object, and run the code

7.5 Go back to the **DBOperations.java** file and create a function insertCustomer to insert the customer details and handle exceptions, if any. Use the **insertOne()** function to insert the customer data.

```java
import com.mongodb.client.MongoClient;
import com.mongodb.client.MongoClients;
import com.mongodb.client.MongoCollection;
import com.mongodb.client.MongoDatabase;

public class DBOperations {

    MongoClient client;
    MongoDatabase database;
    MongoCollection collection;

    public DBOperations() {

        try {
            String connectionString = "mongodb+srv://john:john12345@cluster0.acllp.mongodb.net/myFirstDatabase?retryWrites=true&w=majority";
            client = MongoClients.create(connectionString);
            System.out.println("[DBOperations] Connection Created");
            database = client.getDatabase(Util.DB_NAME);
            System.out.println("[DBOperations] Database Selected as eStore");
            collection = database.getCollection(Util.COLLECTION_NAME);
            System.out.println("[DBOperations] Collection from eStore selected as customers");
        } catch (Exception e) {
            System.out.println("Something went Wrong: "+e);
        }


    }

    public void insertCustomer(Customer customer) {
        try {
            collection.insertOne(customer);
        } catch (Exception e) {
            System.out.println("Something went Wrong: "+e);
        }
    }

}
```

```java
package com.example.mongodbcrudperations;

import java.util.Date;

import com.example.mongodbcrudperations.dao.DBOperations;
import com.example.mongodbcrudperations.model.Customer;

/**
 * Hello world!
 *
 */
public class App
{
    public static void main( String[] args )
    {
        System.out.println("MongoDB CRUD Operations App" );
        DBOperations operations = new DBOperations();

        Customer customer = new Customer("John Watson", "+91 999999 11111", "john@example.com", 98.4f, new Date(), new Date());
        System.out.println("Customer Details: ");
        System.out.println(customer);

        operations.insertCustomer(customer);
    }
}
```

App.java | Customer.java | DBOperations.java | Util.java

```java
import com.mongodb.client.MongoClient;
import com.mongodb.client.MongoClients;
import com.mongodb.client.MongoCollection;
import com.mongodb.client.MongoDatabase;

public class DBOperations {

    MongoClient client;
    MongoDatabase database;
    MongoCollection collection;

    public DBOperations() {

        try {
            String connectionString = "mongodb+srv://john:john12345@cluster0.acllp.mongodb.net/myFirstDatabase?retryWrites=true&w=majority";
            client = MongoClients.create(connectionString);
            System.out.println("[DBOperations] Connection Created");
            database = client.getDatabase(Util.DB_NAME);
            System.out.println("[DBOperations] Database Selected as eStore");
            collection = database.getCollection(Util.COLLECTION_NAME);
            System.out.println("[DBOperations] Collection from eStore selected as customers");
        } catch (Exception e) {
            System.out.println("Something went Wrong: "+e);
        }


    }

    public void insertCustomer(Customer customer) {
        try {
            collection.insertOne(customer);
        } catch (Exception e) {
            System.out.println("Something went Wrong: "+e);
        }
    }

}
```

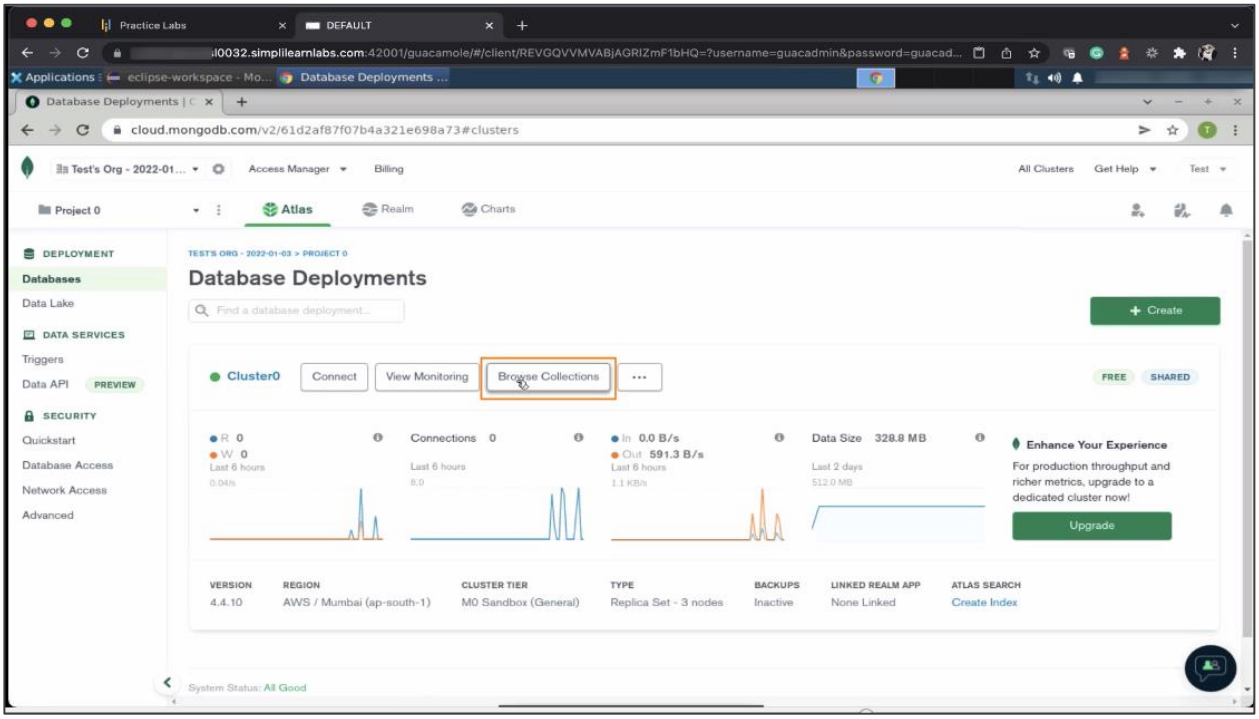
Writable | Smart Insert | 35 : 44 : 1151

