

Lesson 01 Demo 07

Updating Documents Using MongoDB CRUD Operations

Objective: To work with the CRUD operations and update the documents for existing and non-existing customers

Tools required: Eclipse IDE

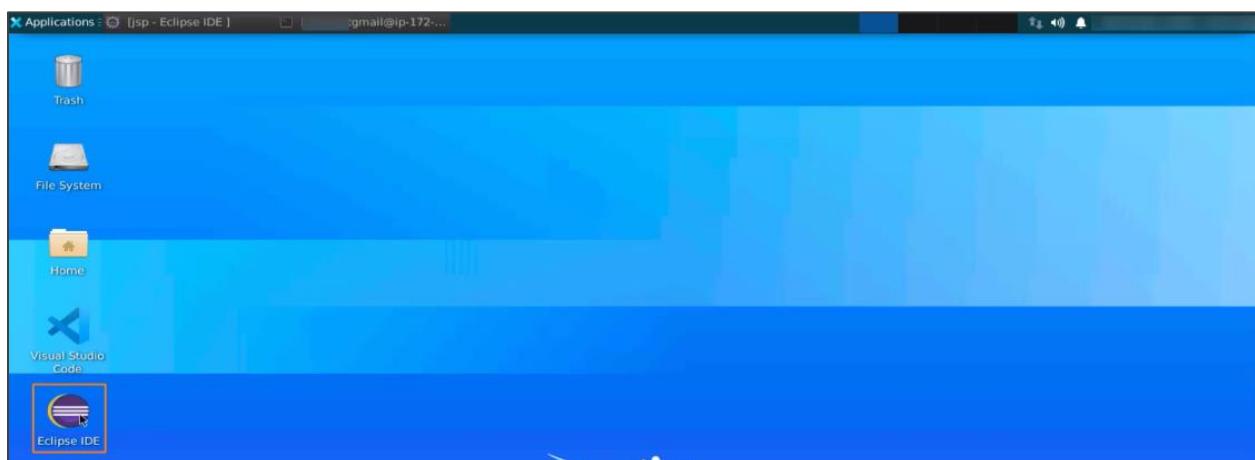
Prerequisites: None

Steps to be followed:

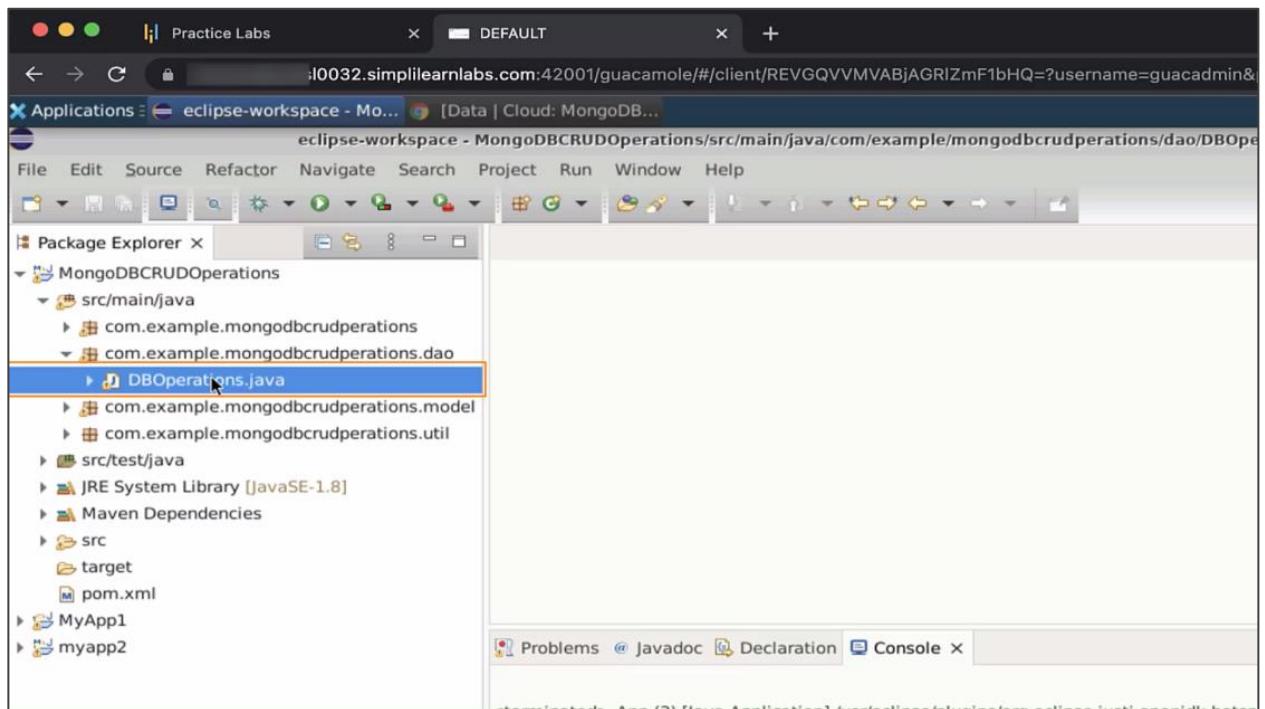
1. Update customer operations
2. Add try catch block
3. Execute another operation for an update
4. Create new function
5. Create more functions
6. Write a new method

Step 1: Update customer operations

1.1 Open the Eclipse IDE



1.2 Navigate to the project **MongoDBCRUDOperations** and open the **DBOperations.java** file



Note: Please refer to the previous demo on how to create the **MongoDBCRUDOperations** project

1.3 Write a new method to update the customer and add inputs to the method

```

    73     customer.setEmail(doc.get("email").toString());
    74
    75     return customer;
    76 }
    77
    78 public List<Customer> getAllCustomers(){
    79     List<Customer> customers = new ArrayList<Customer>();
    80     try {
    81         List<Document> documents = (List<Document>).collection
    82             .find()
    83             .sort(Sorts.ascending("name"))
    84             .into(new ArrayList<Document>());
    85         for(Document doc : documents) {
    86             customers.add(convertDocumentToCustomer(doc));
    87         }
    88     } catch (Exception e) {
    89         System.out.println("Something went Wrong: "+e);
    90     }
    91     return customers;
    92 }
    93
    94
    95 public Customer getCustomerByEmail(String email) {
    96     /*Document filter = new Document("email", email);
    97     Document document = (Document) collection.find(filter).first();*/
    98     Document document = (Document) collection.find(Filters.eq("email", email)).first();
    99     Customer customer = convertDocumentToCustomer(document);
    100    return customer;
    101 }
    102
    103 public void updateCustomer(String email, String key, String value) {
    104     Bson filter = Filters.eq("email", email);
    105     Bson updateOperation = Updates
    106     .set(key, value)
    107     .get();
    108 }
    109
    110 }

```

1.4 Select the `set` method to set the key-value pair to filter the data later

The screenshot shows the same code as above, but with a code completion tooltip for the `set` method appearing over the line `Bson updateOperation = Updates.set`. The tooltip provides the following information:

`set(String fieldName, TItem value) : Bson - Updates`

Creates an update that sets the value of the field with the given name to the given value.

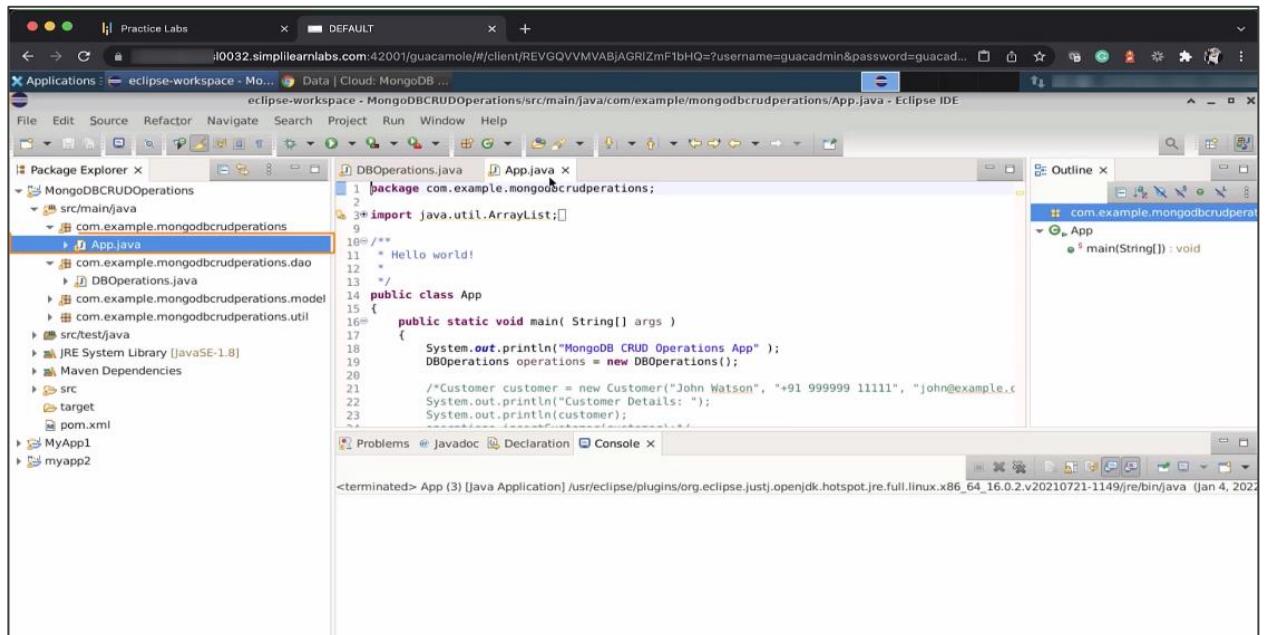
Type Parameters:
`<TItem>` the value type

Parameters:
`fieldName` the non-null field name
`value` the value, which may be null

Returns:
the update

`@mongodb.driver.manual`
reference/operator/update/set/ \$set

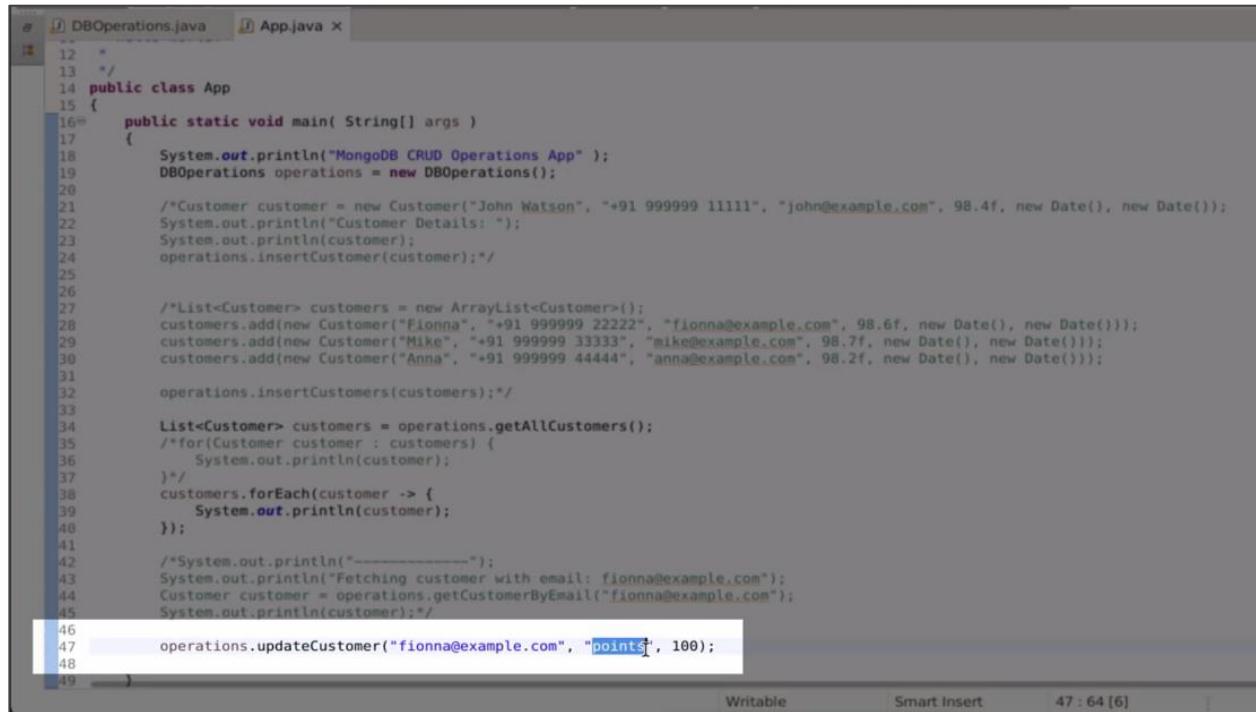
1.5 Navigate back to the App.java file



1.6 Comment out the print statements in the code

The screenshot shows a code editor with the "App.java" file open. A red rectangular box highlights a section of code in the middle of the file, specifically the part where it prints customer details and inserts them into the database. This section is commented out with double slashes (//). The code editor interface includes tabs for "DBOperations.java" and "App.java", and status bars at the bottom indicating "Writable", "Smart Insert", and the timestamp "47 : 9 : 1748".