Step 8: Create a document

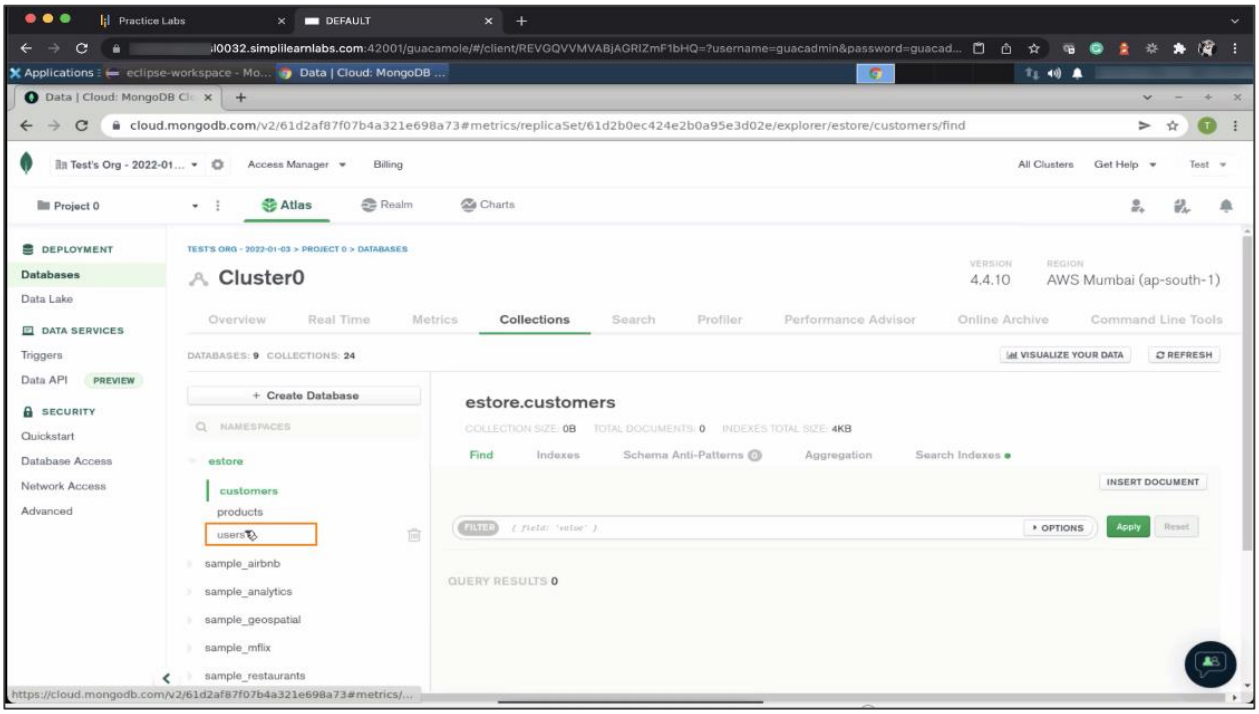
8.1 Create a document in the **Customer.java** file and fetch the details using the **getDocument()** function. Append the customer data to the document using the **append()** function.

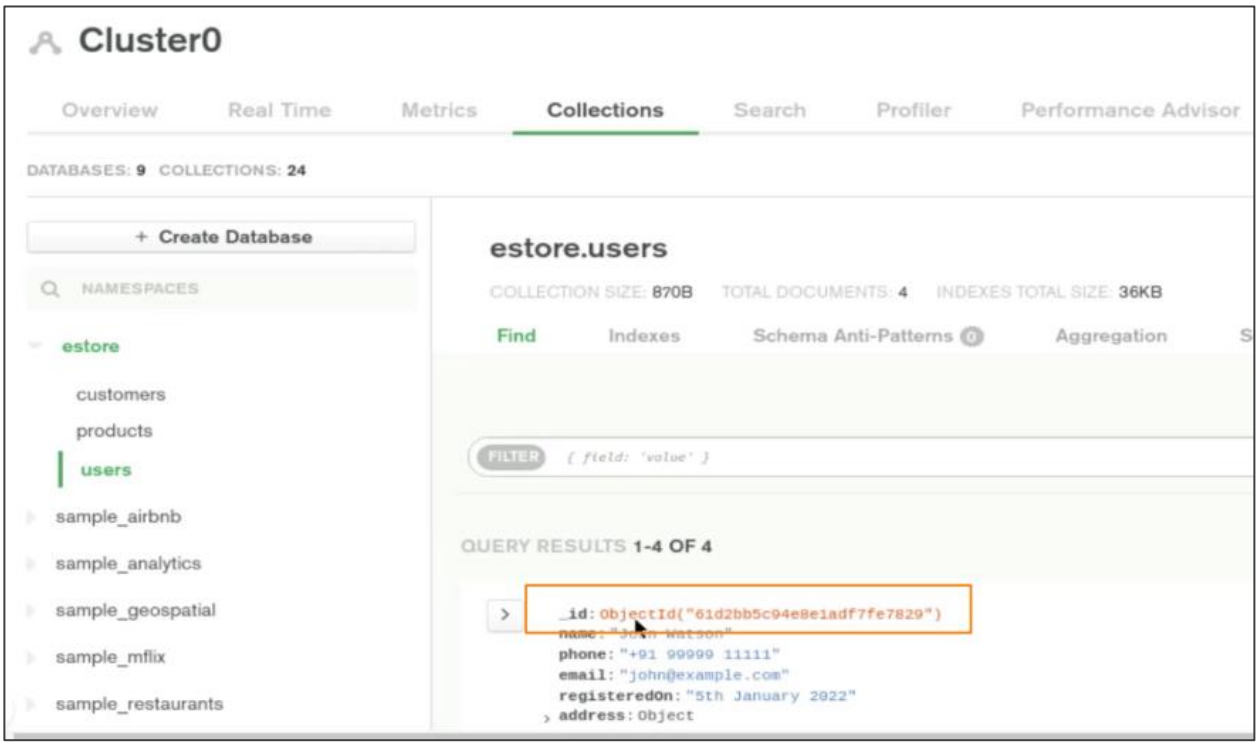
```java
    String name;
    String phone;
    String email;
    Float temperature;
    Date intime;
    Date outtime;

    public Customer() {

    }

    public Customer(String name, String phone, String email, Float temperature, Date intime, Date outtime) {
        this.name = name;
        this.phone = phone;
        this.email = email;
        this.temperature = temperature;
        this.intime = intime;
        this.outtime = outtime;
    }

    @Override
    public String toString() {
        return "Customer [name=" + name + ", phone=" + phone + ", email=" + email + ", temperature=" + temperature
                + ", intime=" + intime + ", outtime=" + outtime + "]";
    }

    public Document getDocument() {
        Document document = new Document();
        document.append("name", name);
        document.append("phone", phone);
        document.append("email", email);
        document.append("temperature", temperature);
        document.append("intime", intime);
        document.append("outtime", outtime);
        return document;
    }

}
```

8.2 Go back to **Databases** and click on **Browser Collections**



8.3 Select the **users** collection and you can find the attribute added as **_id,** which is an identifier for the documents





8.4 To create this id manually, go to the **Customer.java** file and add the **_id** to the document using the **append()** function. Since this gets created automatically, we will comment on it.

```
App.java   Customer.java ×   DBOperations.java   Util.java
10      String name;
11      String phone;
12      String email;
13      Float temperature;
14      Date intime;
15      Date outtime;
16
17      public Customer() {
18
19      }
20
21      public Customer(String name, String phone, String email, Float temperature, Date intime, Date outtime) {
22          this.name = name;
23          this.phone = phone;
24          this.email = email;
25          this.temperature = temperature;
26          this.intime = intime;
27          this.outtime = outtime;
28      }
29
30      @Override
31      public String toString() {
32          return "Customer [name=" + name + ", phone=" + phone + ", email=" + email + ", temperature=" + temperature
33                  + ", intime=" + intime + ", outtime=" + outtime + "]";
34      }
35
36      public Document getDocument() {
37          Document document = new Document();
38          //document.append("_id", new ObjectId());
39          document.append("name", name);
40          document.append("phone", phone);
41          document.append("email", email);
42          document.append("temperature", temperature);
43          document.append("intime", intime);
44          document.append("outtime", outtime);
45          return document;
46      }
47
```