1.7 Use the **updateCustomer** function to update the details of the customer. Note that the key is **points** and the value is **100**



```
12  *
13  */
14 public class App
15 {
16     public static void main( String[] args )
17     {
18         System.out.println("MongoDB CRUD Operations App");
19         DBOperations operations = new DBOperations();
20
21         /*Customer customer = new Customer("John Watson", "+91 999999 1111", "john@example.com", 98.4f, new Date(), new Date());
22         System.out.println("Customer Details: ");
23         System.out.println(customer);
24         operations.insertCustomer(customer);*/
25
26
27         /*List<Customer> customers = new ArrayList<Customer>();
28         customers.add(new Customer("Fionna", "+91 999999 2222", "fionna@example.com", 98.6f, new Date(), new Date());
29         customers.add(new Customer("Mike", "+91 999999 3333", "mike@example.com", 98.7f, new Date(), new Date()));
30         customers.add(new Customer("Anna", "+91 999999 4444", "anna@example.com", 98.2f, new Date(), new Date()));
31
32         operations.insertCustomers(customers);*/
33
34         List<Customer> customers = operations.getAllCustomers();
35         /*for(Customer customer : customers) {
36             System.out.println(customer);
37         }*/
38         customers.forEach(customer -> {
39             System.out.println(customer);
40         });
41
42         /*System.out.println("-----");
43         System.out.println("Fetching customer with email: fionna@example.com");
44         Customer customer = operations.getCustomerByEmail("fionna@example.com");
45         System.out.println(customer);*/
46         operations.updateCustomer("fionna@example.com", "points", 100);
47
48 }
```

1.8 Go back to the **DBOperations.java**, add a collection and execute the **updateOne** function to perform a single update

The screenshot shows a Java code editor with the file `DBOperations.java` open. The cursor is positioned at line 108, where the `collection.update0` method call is being typed. A tooltip provides information about the `updateOne` method:

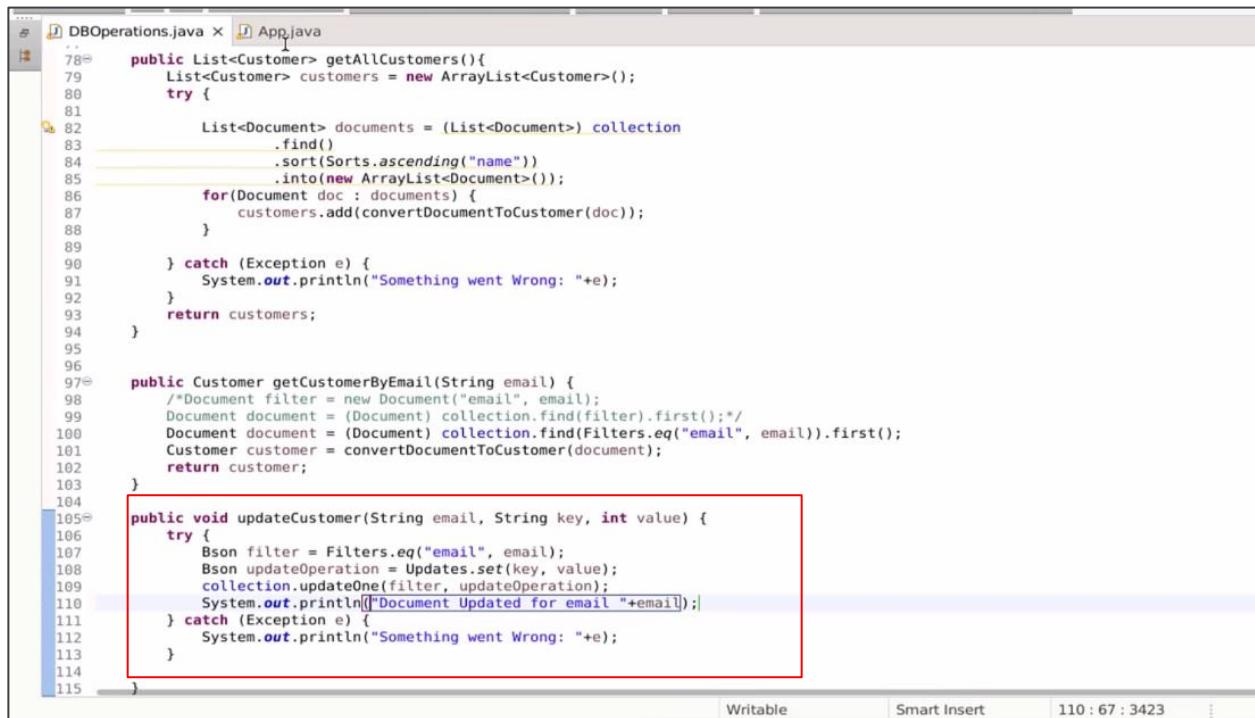
- Update a single document in the collection according to the specified arguments.**
- Use this method to only update the corresponding fields in the document according to the update operators used in the update document. To replace the entire document with a new document, use the corresponding `replaceOne(Bson, Object)` method.**
- Note:** Supports retryable writes on MongoDB server versions 3.6 or higher when the `retryWrites` setting is enabled.
- Parameters:**
 - filter**: a document describing the query filter, which
 - Press 'Tab' from proposal table or click for focus

```

74     return customer;
75   }
76
77   public List<Customer> getAllCustomers(){
78     List<Customer> customers = new ArrayList<Customer>();
79     try {
80       List<Document> documents = (List<Document>) collection
81         .find()
82         .sort(Sorts.ascending("name"))
83         .into(new ArrayList<Document>());
84       for(Document doc : documents) {
85         customers.add(convertDocumentToCustomer(doc));
86       }
87     } catch (Exception e) {
88       System.out.println("Something went Wrong: "+e);
89     }
90     return customers;
91   }
92
93   public Customer
94     /*Document
95      Document do
96      Customer cu
97      return cust
98    */
99
100  public void upd
101    Bson filter
102    Bson update
103    collection.update0
104
105  }
106
107  @
108  }
109
110
111 }
```

Step 2: Add try catch block

2.1 Add a try catch block in the **DBOperations.java** file for the update operation



```
 78    public List<Customer> getAllCustomers(){
 79        List<Customer> customers = new ArrayList<Customer>();
 80        try {
 81            List<Document> documents = (List<Document>) collection
 82                .find()
 83                .sort(Sorts.ascending("name"))
 84                .into(new ArrayList<Document>());
 85            for(Document doc : documents) {
 86                customers.add(convertDocumentToCustomer(doc));
 87            }
 88        } catch (Exception e) {
 89            System.out.println("Something went Wrong: "+e);
 90        }
 91        return customers;
 92    }
 93
 94
 95
 96
 97    public Customer getCustomerByEmail(String email) {
 98        /*Document filter = new Document("email", email);
 99        Document document = (Document) collection.find(filter).first();*/
100        Document document = (Document) collection.find(filters.eq("email", email)).first();
101        Customer customer = convertDocumentToCustomer(document);
102        return customer;
103    }
104
105    public void updateCustomer(String email, String key, int value) {
106        try {
107            Bson filter = Filters.eq("email", email);
108            Bson updateOperation = Updates.set(key, value);
109            collection.updateOne(filter, updateOperation);
110            System.out.println("Document Updated for email "+email);
111        } catch (Exception e) {
112            System.out.println("Something went Wrong: "+e);
113        }
114    }
115}
```

The code editor shows the **DBOperations.java** file. A red box highlights the **try-catch** block within the **updateCustomer** method. The **try** block contains code to update a document in the MongoDB collection using the **Updates.set** method. The **catch** block handles any exceptions that may occur during the update process.

2.2 Go back to App.java and run the code

```

12 /*
13 */
14 public class App
15 {
16     public static void main( String[] args )
17     {
18         System.out.println("MongoDB CRUD Operations App");
19         DBOperations operations = new DBOperations();
20
21         /*Customer customer = new Customer("John Watson", "+91 999999 1111", "john@example.com", 98.4f, new Date(), new Date());
22         System.out.println("Customer Details: ");
23         System.out.println(customer);
24         operations.insertCustomer(customer);*/
25
26
27         /*List<Customer> customers = new ArrayList<Customer>();
28         customers.add(new Customer("Fionna", "+91 999999 2222", "fionna@example.com", 98.6f, new Date(), new Date()));
29         customers.add(new Customer("Mike", "+91 999999 3333", "mike@example.com", 98.7f, new Date(), new Date()));
30         customers.add(new Customer("Anna", "+91 999999 4444", "anna@example.com", 98.2f, new Date(), new Date()));
31
32         operations.insertCustomers(customers);*/
33
34         List<Customer> customers = operations.getAllCustomers();
35         /*for(Customer customer : customers) {
36             System.out.println(customer);
37         }*/
38         customers.forEach(customer -> {
39             System.out.println(customer);
40         });
41
42         /*System.out.println("-----");
43         System.out.println("Fetching customer with email: fionna@example.com");
44         Customer customer = operations.getCustomerByEmail("fionna@example.com");
45         System.out.println(customer);*/
46
47         operations.updateCustomer("fionna@example.com", [points], 100);
48
49     }

```

```

<terminated> App (3) [Java Application] /usr/eclipse/plugins/org.eclipse.justmyjava/jdk.hotspot/bin/java -Dfile.encoding=UTF-8 -jar ./target/app-1.0-SNAPSHOT.jar
MongoDB CRUD Operations App
Jan 04, 2022 3:18:50 PM com.mongodb.diagnostics.logging.Logners should use
WARNING: SLF4J not found on the classpath. Logging is disabled for the 'DBOperations' Connection Created
[DBOperations] Database Selected as eStore
[DBOperations] Collection from eStore selected as customers
Customer [name=Anna, phone=<91 999999 4444, email=anna@example.com, temp=0]
Customer [name=Fionna, phone=<91 999999 2222, email=fionna@example.com, temp=0]
Customer [name=John Watson, phone=<91 999999 1111, email=john@example.com, temp=0]
Customer [name=Mike, phone=<91 999999 3333, email=mike@example.com, temp=0]
Document Updated for email fionna@example.com

```

You can see the output as **Document Updated for email fionna@example.com**.

	Query Results 1-4 of 4
1	<pre>_id: ObjectId("61041c980005aabc1fa230e") name: "John Watson" phone: "+91 999999 11111" email: "john@example.com" temperature: 98.4 intime: 2022-01-04T10:01:41.939+00:00 outtime: 2022-01-04T10:01:41.939+00:00</pre>
2	<pre>_id: ObjectId("61041c980005aabc1fa230e") name: "Fionna" phone: "+91 999999 22222" email: "fionna@example.com" temperature: 98.5 intime: 2022-01-04T10:00:24.105+00:00 outtime: 2022-01-04T10:00:24.105+00:00 points: 100</pre>
3	<pre>_id: ObjectId("61041c980005aabc1fa230e") name: "Mike" phone: "+91 999999 33333" email: "mike@example.com" temperature: 98.6</pre>

A new key-value pair will be added to the database.

2.3 Comment out the `updateCustomer` operation

```

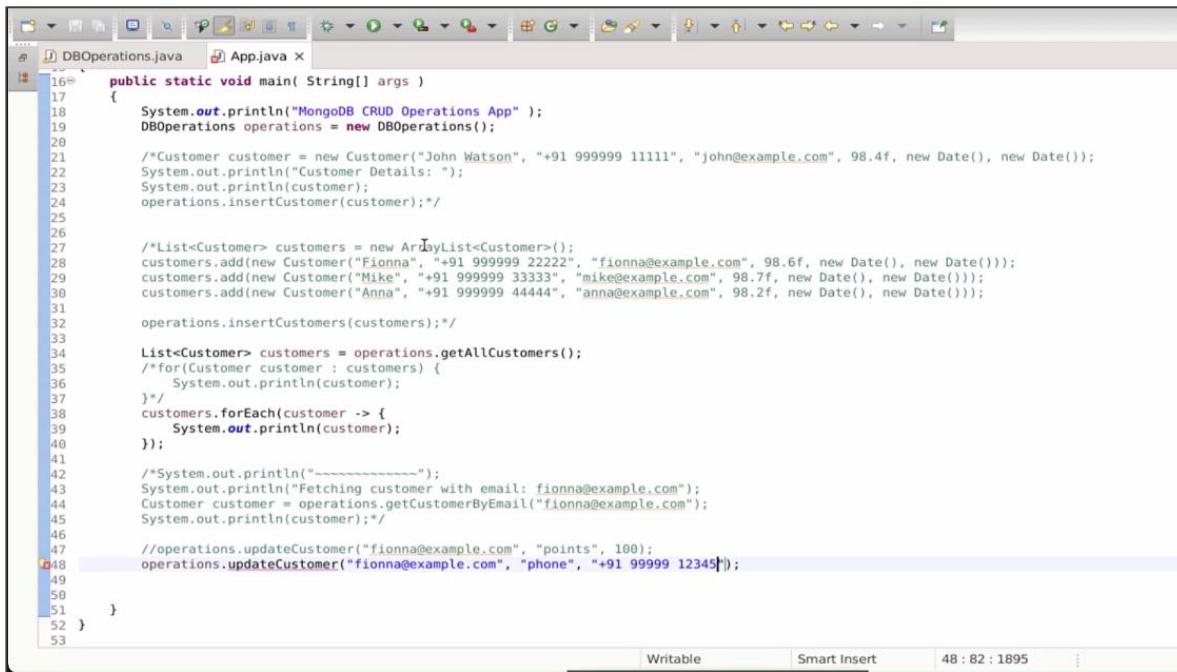
13 /*
14  public class App
15 {
16     public static void main( String[] args )
17     {
18         System.out.println("MongoDB CRUD Operations App" );
19         DBOperations operations = new DBOperations();
20
21         /*Customer customer = new Customer("John Watson", "+91 999999 11111", "john@example.com", 98.4f, new Date(), new Date());
22         System.out.println("Customer Details: ");
23         System.out.println(customer);
24         operations.insertCustomer(customer);
25
26
27         /*List<Customer> customers = new ArrayList<Customer>();
28         customers.add(new Customer("Fionna", "+91 999999 22222", "fionna@example.com", 98.6f, new Date(), new Date()));
29         customers.add(new Customer("Mike", "+91 999999 33333", "mike@example.com", 98.7f, new Date(), new Date()));
30         customers.add(new Customer("Anna", "+91 999999 44444", "anna@example.com", 98.2f, new Date(), new Date()));
31
32         operations.insertCustomers(customers);*/
33
34         List<Customer> customers = operations.getAllCustomers();
35         /*for(Customer customer : customers) {
36             System.out.println(customer);
37         }*/
38         customers.forEach(customer -> {
39             System.out.println(customer);
40         });
41
42         /*System.out.println("~~~~~");
43         System.out.println("Fetching customer with email: fionna@example.com");
44         Customer customer = operations.getCustomerByEmail("fionna@example.com");
45         System.out.println(customer);*/
46
47         //operations.updateCustomer("fionna@example.com", "points", 100);
48
49     }

```

The code block shows a Java application for MongoDB CRUD operations. It includes logic to insert multiple customer documents and then comment out the updateCustomer operation. A red box highlights the commented-out section of the code.