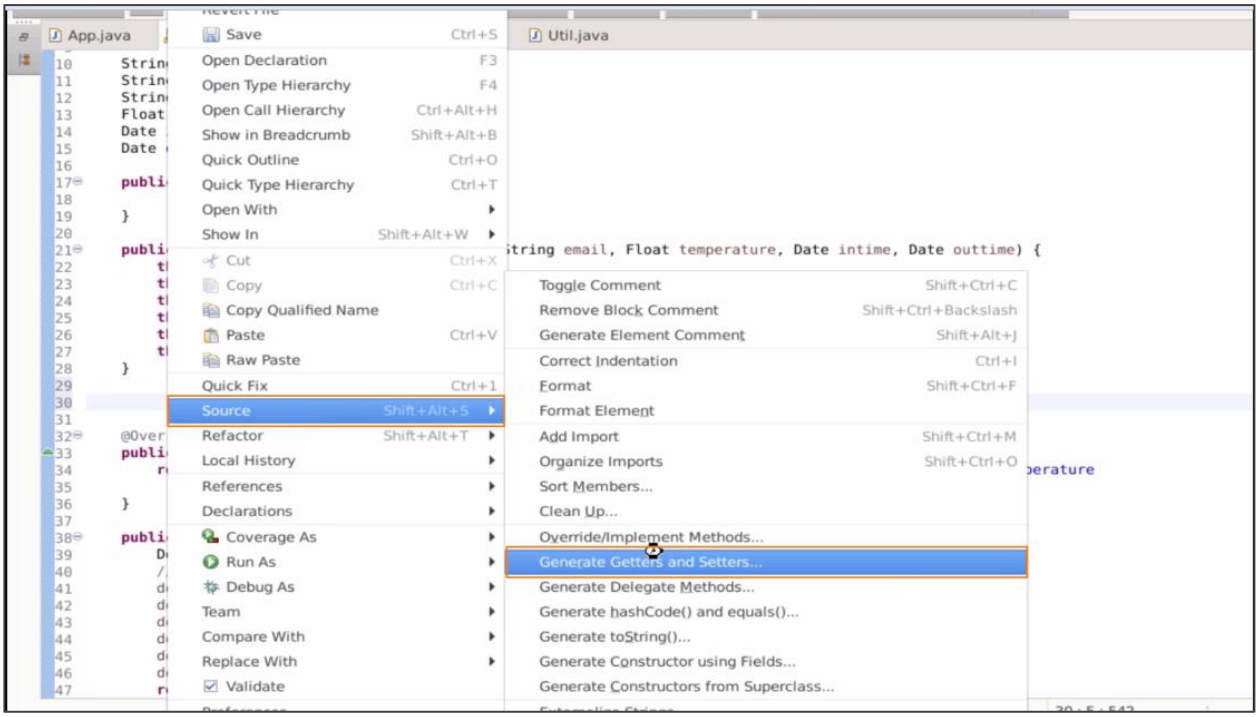
8.5 To insert a new customer, navigate to the **DBOperations.java** file and create a **customer** document object representing the customer's information

```java
import com.mongodb.client.MongoClients;
import com.mongodb.client.MongoCollection;
import com.mongodb.client.MongoDatabase;

public class DBOperations {

    MongoClient client;
    MongoDatabase database;
    MongoCollection collection;

    public DBOperations() {

        try {
            String connectionString = "mongodb+srv://john:john12345@cluster0.acllp.mongodb.net/myFirstDatabase?retryWrites=true&w=ma
            client = MongoClients.create(connectionString);
            System.out.println("[DBOperations] Connection Created");
            database = client.getDatabase(Util.DB_NAME);
            System.out.println("[DBOperations] Database Selected as eStore");
            collection = database.getCollection(Util.COLLECTION_NAME);
            System.out.println("[DBOperations] Collection from eStore selected as customers");
        } catch (Exception e) {
            System.out.println("Something went Wrong: "+e);
        }


    }

    public void insertCustomer(Customer customer) {
        try {
            Document document = customer.getDocument();
            collection.insertOne(document);
            System.out.println("[DBOperations] "+customer);
        } catch (Exception e) {
            System.out.println("Something went Wrong: "+e);
        }
    }

}
```

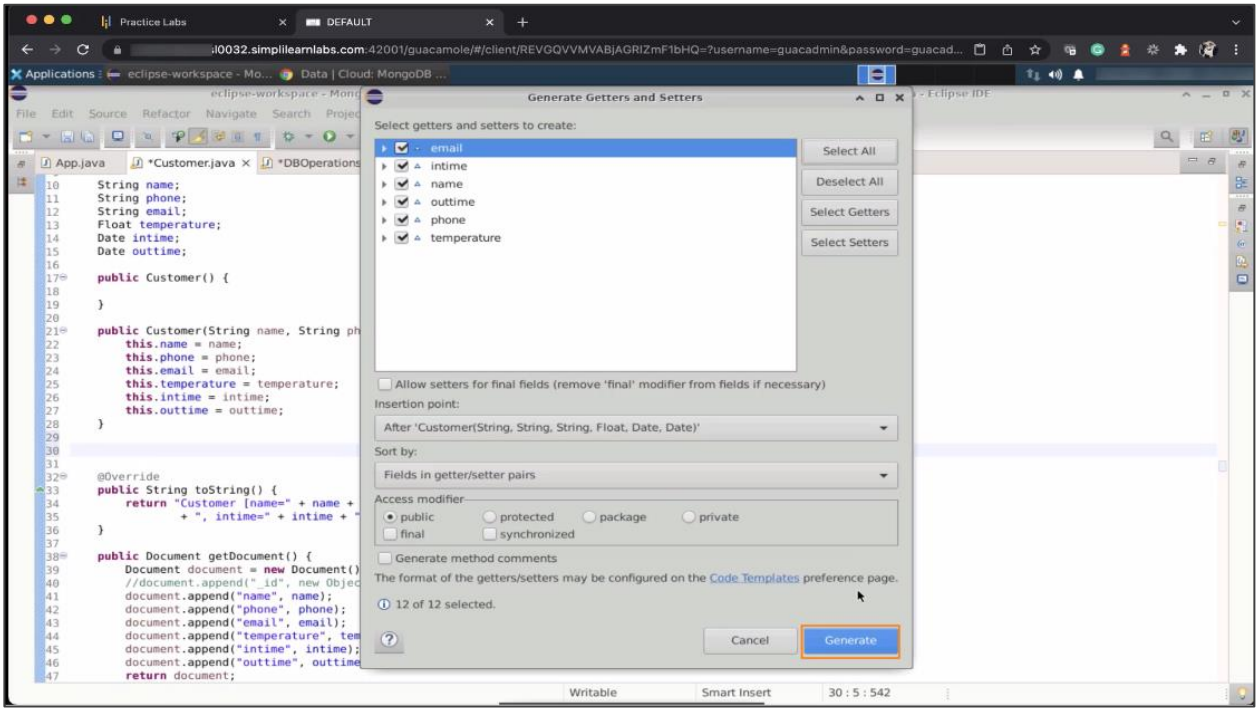
Whenever data is inserted in the MongoDB, it should be a document. Hence, the **document** object is passed as a parameter to the **insertOne()** function.

Step 9: Generate Getters and Setters

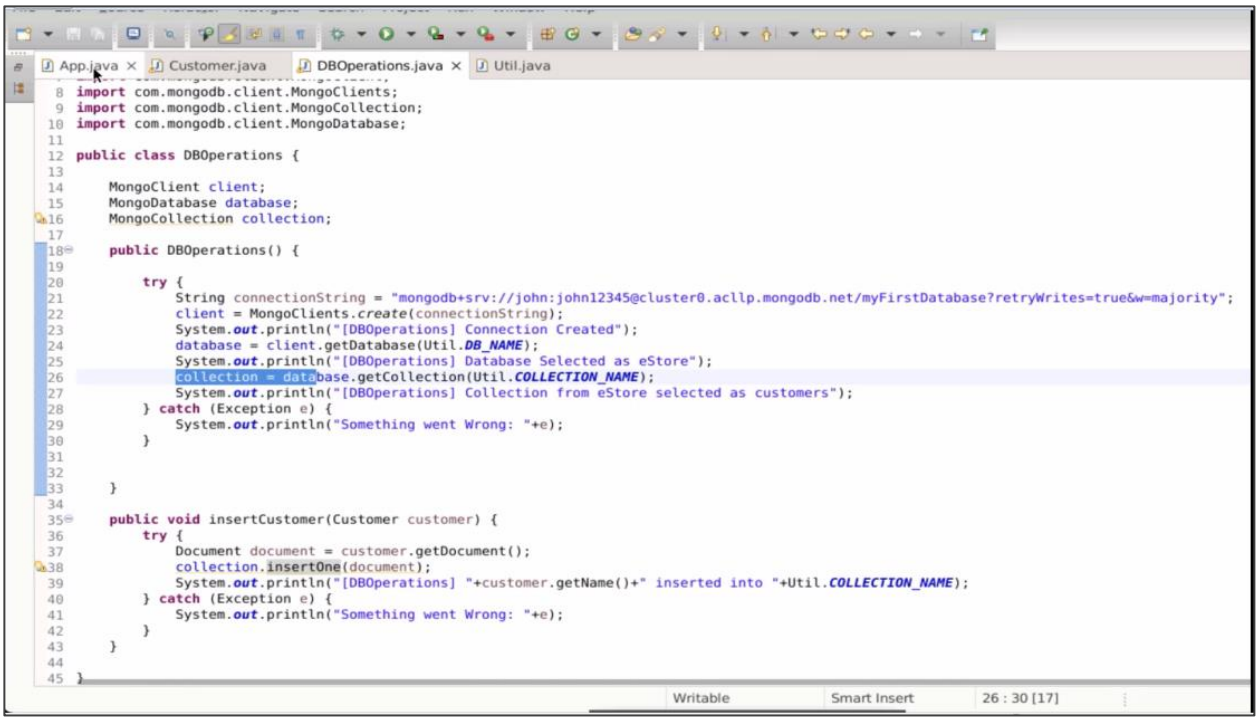
9.1 In **Customer.java**, right-click and select **Source**, then select **Generate Getters and Setters**