Step 3: Execute another operation for an update

3.1 Write another operation for **updateCustomer** to update the phone number of the customer

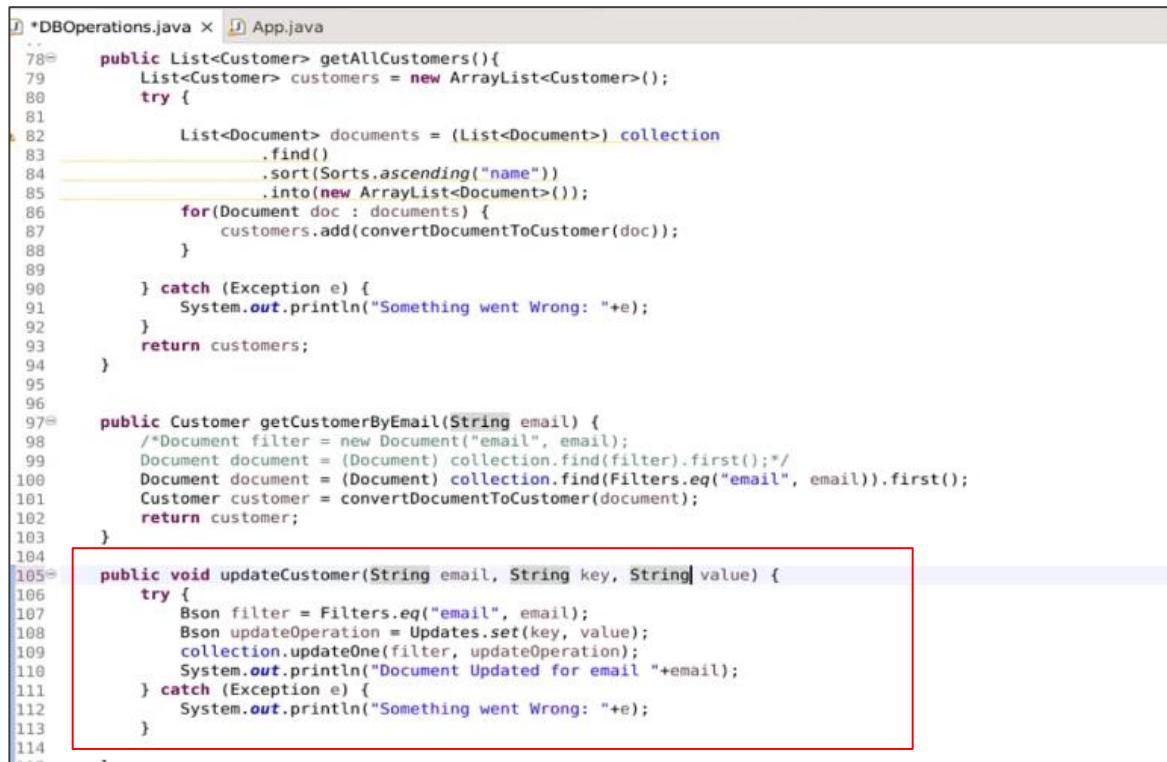


The screenshot shows a Java code editor with two tabs: "DBOperations.java" and "App.java". The "DBOperations.java" tab is active, displaying the following code:

```
16  public static void main( String[] args )
17  {
18      System.out.println("MongoDB CRUD Operations App" );
19      DBOperations operations = new DBOperations();
20
21      /*Customer customer = new Customer("John Watson", "+91 999999 1111", "john@example.com", 98.4f, new Date(), new Date());
22      System.out.println("Customer Details: ");
23      System.out.println(customer);
24      operations.insertCustomer(customer);*/
25
26
27      /*List<Customer> customers = new ArrayList<Customer>();
28      customers.add(new Customer("Fionna", "+91 999999 2222", "fionna@example.com", 98.6f, new Date(), new Date()));
29      customers.add(new Customer("Mike", "+91 999999 3333", "mike@example.com", 98.7f, new Date(), new Date()));
30      customers.add(new Customer("Anna", "+91 999999 4444", "anna@example.com", 98.2f, new Date(), new Date()));
31
32      operations.insertCustomers(customers);*/
33
34      List<Customer> customers = operations.getAllCustomers();
35      /*for(Customer customer : customers) {
36          System.out.println(customer);
37      }*/
38      customers.forEach(customer -> {
39          System.out.println(customer);
40      });
41
42      /*System.out.println("-----");
43      System.out.println("Fetching customer with email: fionna@example.com");
44      Customer customer = operations.getCustomerByEmail("fionna@example.com");
45      System.out.println(customer);*/
46
47      //operations.updateCustomer("fionna@example.com", "points", 100);
48      operations.updateCustomer("fionna@example.com", "phone", "+91 99999 12345");
49
50
51  }
52 }
```

The code implements a `DBOperations` class with methods for inserting and updating customers. The `main` method demonstrates inserting multiple customers and then updating the phone number of the customer with email "fionna@example.com". The line `operations.updateCustomer("fionna@example.com", "phone", "+91 99999 12345");` is highlighted in red, indicating it is the current step being worked on.

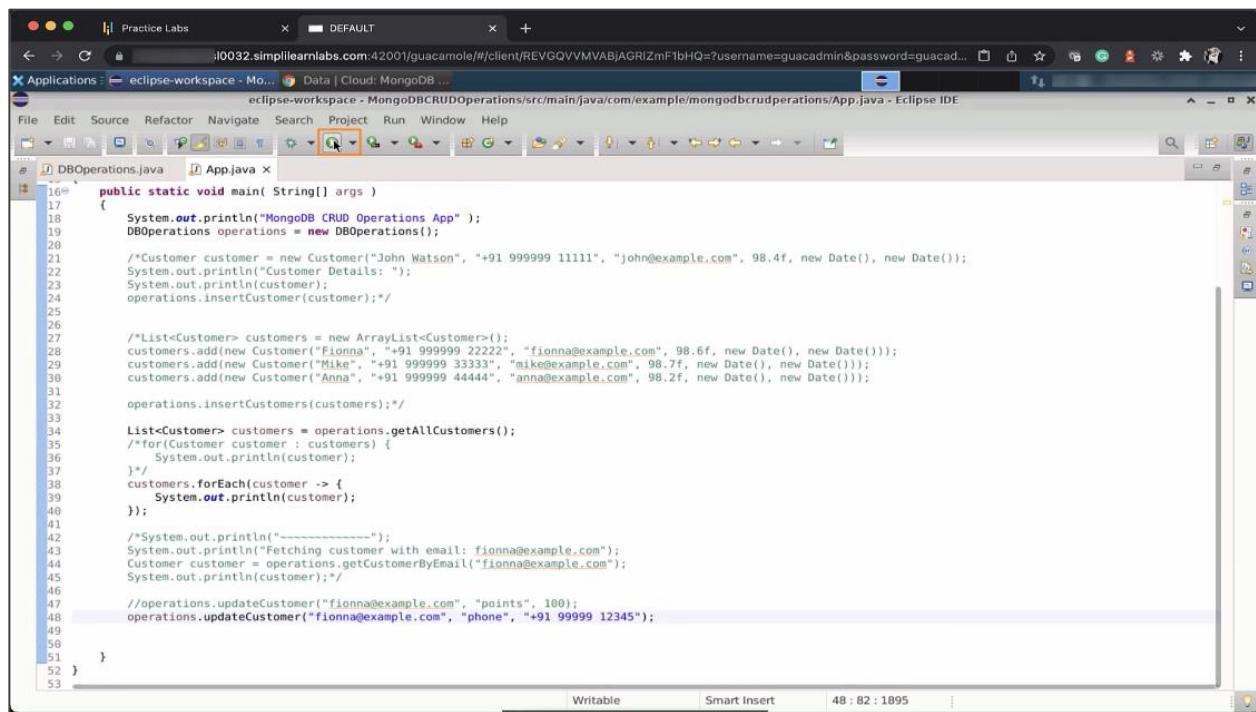
3.2 Go to **DBOperations.java** and change the value to a **String**. Save and run the code again.



```

1 *DBOperations.java x App.java
2
3 public List<Customer> getAllCustomers(){
4     List<Customer> customers = new ArrayList<Customer>();
5     try {
6         List<Document> documents = (List<Document>) collection
7             .find()
8             .sort(Sortsascending("name"))
9             .into(new ArrayList<Document>());
10        for(Document doc : documents) {
11            customers.add(convertDocumentToCustomer(doc));
12        }
13    } catch (Exception e) {
14        System.out.println("Something went Wrong: "+e);
15    }
16    return customers;
17 }
18
19 public Customer getCustomerByEmail(String email) {
20     /*Document filter = new Document("email", email);
21     Document document = (Document) collection.find(filter).first();*/
22     Document document = (Document) collection.find(Filters.eq("email", email)).first();
23     Customer customer = convertDocumentToCustomer(document);
24     return customer;
25 }
26
27 public void updateCustomer(String email, String key, String value) {
28     try {
29         Bson filter = Filters.eq("email", email);
30         Bson updateOperation = Updates.set(key, value);
31         collection.updateOne(filter, updateOperation);
32         System.out.println("Document Updated for email "+email);
33     } catch (Exception e) {
34         System.out.println("Something went Wrong: "+e);
35     }
36 }
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53

```



```

1 Practice Labs x DEFAULT x +
2 http://10032.simplilearnlabs.com:4200/guacamole/#/client/REVGQVVMVABJAGRZmF1bHQ=?username=guacadmin&password=guacadm...
3 Applications: eclipse-workspace - Mo... Data | Cloud: MongoDB ...
4 eclipse-workspace - MongoDBCRUDOperations/src/main/java/com/example/mongodbcrudoperations/App.java - Eclipse IDE
5 File Edit Source Refactor Navigate Search Project Run Window Help
6
7 DBOperations.java x App.java x
8
9 public static void main( String[] args )
10 {
11     System.out.println("MongoDB CRUD Operations App" );
12     DBOperations operations = new DBOperations();
13
14     /*Customer customer = new Customer("John Watson", "+91 999999 1111", "john@example.com", 98.4f, new Date(), new Date());
15     System.out.println("Customer Details: ");
16     System.out.println(customer);
17     operations.insertCustomer(customer);*/
18
19
20     /*List<Customer> customers = new ArrayList<Customer>();
21     customers.add(new Customer("Fionna", "+91 999999 2222", "fionna@example.com", 98.6f, new Date(), new Date()));
22     customers.add(new Customer("Mike", "+91 999999 3333", "mike@example.com", 98.7f, new Date(), new Date()));
23     customers.add(new Customer("Anna", "+91 999999 4444", "anna@example.com", 98.2f, new Date(), new Date()));
24
25     operations.insertCustomers(customers);*/
26
27     List<Customer> customers = operations.getAllCustomers();
28     /*for(Customer customer : customers) {
29         System.out.println(customer);
30     }*/
31     customers.forEach(customer -> {
32         System.out.println(customer);
33     });
34
35     /*System.out.println("-----");
36     System.out.println("Fetching customer with email: fionna@example.com");
37     Customer customer = operations.getCustomerByEmail("fionna@example.com");
38     System.out.println(customer);*/
39
40
41     //operations.updateCustomer("fionna@example.com", "points", 100);
42     operations.updateCustomer("fionna@example.com", "phone", "+91 99999 12345");
43
44
45
46
47
48
49
50
51
52
53

```

You can see the output as **Document Updated for email fionna@example.com.**

```

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 1111", "john@example.com");
    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 2222", "fionna@example.com", 98));
    customers.add(new Customer("Mike", "+91 999999 3333", "mike@example.com", 98.7f));
    customers.add(new Customer("Anna", "+91 999999 4444", "anna@example.com", 98.2f));
    operations.insertCustomers(customers);*/
    List<Customer> customers = operations.getAllCustomers();
    /*for(Customer customer : customers) {
        System.out.println(customer);
    }*/
    customers.forEach(customer -> {
        System.out.println(customer);
    });

    /*System.out.println("-----");
    System.out.println("Fetching customer with email: fionna@example.com");
    Customer customer = operations.getCustomerByEmail("fionna@example.com");
    System.out.println(customer);*/
    //operations.updateCustomer("fionna@example.com", "points", 100);
    operations.updateCustomer("fionna@example.com", "phone", "+91 999999 12345");
}

```

<terminated> App (3) [Java Application] /usr/eclipse/plugins/org.eclipse.jst.java.core.hotsite/MongoDB CRUD Operations App
Jan 04, 2022 3:20:30 PM com.mongodb.diagnostics.logging.Logners shouldUp...
WARNING: SLF4J not found on the classpath. Logging is disabled for the 'org.mongodb.*' package.
[DBOperations] Connection Created
[DBOperations] Database Selected as eStore
[DBOperations] Collection from eStore selected as customers
Customer [name=John, phone=+91 999999 4444, email=john@example.com, temperature=98.0, intime=2022-01-04T10:08:24.105+00:00, outtime=2022-01-04T10:08:24.105+00:00, points=100]
Customer [name=Fionna, phone=+91 999999 2222, email=fionna@example.com, temperature=98.0, intime=2022-01-04T10:08:24.105+00:00, outtime=2022-01-04T10:08:24.105+00:00, points=100]
Customer [name=John Watson, phone=+91 999999 1111, email=john@example.com, temperature=98.0, intime=2022-01-04T10:08:24.105+00:00, outtime=2022-01-04T10:08:24.105+00:00, points=100]
Customer [name=Mike, phone=+91 999999 3333, email=mike@example.com, temperature=98.0, intime=2022-01-04T10:08:24.105+00:00, outtime=2022-01-04T10:08:24.105+00:00, points=100]
Document Updated for email fionna@example.com

3.3 Go back and refresh the collection

The screenshot shows the MongoDB Cloud interface with the URL <https://v2.61d2af87f07b4a321e698a73#metrics/replicaSet/61d2b0ec424e2b0a95e3d02e/explorer/estore/customers/find>. A specific document for 'Fionna' is highlighted, showing the updated phone number '+91 999999 12345'. Other documents for 'Mike' and 'John Watson' are also visible below it.

The phone number for **Fionna** should now be updated.

3.4 Now, use another method **Updates.push** to see how you can update the data

```

79
80     try {
81
82         List<Document> documents = (List<Document>) collection
83             .find()
84             .sort(Sorts.ascending("name"))
85             .into(new ArrayList<Document>());
86
87         for(Document doc : documents) {
88             customers.add(convertDocumentToCustomer(doc));
89         }
90
91     } catch (Exception e) {
92         System.out.println("Something went Wrong: "+e);
93     }
94     return customers;
95
96
97     public Customer getCustomerByEmail(String email) {
98
99         /*Document filter = new Document("email", email);
100        Document document = (Document) collection.find(filter).first();*/
101        Document document = (Document) collection.find(Filters.eq("email", email)).first();
102        Customer customer = convertDocumentToCustomer(document);
103
104    }
105
106    public void updateCustomer(String email, String key, String value) {
107        try {
108            Bson filter = Filters.eq("email", email);
109            //Bson updateOperation = Updates.set(key, value);
110            Bson updateOperation = Updates.push(key, value);
111            collection.updateOne(filter, updateOperation);
112            System.out.println("Document Updated for email "+email);
113        } catch (Exception e) {
114            System.out.println("Something went Wrong: "+e);
115        }
116    }

```

Writable Smart Insert 109 : 48 : 335

3.5 Go back to **App.java** and comment out the earlier part of the code

```

17
18     System.out.println("MongoDB CRUD Operations App");
19     DBOperations operations = new DBOperations();
20
21     /*Customer customer = new Customer("John Watson", "+91 999999 1111", "john@example.com", 98.4f, new Date(), new
22     System.out.println("Customer Details: ");
23     System.out.println(customer);
24     operations.insertCustomer(customer);*/
25
26
27     /*List<Customer> customers = new ArrayList<Customer>();
28     customers.add(new Customer("Fionna", "+91 999999 2222", "fionna@example.com", 98.6f, new Date(), new Date()));
29     customers.add(new Customer("Mike", "+91 999999 3333", "mike@example.com", 98.7f, new Date(), new Date()));
30     customers.add(new Customer("Anna", "+91 999999 4444", "anna@example.com", 98.2f, new Date(), new Date()));
31
32     operations.insertCustomers(customers);*/
33
34     List<Customer> customers = operations.getAllCustomers();
35     /*for(Customer customer : customers) {
36         System.out.println(customer);
37     }*/
38     customers.forEach(customer -> {
39         System.out.println(customer);
40     });
41
42     /*System.out.println("-----");
43     System.out.println("Fetching customer with email: fionna@example.com");
44     Customer customer = operations.getCustomerByEmail("fionna@example.com");
45     System.out.println(customer);*/
46
47     //operations.updateCustomer("fionna@example.com", "points", 100);
48     //operations.updateCustomer("fionna@example.com", "phone", "+91 99999 12345");
49
50
51 }
52 }
53

```

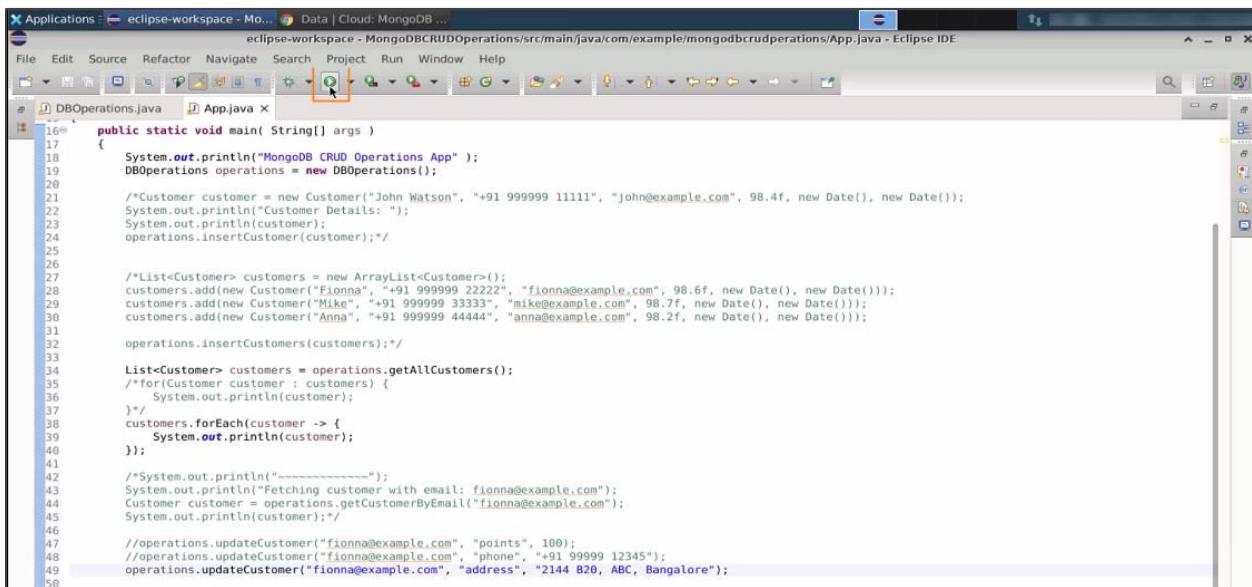
Writable Smart Insert 48 : 11 : 1824

3.6 Use the `updateCustomer` method and enter a new address, for example, **2144 B20, ABC, Bangalore**

```

17   System.out.println("MongoDB CRUD Operations App" );
18   DBOperations operations = new DBOperations();
19
20   /*Customer customer = new Customer("John Watson", "+91 999999 11111", "john@example.com", 98.4f, new Date(), new
21   System.out.println("Customer Details: ");
22   System.out.println(customer);
23   operations.insertCustomer(customer);*/
24
25
26
27   /*List<Customer> customers = new ArrayList<Customer>();
28   customers.add(new Customer("Fionna", "+91 999999 22222", "fionna@example.com", 98.6f, new Date(), new Date()));
29   customers.add(new Customer("Mike", "+91 999999 33333", "mike@example.com", 98.7f, new Date(), new Date()));
30   customers.add(new Customer("Anna", "+91 999999 44444", "anna@example.com", 98.2f, new Date(), new Date()));
31
32   operations.insertCustomers(customers);*/
33
34   List<Customer> customers = operations.getAllCustomers();
35   /*for(Customer customer : customers) {
36     System.out.println(customer);
37   }*/
38   customers.forEach(customer -> {
39     System.out.println(customer);
40   });
41
42   /*System.out.println("~~~~~");
43   System.out.println("Fetching customer with email: fionna@example.com");
44   Customer customer = operations.getCustomerByEmail("fionna@example.com");
45   System.out.println(customer);*/
46
47   //operations.updateCustomer("fionna@example.com", "points", 100);
48   //operations.updateCustomer("fionna@example.com", "phone", "+91 99999 12345");
49   operations.updateCustomer("fionna@example.com", "address", "2144 B20, ABC, Bangalore");
50
51
52 }
53 }
```

3.7 Save the file and run the code



The screenshot shows the Eclipse IDE interface with the following details:

- Project Explorer:** Shows "eclipse-workspace" and "MongoDBCRUDOperations" as selected projects.
- Code Editor:** Displays the `DBOperations.java` file containing Java code for MongoDB CRUD operations.
- Console Output:** Shows the execution results of the code, including the creation of a database, collection, and insertion of documents. It also shows an update operation for a document with email `fionna@example.com`.

```

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 1111", "john@example.com");
    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 2222", "fionna@example.com", 98.6f));
    customers.add(new Customer("Mike", "+91 999999 3333", "mike@example.com", 98.7f));
    customers.add(new Customer("Anna", "+91 999999 4444", "anna@example.com", 98.2f));
    operations.insertCustomers(customers);*/

    List<Customer> customers = operations.getAllCustomers();
    /*for(Customer customer : customers) {
        System.out.println(customer);
    }*/
    customers.forEach(customer -> {
        System.out.println(customer);
    });

    /*System.out.println("-----");
    System.out.println("Fetching customer with email: fionna@example.com");
    Customer customer = operations.getCustomerByEmail("fionna@example.com");
    System.out.println(customer);*/

    //operations.updateCustomer("fionna@example.com", "points", 100);
    //operations.updateCustomer("fionna@example.com", "phone", "+91 999999 12345");
    operations.updateCustomer("fionna@example.com", "address", "2144 B2B, ABC, Bangalore");
}

```

You can see the output as **Document Updated for email fionna@example.com**.

3.8 Go back to the **customers** collection and refresh the document list

The screenshot shows the MongoDB Compass interface with the following details:

- Sidebar:** Shows the database `eStore` and collections `customers`, `products`, and `users`.
- Find Bar:** Contains a `FILTER` button and a placeholder `{ field: 'value' }`.
- Table Headers:** `Find`, `Indexes`, `Schema Anti-Patterns`, `Aggregation`, `Search Index`.
- Table Content:**

QUERY RESULTS 1-4 OF 4	
<code>_id: ObjectId("61d41b05e3dd47568840122e")</code>	<code>name: "John Watson"</code>
	<code>phone: "+91 999999 1111"</code>
	<code>email: "john@example.com"</code>
	<code>temperature: 98.4000015258789</code>
	<code>intime: 2022-01-04T10:01:41.939+00:00</code>
	<code>outtime: 2022-01-04T10:01:41.939+00:00</code>
<code>_id: ObjectId("61d41c988d05da6c1fa283e9")</code>	<code>name: "Fionna"</code>
	<code>phone: "+91 999999 12345"</code>
	<code>email: "fionna@example.com"</code>
	<code>temperature: 98.5999984741211</code>
	<code>intime: 2022-01-04T10:08:24.105+00:00</code>
	<code>outtime: 2022-01-04T10:08:24.105+00:00</code>
	<code>points: 100</code>
	<code>> address: Array</code>

The screenshot shows the MongoDB Compass interface. On the left, a sidebar lists database collections: customers, products, users, sample_airbnb, sample_analytics, sample_geospatial, sample_mflix, sample_restaurants, sample_supplies, sample_training, and sample_weatherdata. The main area is titled "QUERY RESULTS 1-4 OF 4". It displays two documents from the "customers" collection. The first document is for "John Watson" with fields: _id, name, phone, email, temperature, intime, and outtime. The second document is for "Fionna" with fields: _id, name, phone, email, temperature, intime, outtime, points, and address (an array containing a single element: "2144 B20, ABC, Bangalore"). A "FILTERED" button and a "(field: 'value')" search bar are visible at the top.

```
_id: ObjectId("61d41b05e3dd47568840122e")
name: "John Watson"
phone: "+91 999999 11111"
email: "john@example.com"
temperature: 98.4+000015258789
intime: 2022-01-04T10:01:41.939+00:00
outtime: 2022-01-04T10:01:41.939+00:00

_id: ObjectId("61d41c088d05da6c1fa283e9")
name: "Fionna"
phone: "+91 999999 12345"
email: "fionna@example.com"
temperature: 98.5+000984741211
intime: 2022-01-04T10:08:24.105+00:00
outtime: 2022-01-04T10:08:24.105+00:00
points: 100
address: Array
  0: "2144 B20, ABC, Bangalore"
```

The push operation should have created an array of addresses.

Step 4: Create a new function

4.1 In the **DBOperations.java** file, create a new function called **upsertCustomer()** to update new records in the document

```
88         }
89     } catch (Exception e) {
90         System.out.println("Something went Wrong: "+e);
91     }
92     return customers;
93 }
94
95
96
97 public Customer getCustomerByEmail(String email) {
98     /*Document filter = new Document("email", email);
99     Document document = (Document) collection.find(filter).first();*/
100    Document document = (Document) collection.find(Filters.eq("email", email)).first();
101    Customer customer = convertDocumentToCustomer(document);
102    return customer;
103 }
104
105 public void updateCustomer(String email, String key, String value) {
106     try {
107         Bson filter = Filters.eq("email", email);
108         //Bson updateOperation = Updates.set(key, value);
109         Bson updateOperation = Updates.push(key, value);
110         collection.updateOne(filter, updateOperation);
111         System.out.println("Document Updated for email "+email);
112     } catch (Exception e) {
113         System.out.println("Something went Wrong: "+e);
114     }
115 }
116
117
118 public void upsertCustomer(String email, String key, String value) {
119
120
121 }
122 }
```

Writable Smart Insert 118 : 70 : 36

4.2 Create a try catch block, perform a push operation on the key-value pair, and use the **upsert** function to pass the value as **true**

```
99✉  public Customer getCustomerByEmail(String email) {  
100     /*Document filter = new Document("email", email);  
101     Document document = (Document) collection.find(filter).first();*/  
102     Document document = (Document) collection.find(Filters.eq("email", email)).first();  
103     Customer customer = convertDocumentToCustomer(document);  
104     return customer;  
105 }  
106  
107✉  public void updateCustomer(String email, String key, String value) {  
108     try {  
109         Bson filter = Filters.eq("email", email);  
110         //Bson updateOperation = Updates.set(key, value);  
111         Bson updateOperation = Updates.push(key, value);  
112         collection.updateOne(filter, updateOperation);  
113         System.out.println("Document Updated for email "+email);  
114     } catch (Exception e) {  
115         System.out.println("Something went Wrong: "+e);  
116     }  
117 }  
118  
119  
120✉  public void upsertCustomer(String email, String key, String value) {  
121     try {  
122         Bson filter = Filters.eq("email", email);  
123         Bson updateOperation = Updates.push(key, value);  
124         UpdateOptions options = new UpdateOptions();  
125         options.upsert(true);  
126         //collection.updateOne(filter, updateOperation);  
127         UpdateResult result = collection.updateOne(filter, updateOperation, options);  
128         System.out.println("Result is: "+result);  
129         System.out.println("Document Updated for email "+email);  
130     } catch (Exception e) {  
131         System.out.println("Something went Wrong: "+e);  
132     }  
133 }  
134  
135 }
```

Writable Smart Insert 128 : 52 : 4089

4.3 Go back to App.java, use the operation `upsertCustomer(String email, String key, String value)` to update an existing document

The screenshot shows a Java code editor with the following code:

```

19    DocumentOperations operations = new DocumentOperations();
20
21    /*Customer customer = new Customer("John Watson", "+91 999999 11111", "john@example.com", 98.4f, new Date());
22    System.out.println("Customer Details: ");
23    System.out.println(customer);
24    operations.insertCustomer(customer);*/
25
26
27    /*List<Customer> customers = new ArrayList<Customer>();
28    customers.add(new Customer("Fionna", "+91 999999 22222", "fionna@example.com", 98.6f, new Date(), new Date());
29    customers.add(new Customer("Mike", "+91 999999 33333", "mike@example.com", 98.7f, new Date(), new Date()));
30    customers.add(new Customer("Anna", "+91 999999 44444", "anna@example.com", 98.2f, new Date(), new Date()));
31
32    operations.insertCustomers(customers);*/
33
34    List<Customer> customers = operations.getAllCustomers();
35    /*for(Customer customer : customers) {
36        System.out.println(upsertCustomer(String email, String key, String value));
37    }*/
38    customers.forEach(customer -> {
39        System.out.println(upsertCustomer(String email, String key, String value));
40    });
41
42    /*System.out.println(upsertCustomer(String email, String key, String value));
43    System.out.println(upsertCustomer(String email, String key, String value));
44    System.out.println(upsertCustomer(String email, String key, String value));
45    System.out.println(upsertCustomer(String email, String key, String value));
46    System.out.println(upsertCustomer(String email, String key, String value));
47    System.out.println(upsertCustomer(String email, String key, String value));
48    System.out.println(upsertCustomer(String email, String key, String value));
49    System.out.println(upsertCustomer(String email, String key, String value));
50    operations.upse
51
52
53    */

```

A code completion dropdown is open at line 36, showing the method signature `upsertCustomer(String email, String key, String value)`. The dropdown also includes a note: "Press 'Ctrl+Space' to show Template Proposals". The status bar at the bottom right shows "Writable" and "Smart Insert".