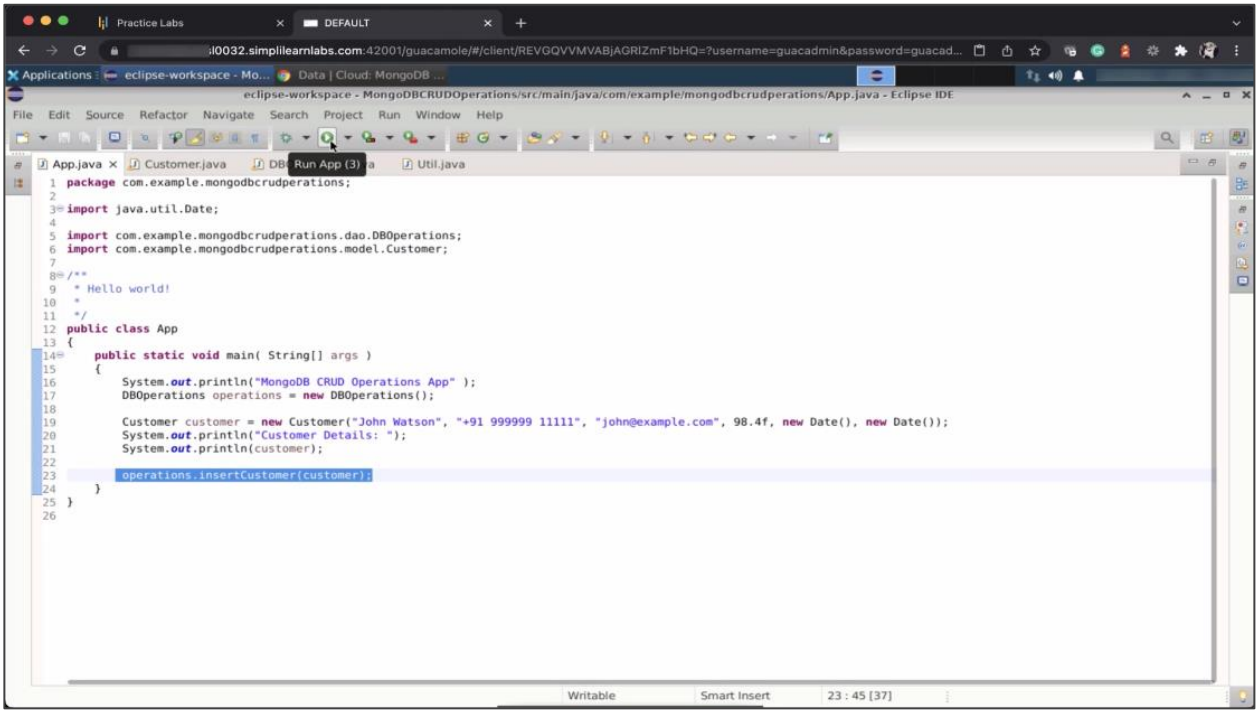
9.2 Click on **Generate**

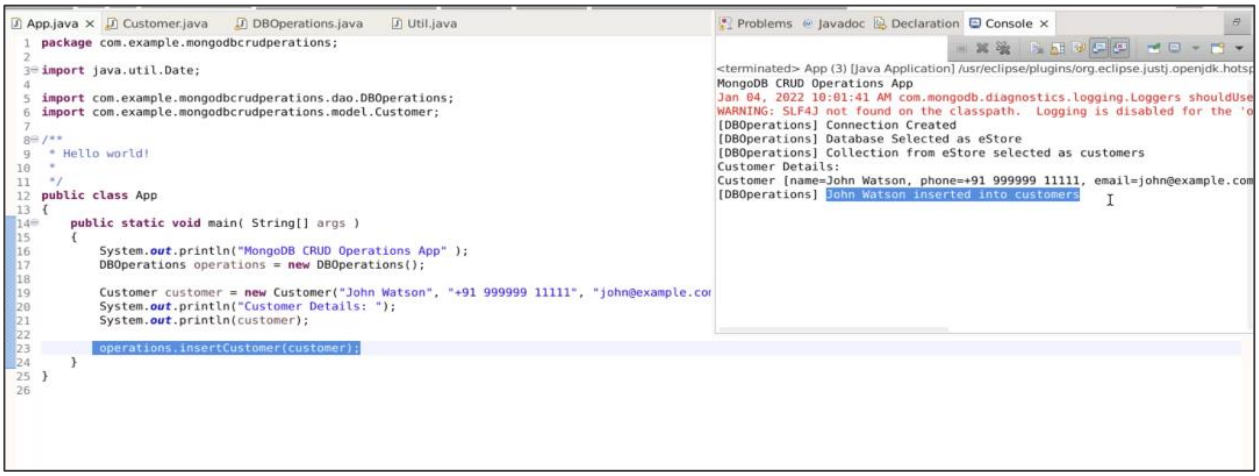


9.3 In **DBOperations.java**, insert the customer name in the **Util.CUSTOMER_NAME** (line 39)



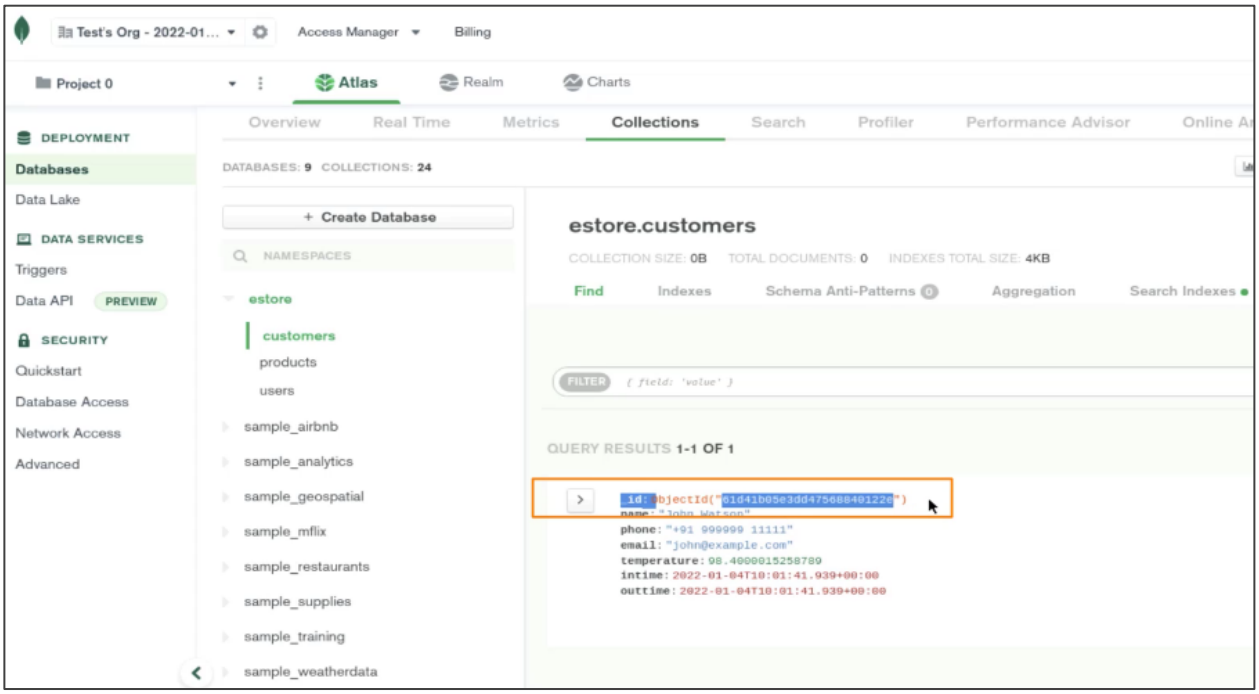
9.4 In **App.java**, run the program and see how the **insertCustomer()** function works





You can see that it says **John Watson inserted into Customers.**

9.5 Go back to **Databases**, navigate to the **customers,** and you can see **_id** coming up automatically, which means there is no need to create that attribute



9.6 In **DBOperations.java**, use the **insertCustomers()** method (lines 48 to 60) to insert multiple customers at once. Create a list of documents and enter the documents.