4.4 Add inputs to the key-value pair as **feedback** and **A wonderful learning**

The screenshot shows the same Java code editor with the completed code:

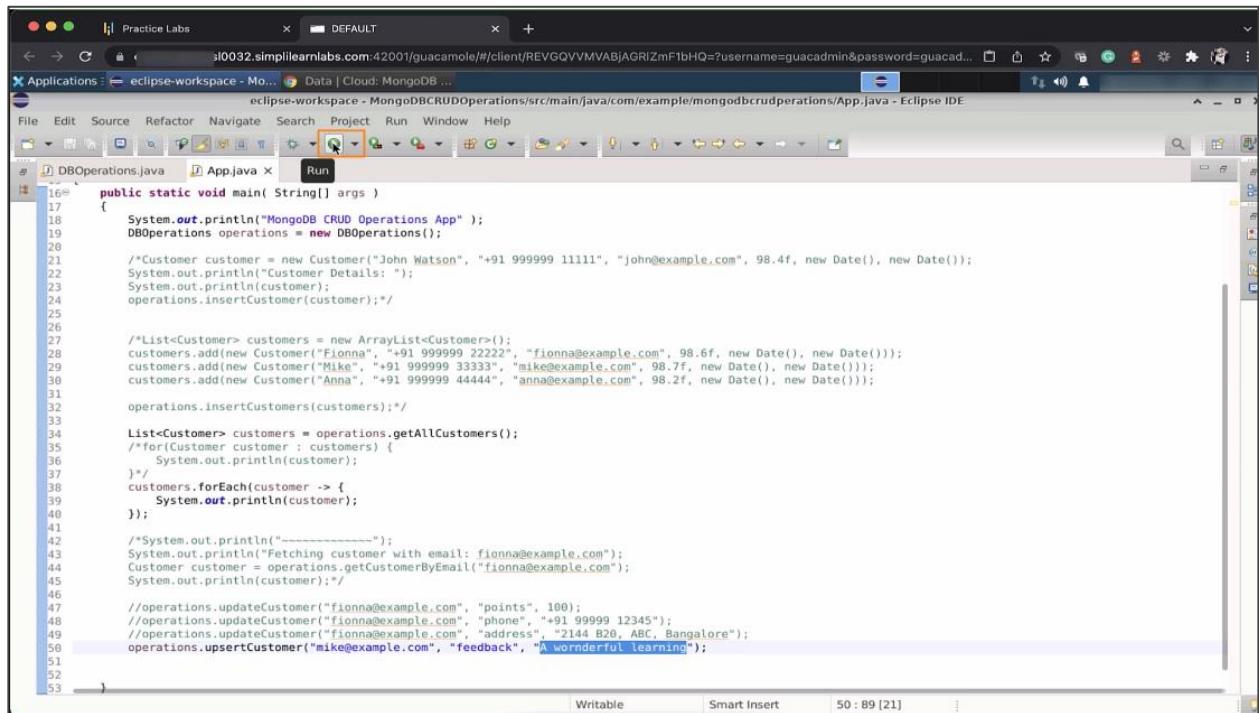
```

25
26
27    /*List<Customer> customers = new ArrayList<Customer>();
28    customers.add(new Customer("Fionna", "+91 999999 22222", "fionna@example.com", 98.6f, new Date());
29    customers.add(new Customer("Mike", "+91 999999 33333", "mike@example.com", 98.7f, new Date(), new Date()));
30    customers.add(new Customer("Anna", "+91 999999 44444", "anna@example.com", 98.2f, new Date(), new Date()));
31
32    operations.insertCustomers(customers);*/
33
34    List<Customer> customers = operations.getAllCustomers();
35    /*for(Customer customer : customers) {
36        System.out.println(customer);
37    }*/
38    customers.forEach(customer -> {
39        System.out.println(customer);
40    });
41
42    /*System.out.println("-----");
43    System.out.println("Fetching customer with email: fionna@example.com");
44    Customer customer = operations.getCustomerByEmail("fionna@example.com");
45    System.out.println(customer);*/
46
47    //operations.updateCustomer("fionna@example.com", "points", 100);
48    //operations.updateCustomer("fionna@example.com", "phone", "+91 999999 12345");
49    //operations.updateCustomer("fionna@example.com", "address", "2144 B20, ABC, Bangalore");
50    operations.upsertCustomer("mike@example.com", "feedback", "A wonderful learning");
51
52
53    */

```

The status bar at the bottom right shows "Writable" and "Smart Insert".

4.5 Save and run the code

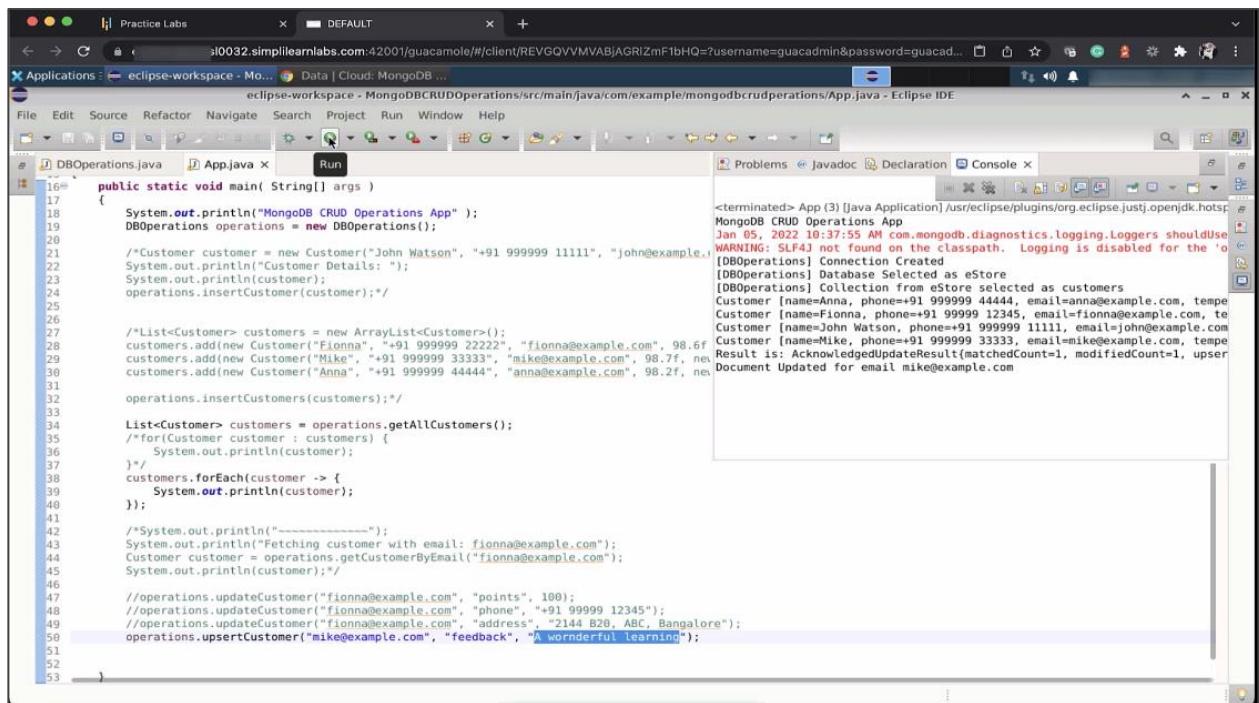


The screenshot shows the Eclipse IDE interface with the 'DBOperations.java' file open in the editor. The 'Run' button in the toolbar is highlighted with an orange box. The code in the editor is a Java application for MongoDB CRUD operations.

```

16  public static void main( String[] args )
17  {
18      System.out.println("MongoDB CRUD Operations App" );
19      DBOperations operations = new DBOperations();
20
21      /*Customer customer = new Customer("John Watson", "+91 999999 1111", "john@example.com", 98.4f, new Date(), new Date());
22      System.out.println("Customer Details: ");
23      System.out.println(customer);
24      operations.insertCustomer(customer);*/
25
26
27      /*List<Customer> customers = new ArrayList<Customer>();
28      customers.add(new Customer("Fionna", "+91 999999 2222", "fionna@example.com", 98.6f, new Date(), new Date()));
29      customers.add(new Customer("Mike", "+91 999999 3333", "mike@example.com", 98.7f, new Date(), new Date()));
30      customers.add(new Customer("Anna", "+91 999999 4444", "anna@example.com", 98.2f, new Date(), new Date()));
31
32      operations.insertCustomers(customers);*/
33
34      List<Customer> customers = operations.getAllCustomers();
35      /*for(Customer customer : customers) {
36          System.out.println(customer);
37      }*/
38      customers.forEach(customer -> {
39          System.out.println(customer);
40      });
41
42      /*System.out.println("-----");
43      System.out.println("Fetching customer with email: fionna@example.com");
44      Customer customer = operations.getCustomerByEmail("fionna@example.com");
45      System.out.println(customer);*/
46
47      //operations.updateCustomer("fionna@example.com", "points", 100);
48      //operations.updateCustomer("fionna@example.com", "phone", "+91 99999 12345");
49      //operations.updateCustomer("fionna@example.com", "address", "2144 B20, ABC, Bangalore");
50      operations.upsertCustomer("mike@example.com", "feedback", "A wonderful learning");
51
52
53 }

```



The screenshot shows the Eclipse IDE interface with the 'Console' tab selected. The output window displays the execution log for the Java application. It shows the application starting, connecting to the database, and performing various operations like inserting and updating documents.

```

<terminated> App (3) [java Application] /usr/eclipse/plugins/org.eclipse.jst.jdt.openjdk.hotspot
MongoDB CRUD Operations App
Jan 05, 2022 10:37:55 AM com.mongodb.diagnostics.logging.Loggers shouldUse
WARNING: SLF4J not found on the classpath. Logging is disabled for the 'org.mongodb.driver' package.
[DBOperations] Connection Created
[DBOperations] Database Selected as eStore
[DBOperations] Collection from eStore selected as customers
Customer [name=Fionna, phone=+91 999999 2222, email=fionna@example.com, temp_email=null]
Customer [name=John Watson, phone=+91 999999 1111, email=john@example.com, temp_email=null]
Customer [name=Mike, phone=+91 999999 3333, email=mike@example.com, temp_email=null]
Customer [name=Anna, phone=+91 999999 4444, email=anna@example.com, temp_email=null]
Result is: AcknowledgedUpdateResult{matchedCount=1, modifiedCount=1, upsertedCount=1}
Document Updated for email mike@example.com

```

You can see the output as Document Updated for email mike@example.com.

```

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 1111", "john@example.com", "password");
    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 2222", "fionna@example.com", "password"));
    customers.add(new Customer("Mike", "+91 999999 3333", "mike@example.com", "password"));
    customers.add(new Customer("Anna", "+91 999999 4444", "anna@example.com", "password"));

    operations.insertCustomers(customers);*/

    List<Customer> customers = operations.getAllCustomers();
    for(Customer customer : customers) {
        System.out.println(customer);
    }
    customers.forEach(customer -> {
        System.out.println(customer);
    });

    System.out.println("-----");
    System.out.println("Fetching customer with email: fionna@example.com");
    Customer customer = operations.getCustomerByEmail("fionna@example.com");
    System.out.println(customer);

    //operations.updateCustomer("fionna@example.com", "points", 100);
    //operations.updateCustomer("fionna@example.com", "phone", "+91 999999 12345");
    //operations.updateCustomer("fionna@example.com", "address", "2144 B20, ABC, Bangalore");
    operations.upsertCustomer("mike@example.com", "feedback", "A wonderful learning");
}

```

The screenshot shows the Eclipse IDE interface with the Java code for MongoDB CRUD operations. The code creates a `Customer` object, adds it to a list, and then prints all customers. It also includes logic for fetching a customer by email and updating their points or phone number. The right side of the screen displays the execution output in the Console tab, showing the printed customer details and the result of the upsert operation.

You can see the results state as **AcknowledgeUpdateResult**.