App.java × Customer.java DBOperations.java × Util.java

```
28              System.out.println("[DBOperations] Database Selected as eStore");
29              collection = database.getCollection(Util.COLLECTION_NAME);
30              System.out.println("[DBOperations] Collection from eStore selected as customers");
31          } catch (Exception e) {
32              System.out.println("Something went Wrong: "+e);
33          }
34
35
36      }
37
38      public void insertCustomer(Customer customer) {
39          try {
40              Document document = customer.getDocument();
41              collection.insertOne(document);
42              System.out.println("[DBOperations] "+customer.getName()+" inserted into "+Util.COLLECTION_NAME);
43          } catch (Exception e) {
44              System.out.println("Something went Wrong: "+e);
45          }
46      }
47
48      public void insertCustomers(List<Customer> customers) {
49          try {
50
51              List<Document> documents = new ArrayList<Document>();
52
53              for(Customer customer : customers) {
54                  documents.add(customer.getDocument());
55              }
56
57              collection.insertMany(documents);
58              System.out.println("[DBOperations] "+customers.size()+" Customers inserted into "+Util.COLLECTION_NAME);
59          } catch (Exception e) {
60              System.out.println("Something went Wrong: "+e);
61          }
62      }
63
64  }
65
```

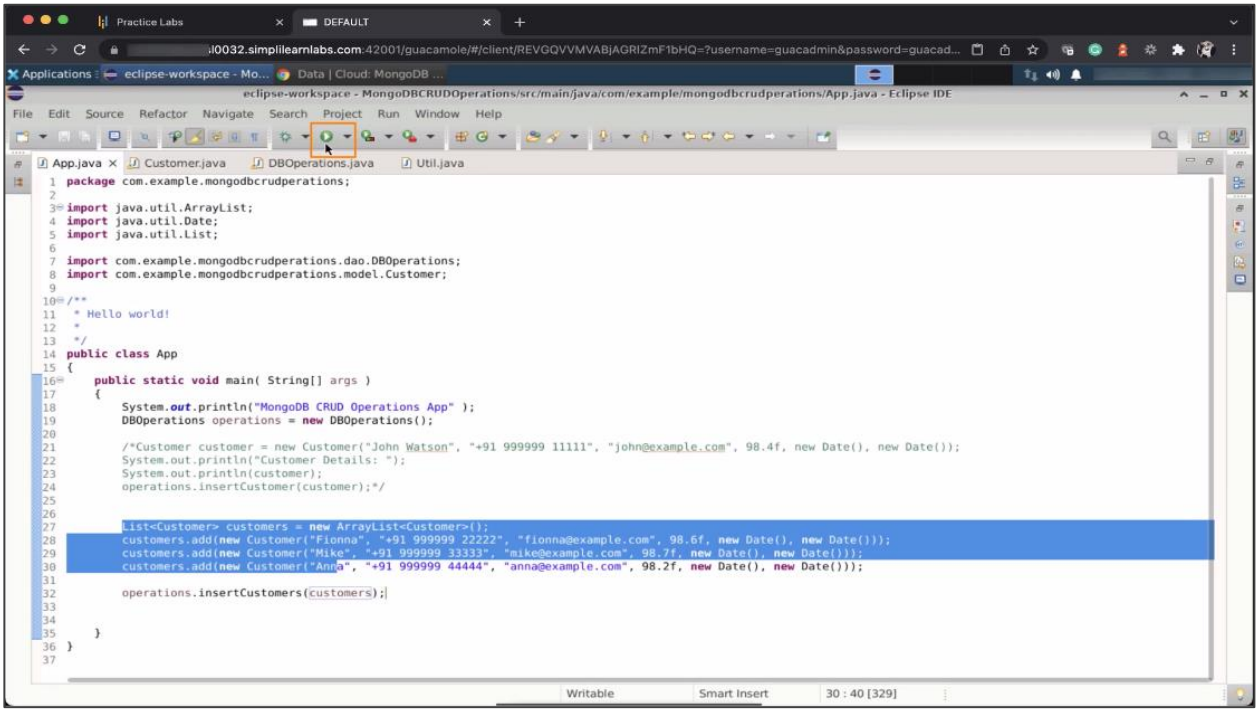
Writable | Smart Insert | 55 : 14 : 1681

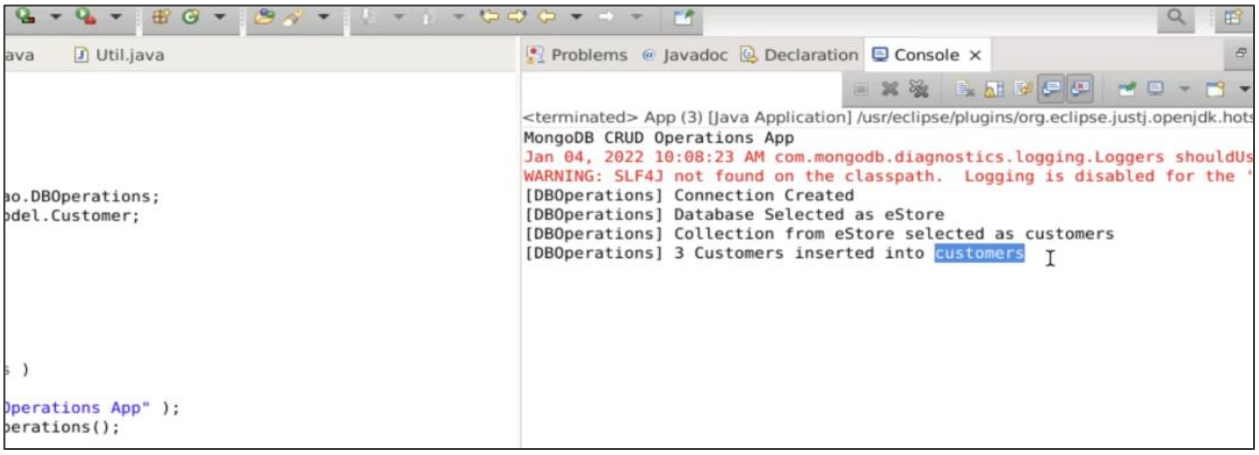
You can see that we have used the **insertMany()** function to insert multiple documents.