4.6 Go back to the database and refresh the screen

The screenshot shows the MongoDB Atlas UI with the 'customers' collection selected. The collection contains three documents:

- Fionna**: `_id: ObjectId("61d41c988d85da6c1fa283e9")`, `name: "Fionna"`, `phone: "+91 99999 12345"`, `email: "fionna@example.com"`, `temperature: 98.5999984741211`, `intime: 2022-01-04T10:08:24.105+00:00`, `outtime: 2022-01-04T10:08:24.105+00:00`, `points: 100`, `address: Array`
- Mike**: `_id: ObjectId("61d41c988d85da6c1fa283ea")`, `name: "Mike"`, `phone: "+91 99999 3333"`, `email: "mike@example.com"`, `temperature: 98.0999994824219`, `intime: 2022-01-04T10:08:24.105+00:00`, `outtime: 2022-01-04T10:08:24.105+00:00`
- Anna**: `_id: ObjectId("61d41c988d85da6c1fa283eb")`, `name: "Anna"`, `phone: "+91 99999 4444"`, `email: "anna@example.com"`, `temperature: 98.1999994824219`, `intime: 2022-01-04T10:08:24.105+00:00`, `outtime: 2022-01-04T10:08:24.105+00:00`

You can see the field is updated.

```

sample_weatherdata
{
  "_id": "ObjectId(\"61d41c988d05da6c1fa283ea\")",
  "name": "Fionna",
  "phone": "+91 99999 12345",
  "email": "fionna@example.com",
  "temperature": 98.5,
  "intime": 2022-01-04T10:08:24.105+00:00,
  "outtime": 2022-01-04T10:08:24.105+00:00,
  "points": 100,
  "address": []
}

{
  "_id": "ObjectId(\"61d41c988d05da6c1fa283ea\")",
  "name": "Mike",
  "phone": "+91 99999 33333",
  "email": "mike@example.com",
  "temperature": 98.6,
  "intime": 2022-01-04T10:08:24.105+00:00,
  "outtime": 2022-01-04T10:08:24.105+00:00,
  "feedback": []
}

{
  "_id": "ObjectId(\"61d41c988d05da6c1fa283eb\")",
  "name": "Anna",
  "phone": "+91 99999 44444",
  "email": "anna@example.com",
  "temperature": 98.1,
  "intime": 2022-01-04T10:08:24.105+00:00,
  "outtime": 2022-01-04T10:08:24.105+00:00
}

```

4.7 Go back to **App.java** and change the inputs to **leo@example.com**, which is non-existing data

```

16  public static void main( String[] args )
17  {
18      System.out.println("MongoDB CRUD Operations App" );
19      DBOperations operations = new DBOperations();
20
21      /*Customer customer = new Customer("John Watson", "+91 999999 11111", "john@example.com", 98.4f, new Date(), new Date());
22      System.out.println("Customer Details: ");
23      System.out.println(customer);
24      operations.insertCustomer(customer);*/
25
26
27      /*List<Customer> customers = new ArrayList<Customer>();
28      customers.add(new Customer("Fionna", "+91 999999 22222", "fionna@example.com", 98.6f, new Date(), new Date()));
29      customers.add(new Customer("Mike", "+91 999999 33333", "mike@example.com", 98.7f, new Date(), new Date());
30      customers.add(new Customer("Anna", "+91 999999 44444", "anna@example.com", 98.2f, new Date(), new Date()));
31
32      operations.insertCustomers(customers);*/
33
34      List<Customer> customers = operations.getAllCustomers();
35      /*for(Customer customer : customers) {
36          System.out.println(customer);
37      }*/
38      customers.forEach(customer -> {
39          System.out.println(customer);
40      });
41
42      /*System.out.println("-----");
43      System.out.println("Fetching customer with email: fionna@example.com");
44      Customer customer = operations.getCustomerByEmail("fionna@example.com");
45      System.out.println(customer);*/
46
47      //operations.updateCustomer("fionna@example.com", "points", 100);
48      //operations.updateCustomer("fionna@example.com", "phone", "+91 99999 12345");
49      //operations.updateCustomer("fionna@example.com", "address", "2144-B20, ABC, Bangalore");
50      operations.upsertCustomer("leo@example.com", "feedback", "A wonderful learning experience!");
51
52  }

```

4.8 Save and run the code

The screenshot shows the NetBeans IDE interface with the following details:

- File Menu:** File, Edit, Source, Refactor, Navigate, Search, Project, Run, Window, Help.
- Toolbar:** Includes icons for New, Open, Save, Cut, Copy, Paste, Find, and others.
- Code Editor:** Displays Java code for a MongoDB CRUD Operations App. The code includes methods for inserting, updating, and fetching customers from a MongoDB database.
- Code Content:**

```
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 1111", "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 2222", "fionna@example.com", 98.6f, new Date(), new Date()));
    customers.add(new Customer("Mike", "+91 999999 3333", "mike@example.com", 98.7f, new Date(), new Date()));
    customers.add(new Customer("Anna", "+91 999999 4444", "anna@example.com", 98.2f, new Date(), new Date()));

    operations.insertCustomers(customers);*/

    List<Customer> customers = operations.getAllCustomers();
    /*for(Customer customer : customers) {
        System.out.println(customer);
    }*/
    customers.forEach(customer -> {
        System.out.println(customer);
    });

    /*System.out.println("-----");
    System.out.println("Fetching customer with email: fionna@example.com");
    Customer customer = operations.getCustomerByEmail("fionna@example.com");
    System.out.println(customer);*/

    //operations.updateCustomer("fionna@example.com", "points", 100);
    //operations.updateCustomer("fionna@example.com", "phone", "+91 99999 12345");
    //operations.updateCustomer("fionna@example.com", "address", "2144 B20, ABC, Bangalore");
    operations.upsertCustomer("leo@example.com", "feedback", "A wonderful learning experience");
}
```

You can see that in the acknowledgement result, no match was found, that is, **modifiedCount=0**.

4.9 Go back to the database and refresh the screen

The screenshot shows the MongoDB Atlas Data Explorer interface. On the left, the sidebar includes 'Project 0', 'DEPLOYMENT', 'Databases' (selected), 'Data Lake', 'DATA SERVICES', 'Triggers', 'Data API' (with a 'PREVIEW' button), and 'SECURITY'. The main area displays a database named 'sample_weatherdata'. It contains two documents:

```

_id: ObjectId("61d41c988d05da0c1fa283e9")
name: "Fionna"
phone: "+91 99999 12345"
email: "fionna@example.com"
temperature: 98.599984741211
intime: 2022-01-04T10:08:24.105+00:00
outtime: 2022-01-04T10:08:24.105+00:00
points: 100
address: Array

_id: ObjectId("61d41c988d05da0c1fa283ea")
name: "Mike"
phone: "+91 99999 33333"
email: "mike@example.com"
temperature: 98.6999664824219
intime: 2022-01-04T10:08:24.105+00:00
outtime: 2022-01-04T10:08:24.105+00:00
feedback: Array
  0: "A wonderful learning"

```

You can see the **upsertOperation** in action.

This screenshot shows the same MongoDB Atlas Data Explorer interface after an upsert operation. The 'Data API' section is highlighted. The document for 'Mike' now has a 'feedback' field with one element: "A wonderful learning".

Step 5: Create more functions

5.1 Create a new function `upsertManyCustomers(String email, String key, String value)` to update multiple documents

```

public void upsertCustomer(String email, String key, String value) {
    try {
        Bson filter = Filters.eq("email", email);
        Bson updateOperation = Updates.push(key, value);
        UpdateOptions options = new UpdateOptions();
        options.upsert(true);
        //collection.updateOne(filter, updateOperation);
        UpdateResult result = collection.updateOne(filter, updateOperation, options);
        System.out.println("Result is: "+result);
        System.out.println("Document Updated for email "+email);
    } catch (Exception e) {
        System.out.println("Something went Wrong: "+e);
    }
}

public void upsertManyCustomers() {
    try {
        Bson filter = Filters.gte("points", 50);
        Bson updateOperation = Updates.push("offer", "CASHBACK200");
        UpdateResult result = collection.updateMany(filter, updateOperation);
        System.out.println("Result is: "+result);
    } catch (Exception e) {
        System.out.println("Something went Wrong: "+e);
    }
}

```

5.2 Add the field points in the database

The screenshot shows the MongoDB Atlas interface with the 'Data API' tab selected. On the left, there's a sidebar with 'Project 0', 'Atlas', 'Realm', and 'Charts'. Under 'DEPLOYMENT', 'Databases' is selected, showing 'Data Lake'. Under 'DATA SERVICES', 'Triggers' is listed. Under 'SECURITY', 'Quickstart', 'Database Access', 'Network Access', and 'Advanced' are listed.

In the main area, there's a preview of a document:

```

phone: "+91 99999 12345"
email: "fionna@example.com"
temperature: 98.5999984741211
intime: 2022-01-04T10:08:24.105+00:00
outtime: 2022-01-04T10:08:24.105+00:00
points: 100
> address: Array

```

Below this, there's a code editor with some JSON-like code:

```

1 _id: ObjectId("61d41c988d05da6c1fa283ea")
2 name: "Mike"
3 phone: "+91 999999 33333"
4 email: "mike@example.com"
5 temperature: 98.69999604824219
6 intime: 2022-01-04T10:08:24.105+00:00
7 outtime: 2022-01-04T10:08:24.105+00:00
8 > feedback: Array
9 points: 70

```

A status bar at the bottom says 'Updating Document.'

At the bottom, another preview of a document is shown:

```

_id: ObjectId("61d41c988d05da6c1fa283eb")
name: "Anna"
phone: "+91 999999 44444"
email: "anna@example.com"
temperature: 98.19999694824219
intime: 2022-01-04T10:08:24.105+00:00
outtime: 2022-01-04T10:08:24.105+00:00

```

5.3 Update the try catch block with points as **70** and promocode as **CASHBACK200**

```

110     //Bson updateOperation = Updates.set(key, value);
111     Bson updateOperation = Updates.push(key, value);
112     collection.updateOne(filter, updateOperation);
113     System.out.println("Document Updated for email "+email);
114 } catch (Exception e) {
115     System.out.println("Something went Wrong: "+e);
116 }
117 }
118 }
119
120 public void upsertCustomer(String email, String key, String value) {
121     try {
122         Bson filter = Filters.eq("email", email);
123         Bson updateOperation = Updates.push(key, value);
124         UpdateOptions options = new UpdateOptions();
125         options.upsert(true);
126         //collection.updateOne(filter, updateOperation);
127         UpdateResult result = collection.updateOne(filter, updateOperation, options);
128         System.out.println("Result is: "+result);
129         System.out.println("Document Updated for email "+email);
130     } catch (Exception e) {
131         System.out.println("Something went Wrong: "+e);
132     }
133 }
134
135 public void upsertManyCustomers() {
136     try {
137         Bson filter = Filters.gte("points", 70);
138         Bson updateOperation = Updates.push("promoCode", "CASHBACK200");
139         UpdateResult result = collection.updateMany(filter, updateOperation);
140         System.out.println("Result is: "+result);
141     } catch (Exception e) {
142         System.out.println("Something went Wrong: "+e);
143     }
144 }
145
146 }
147

```

Writable Smart Insert 138 : 59 : 4376

5.4 Go back to App.java and use the **operation.upsertManyCustomer()** method

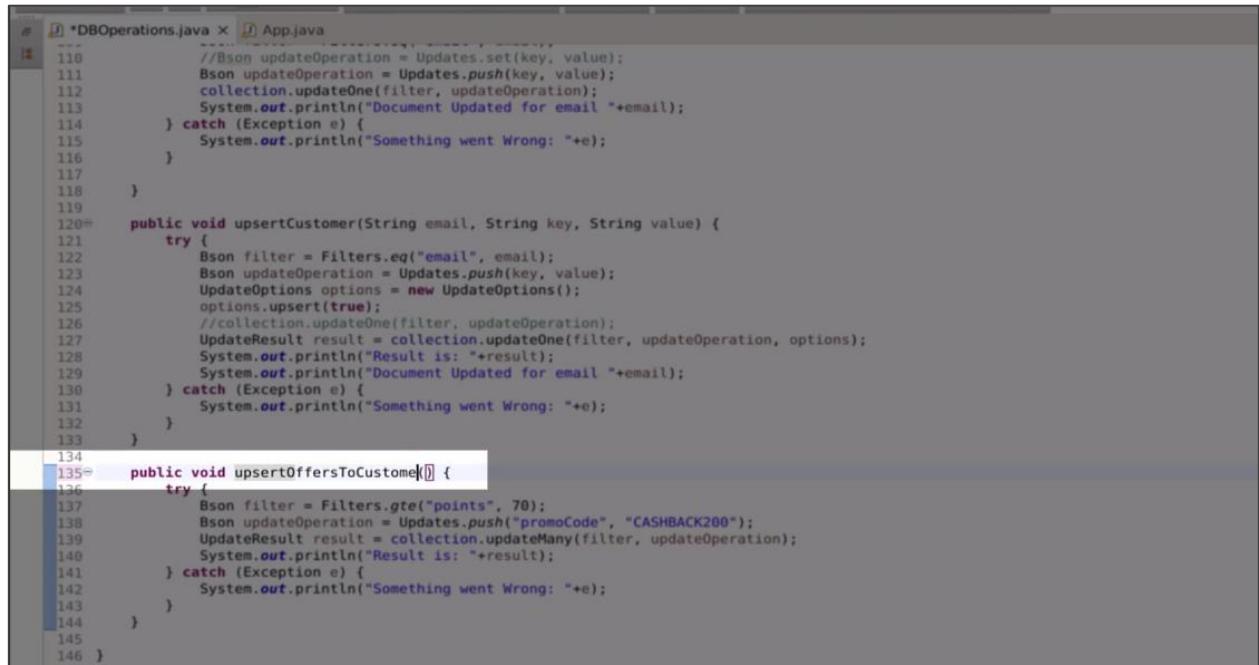
```

16 public static void main( String[] args )
17 {
18     System.out.println("MongoDB CRUD Operations App" );
19     DBOperations operations = new DBOperations();
20
21     /*Customer customer = new Customer("John Watson", "+91 999999 1111", "john@example.com", 98.4f, new Date(), new Date());
22     System.out.println("Customer Details: ");
23     System.out.println(customer);
24     operations.insertCustomer(customer);*/
25
26
27     /*List<Customer> customers = new ArrayList<Customer>();
28     customers.add(new Customer("Fionna", "+91 999999 2222", "fionna@example.com", 98.6f, new Date(), new Date()));
29     customers.add(new Customer("Mike", "+91 999999 3333", "mike@example.com", 98.7f, new Date(), new Date()));
30     customers.add(new Customer("Anna", "+91 999999 4444", "anna@example.com", 98.2f, new Date(), new Date()));
31
32     operations.insertCustomers(customers);*/
33
34     List<Customer> customers = operations.getAllCustomers();
35     /*for(Customer customer : customers) {
36         System.out.println(customer);
37     }*/
38     customers.forEach(customer ->
39     System.out.println(customer));
40 }
41
42 /*System.out.
43 System.out.
44 Customer cu
45 System.out.
46
47 //operation
48 //operation
49 //operation
50 //operation
51 operations.up
52
53 */

```

Writable Smart Insert 51 : 22 : 2123

5.5 Go back to the **DBOperations.java** file, change the name of the method to **upsertOffersToCustomers()**, and this method will display the promo codes

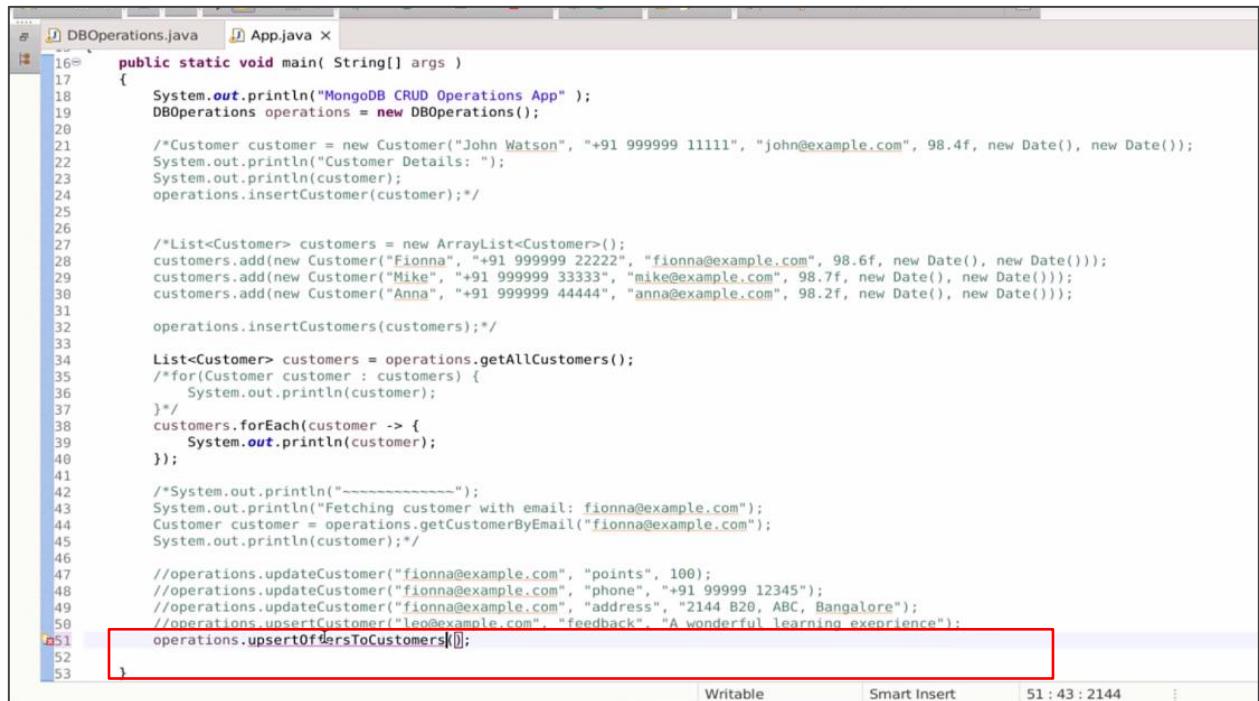


```

110         //Bson updateOperation = Updates.set(key, value);
111         Bson updateOperation = Updates.push(key, value);
112         collection.updateOne(filter, updateOperation);
113         System.out.println("Document Updated for email "+email);
114     } catch (Exception e) {
115         System.out.println("Something went Wrong: "+e);
116     }
117 }
118 }
119
120 public void upsertCustomer(String email, String key, String value) {
121     try {
122         Bson filter = Filters.eq("email", email);
123         Bson updateOperation = Updates.push(key, value);
124         UpdateOptions options = new UpdateOptions();
125         options.upsert(true);
126         //collection.updateOne(filter, updateOperation);
127         UpdateResult result = collection.updateOne(filter, updateOperation, options);
128         System.out.println("Result is: "+result);
129         System.out.println("Document Updated for email "+email);
130     } catch (Exception e) {
131         System.out.println("Something went Wrong: "+e);
132     }
133 }
134
135 public void upsertOffersToCustomer() {
136     try {
137         Bson filter = Filters.gte("points", 70);
138         Bson updateOperation = Updates.push("promoCode", "CASHBACK200");
139         UpdateResult result = collection.updateMany(filter, updateOperation);
140         System.out.println("Result is: "+result);
141     } catch (Exception e) {
142         System.out.println("Something went Wrong: "+e);
143     }
144 }
145
146 }

```

5.6 Go back to **App.java** and change the operation name to **upsertOffersToCustomers()**



```

16 public static void main( String[] args )
17 {
18     System.out.println("MongoDB CRUD Operations App" );
19     DBOperations operations = new DBOperations();
20
21     /*Customer customer = new Customer("John Watson", "+91 999999 11111", "john@example.com", 98.4f, new Date(), new Date());
22     System.out.println("Customer Details: ");
23     System.out.println(customer);
24     operations.insertCustomer(customer);*/
25
26
27     /*List<Customer> customers = new ArrayList<Customer>();
28     customers.add(new Customer("Fionna", "+91 999999 22222", "fionna@example.com", 98.6f, new Date(), new Date());
29     customers.add(new Customer("Mike", "+91 999999 33333", "mike@example.com", 98.7f, new Date(), new Date()));
30     customers.add(new Customer("Anna", "+91 999999 44444", "anna@example.com", 98.2f, new Date(), new Date()));
31
32     operations.insertCustomers(customers);*/
33
34     List<Customer> customers = operations.getAllCustomers();
35     /*for(Customer customer : customers) {
36         System.out.println(customer);
37     }*/
38     customers.forEach(customer -> {
39         System.out.println(customer);
40     });
41
42     /*System.out.println("-----");
43     System.out.println("Fetching customer with email: fionna@example.com");
44     Customer customer = operations.getCustomerByEmail("fionna@example.com");
45     System.out.println(customer);*/
46
47     //operations.updateCustomer("fionna@example.com", "points", 100);
48     //operations.updateCustomer("fionna@example.com", "phone", "+91 99999 12345");
49     //operations.updateCustomer("fionna@example.com", "address", "2144 B20, ABC, Bangalore");
50     //operations.upsertCustomer("Leo@example.com", "feedback", "A wonderful learning experience");
51     operations.upsertOffersToCustomer();
52
53 }

```

5.7 Save the file and run the code

```

16-    public static void main( String[] args )
17-    {
18-        System.out.println("MongoDB CRUD Operations App");
19-        DBOperations operations = new DBOperations();
20-
21-        /*Customer customer = new Customer("John Watson", "+91 999999 1111", "john@example.com", 98.4f, new Date(), new Date());
22-        System.out.println("Customer Details: ");
23-        System.out.println(customer);
24-        operations.insertCustomer(customer);*/
25-
26-
27-        /*List<Customer> customers = new ArrayList<Customer>();
28-        customers.add(new Customer("Fionna", "+91 999999 2222", "fionna@example.com", 98.6f, new Date(), new Date()));
29-        customers.add(new Customer("Mike", "+91 999999 3333", "mike@example.com", 98.7f, new Date(), new Date()));
30-        customers.add(new Customer("Anna", "+91 999999 4444", "anna@example.com", 98.2f, new Date(), new Date()));
31-
32-        operations.insertCustomers(customers);*/
33-
34-        List<Customer> customers = operations.getAllCustomers();
35-        /*forCustomer customer : customers{
36-            System.out.println(customer);
37-        }*/
38-        customers.forEach(customer -> {
39-            System.out.println(customer);
40-        });
41-
42-        /*System.out.println("-----");
43-        System.out.println("Fetching customer with email: fionna@example.com");
44-        Customer customer = operations.getCustomerByEmail("fionna@example.com");
45-        System.out.println(customer);*/
46-
47-        //operations.updateCustomer("fionna@example.com", "points", 100);
48-        //operations.updateCustomer("fionna@example.com", "phone", "+91 999999 12345");
49-        //operations.updateCustomer("fionna@example.com", "address", "2144 B2B, ABC, Bangalore");
50-        //operations.upsertCustomer("leo@example.com", "feedback", "A wonderful learning experience");
51-        operations.upsertOffersToCustomers();
52-
53-    }

```

The screenshot shows the Eclipse IDE interface with the code editor and a terminal window. The terminal output is as follows:

```

<terminated> App (3) [Java Application] /usr/eclipse/plugins/org.eclipse.justj.openjdk.hotspot/jdk11/bin/java -Dfile.encoding=UTF-8 com.mongodb.diagnostics.logging.Logger$SLF4J
MongoDB CRUD Operations App
Jan 05, 2022 10:45:12 AM com.mongodb.diagnostics.logging.Logger$SLF4J
WARNING: SLF4J not found on the classpath. Logging is disabled for the 'org.mongodb.*' package.
DBOperations] Connection Created
DBOperations] Database Selected as eStore
DBOperations] Collection 'customers' from eStore selected as customers
Customer [name=Fionna, phone=+91 999999 2222, email=fionna@example.com, tempId=1]
Customer [name=Anna, phone=+91 999999 4444, email=anna@example.com, tempId=2]
Customer [name=John, phone=+91 999999 1111, email=john@example.com, tempId=3]
Customer [name=Mike, phone=+91 999999 3333, email=mike@example.com, tempId=4]
Result is: AcknowledgedUpdateResult{matchedCount=3, modifiedCount=3, upsertedCount=0}

```

You can see that three documents were matched, and three documents were modified.

5.8 Go back to the database and refresh the screen

The screenshot shows the MongoDB Cloud interface with the URL `cloud.mongodb.com/v2/61d2af87f07b4a321e698a73#metrics/replicaSet/61d2b0ec424e2b0a95e3d02e/explorer/estore/customers/find`. The left sidebar shows 'Project 0' and 'Atlas'. The main area displays two customer documents:

```

{
  "_id": ObjectId("61d41c988d05da6c1fa283ea"),
  "name": "Fionna",
  "phone": "+91 99999 12345",
  "email": "fionna@example.com",
  "temperature": 98.5999984741211,
  "intime": 2022-01-04T10:08:24.105+00:00,
  "outtime": 2022-01-04T10:08:24.105+00:00,
  "points": 100,
  "address": {
    "city": "Mumbai",
    "state": "Maharashtra"
  },
  "feedback": [
    {
      "rating": 5,
      "comment": "Great service!"
    }
  ],
  "promoCode": "CASHBACK200"
}

{
  "_id": ObjectId("61d41c988d05da6c1fa283ea"),
  "name": "Mike",
  "phone": "+91 99999 33333",
  "email": "mike@example.com",
  "temperature": 98.69999694824219,
  "intime": 2022-01-04T10:08:24.105+00:00,
  "outtime": 2022-01-04T10:08:24.105+00:00,
  "points": 70
}

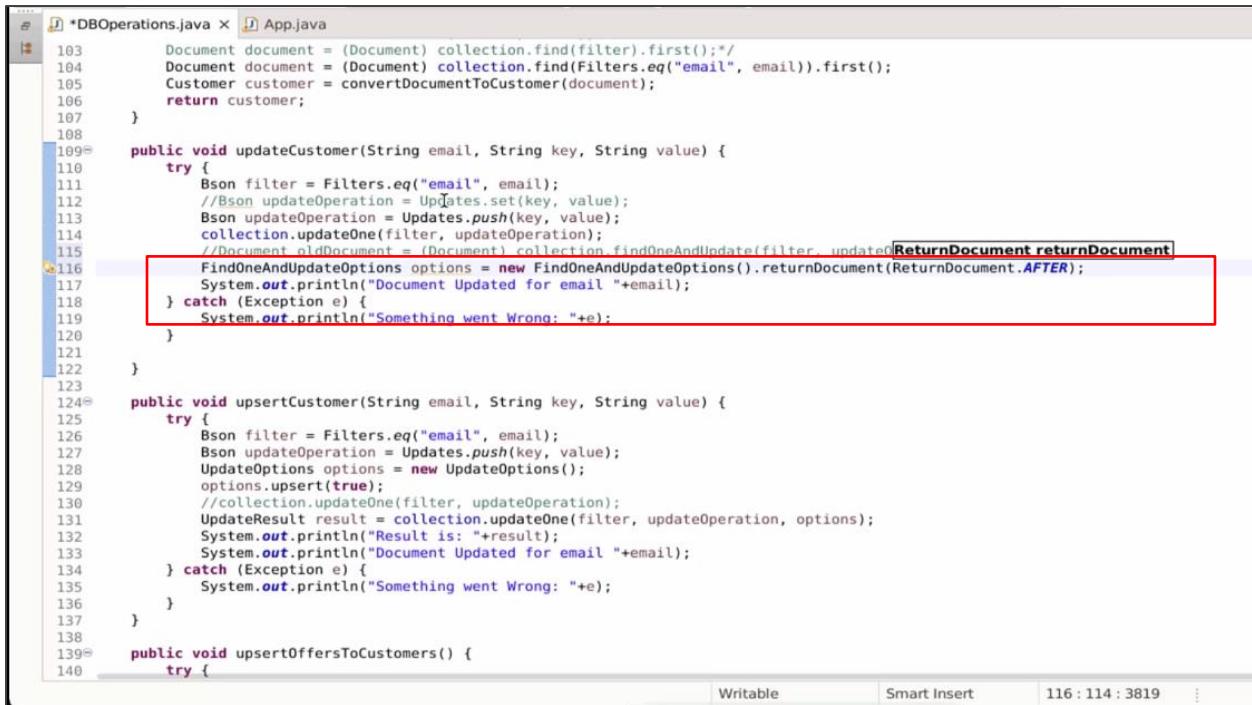
```

You can see a promo code for points **100** as the condition was greater than or equal to **70**.