9.7 In the **App.java** file, add a list of multiple customers. Create a **customers** object and use the **add()** function to add the new customer.

```java
package com.example.mongodbcrudperations;

import java.util.ArrayList;
import java.util.Date;
import java.util.List;

import com.example.mongodbcrudperations.dao.DBOperations;
import com.example.mongodbcrudperations.model.Customer;

/**
 * Hello world!
 *
 */
public class App
{
    public static void main( String[] args )
    {
        System.out.println("MongoDB CRUD Operations App" );
        DBOperations operations = new DBOperations();

        /*Customer customer = new Customer("John Watson", "+91 999999 11111", "john@example.com", 98.4f, new Date(), new Date());
        System.out.println("Customer Details: ");
        System.out.println(customer);
        operations.insertCustomer(customer);*/


        List<Customer> customers = new ArrayList<Customer>();
        customers.add(new Customer("Fionna", "+91 999999 22222", "fionna@example.com", 98.6f, new Date(), new Date()));
        customers.add(new Customer("Mike", "+91 999999 33333", "mike@example.com", 98.7f, new Date(), new Date()));
        customers.add(new Customer("Anna", "+91 999999 44444", "anna@example.com", 98.2f, new Date(), new Date()));

        operations.insertCustomers(customers);


    }
}
```

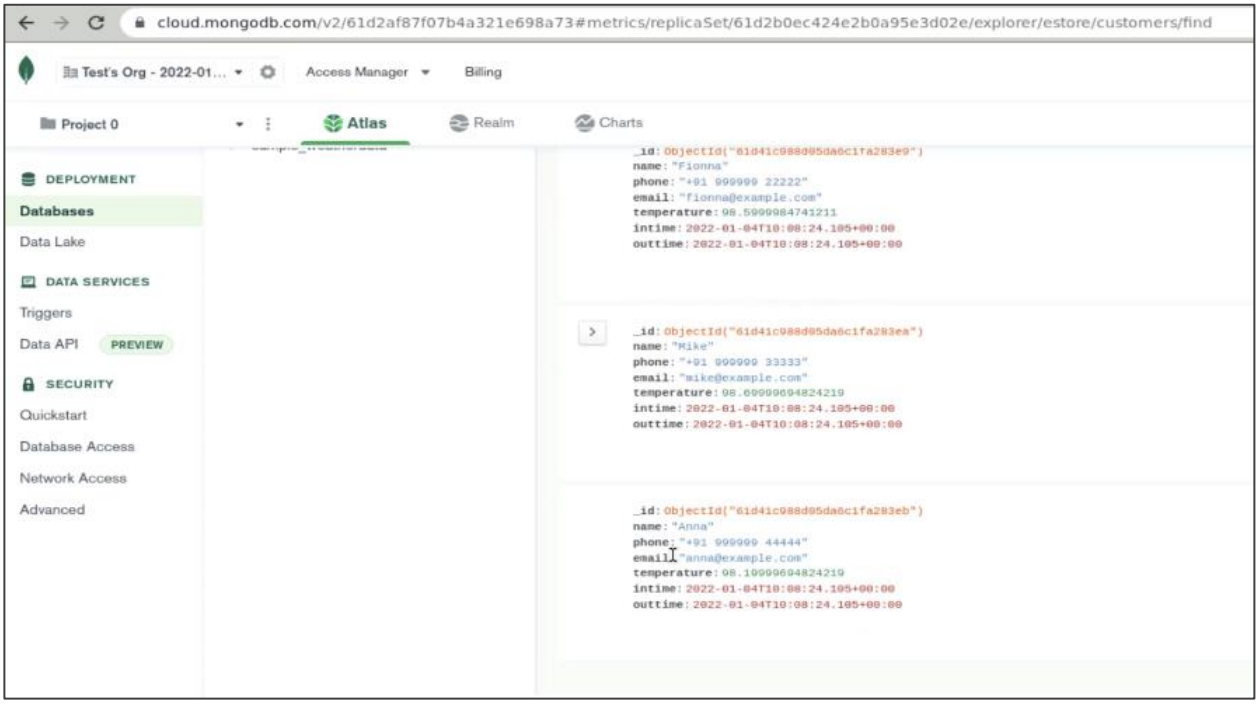
9.8 Save and run the app





You can see that three customers are inserted in the **customers** collection.

9.9 Go to the **Databases** and refresh the **customers** tab



You can see that new customers **Fionna**, **Mike**, and **Anna** have been inserted using the **insertMany()** method. You can use **insertOne()** or **insertMany()** per your requirement.

By following these steps, you have successfully inserted documents in MongoDB using CRUD operations, created a project, and generated Getters and Setters.