The screenshot shows the MongoDB Cloud interface with the same URL as before. The main area displays the two customer documents. The second document now has a 'promoCode' field with the value 'CASHBACK200', which is highlighted with a red box.

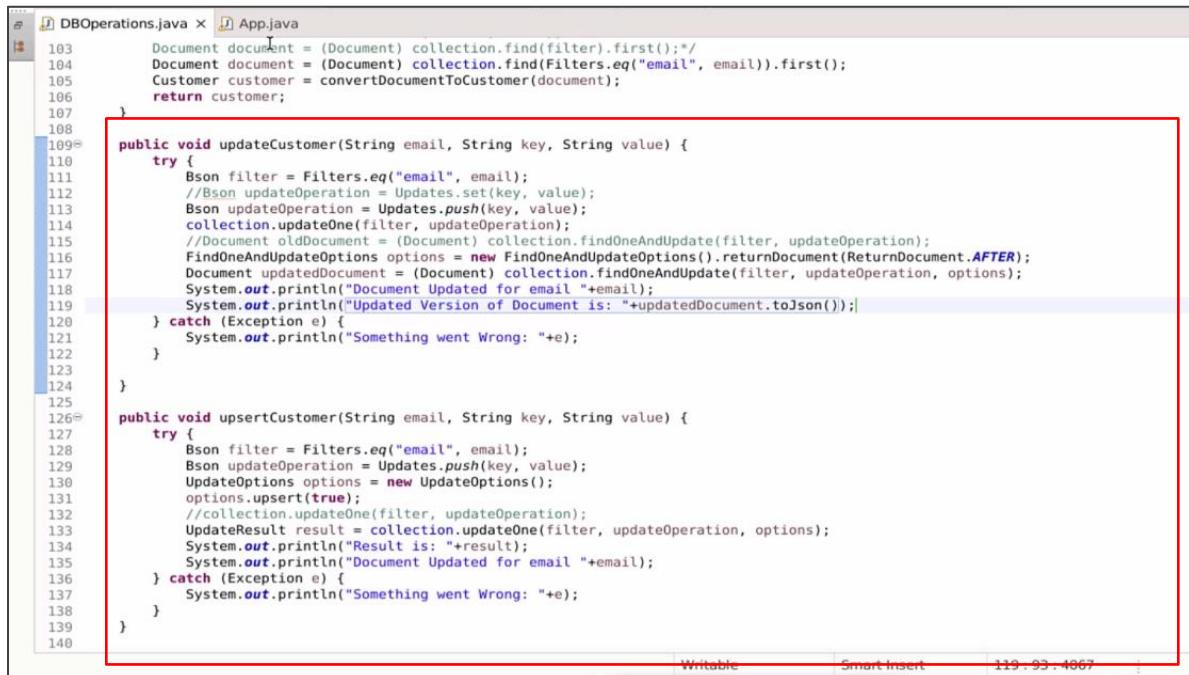
Step 6: Write a new method

6.1 Go back to the **DBOperations.java**, create a new collection, and use the **findOneAndUpdate** method, which will take the update operation as input to update the document



```
103     Document document = (Document) collection.find(filter).first();*/
104     Document document = (Document) collection.find(Filters.eq("email", email)).first();
105     Customer customer = convertDocumentToCustomer(document);
106     return customer;
107 }
108
109 public void updateCustomer(String email, String key, String value) {
110     try {
111         Bson filter = Filters.eq("email", email);
112         //Bson updateOperation = Updates.set(key, value);
113         Bson updateOperation = Updates.push(key, value);
114         collection.updateOne(filter, updateOperation);
115         //Document oldDocument = (Document) collection.findOneAndUpdate(filter, updateOperation, ReturnDocument.AFTER);
116         FindOneAndUpdateOptions options = new FindOneAndUpdateOptions().returnDocument(ReturnDocument.AFTER);
117         System.out.println("Document Updated for email "+email);
118     } catch (Exception e) {
119         System.out.println("Something went Wrong: "+e);
120     }
121 }
122
123
124 public void upsertCustomer(String email, String key, String value) {
125     try {
126         Bson filter = Filters.eq("email", email);
127         Bson updateOperation = Updates.push(key, value);
128         UpdateOptions options = new UpdateOptions();
129         options.upsert(true);
130         //collection.updateOne(filter, updateOperation);
131         UpdateResult result = collection.updateOne(filter, updateOperation, options);
132         System.out.println("Result is: "+result);
133         System.out.println("Document Updated for email "+email);
134     } catch (Exception e) {
135         System.out.println("Something went Wrong: "+e);
136     }
137 }
138
139 public void upsertOffersToCustomers() {
140     try {
```

The final code appears as shown below:

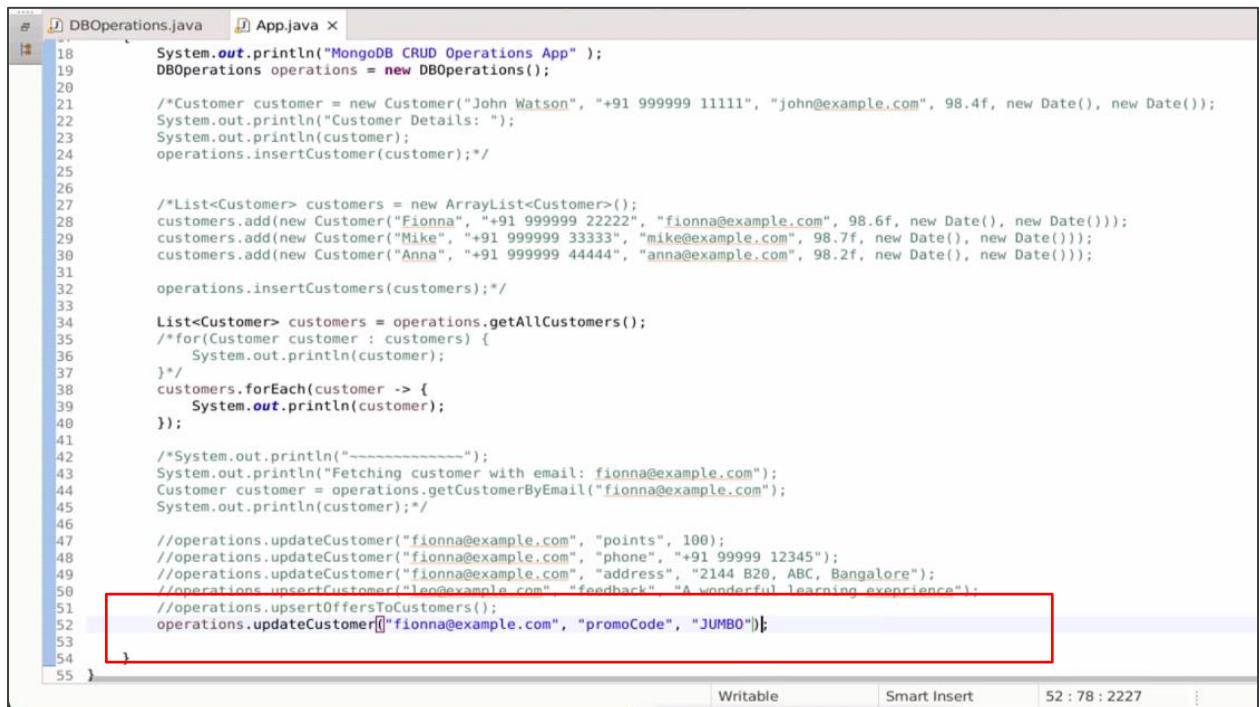


```

103     Document document = (Document) collection.find(filter).first();*/
104     Document document = (Document) collection.find(Filters.eq("email", email)).first();
105     Customer customer = convertDocumentToCustomer(document);
106     return customer;
107 }
108
109 public void updateCustomer(String email, String key, String value) {
110     try {
111         Bson filter = Filters.eq("email", email);
112         //Bson updateOperation = Updates.set(key, value);
113         Bson updateOperation = Updates.push(key, value);
114         collection.updateOne(filter, updateOperation);
115         //Document oldDocument = (Document) collection.findOneAndUpdate(filter, updateOperation);
116         FindOneAndUpdateOptions options = new FindOneAndUpdateOptions().returnDocument(ReturnDocument.AFTER);
117         Document updatedDocument = (Document) collection.findOneAndUpdate(filter, updateOperation, options);
118         System.out.println("Document Updated for email "+email);
119         System.out.println("Updated Version of Document is: "+updatedDocument.toJson());
120     } catch (Exception e) {
121         System.out.println("Something went Wrong: "+e);
122     }
123 }
124
125 public void upsertCustomer(String email, String key, String value) {
126     try {
127         Bson filter = Filters.eq("email", email);
128         Bson updateOperation = Updates.push(key, value);
129         UpdateOptions options = new UpdateOptions();
130         options.upsert(true);
131         //collection.updateOne(filter, updateOperation);
132         UpdateResult result = collection.updateOne(filter, updateOperation, options);
133         System.out.println("Result is: "+result);
134         System.out.println("Document Updated for email "+email);
135     } catch (Exception e) {
136         System.out.println("Something went Wrong: "+e);
137     }
138 }
139
140

```

6.2 Go back to the **App.java** file and edit the **promoCode** as **JUMBO** in the **updateCustomer** method

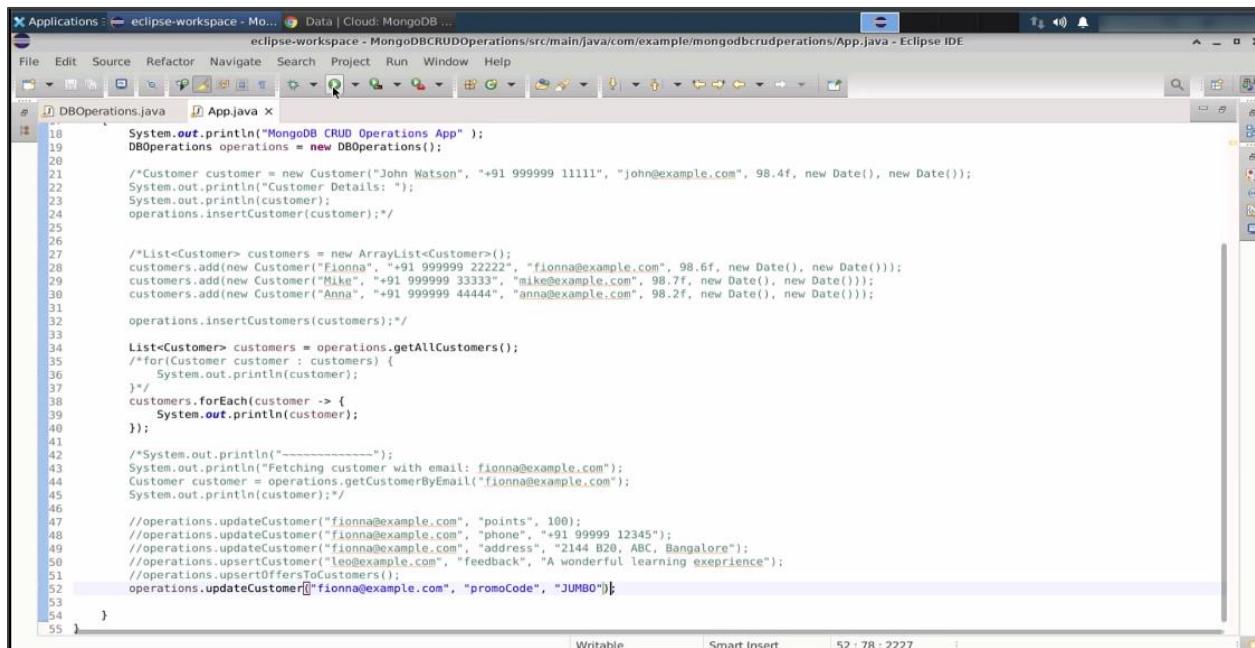


```

18     System.out.println("MongoDB CRUD Operations App");
19     DBOperations operations = new DBOperations();
20
21     /*Customer customer = new Customer("John Watson", "+91 999999 1111", "john@example.com", 98.4f, new Date(), new Date());
22     System.out.println("Customer Details: ");
23     System.out.println(customer);
24     operations.insertCustomer(customer);*/
25
26
27     /*List<Customer> customers = new ArrayList<Customer>();
28     customers.add(new Customer("Fionna", "+91 999999 2222", "fionna@example.com", 98.6f, new Date(), new Date());
29     customers.add(new Customer("Mike", "+91 999999 3333", "mike@example.com", 98.7f, new Date(), new Date()));
30     customers.add(new Customer("Anna", "+91 999999 4444", "anna@example.com", 98.2f, new Date(), new Date()));
31
32     operations.insertCustomers(customers);*/
33
34     List<Customer> customers = operations.getAllCustomers();
35     /*for(Customer customer : customers) {
36         System.out.println(customer);
37     }*/
38     customers.forEach(customer -> {
39         System.out.println(customer);
40     });
41
42     /*System.out.println("-----");
43     System.out.println("Fetching customer with email: fionna@example.com");
44     Customer customer = operations.getCustomerByEmail("fionna@example.com");
45     System.out.println(customer);*/
46
47     //operations.updateCustomer("fionna@example.com", "points", 100);
48     //operations.updateCustomer("fionna@example.com", "phone", "+91 999999 12345");
49     //operations.updateCustomer("fionna@example.com", "address", "2144 B20, ABC, Bangalore");
50     //operations.upsertCustomer("leo@example.com", "feedback", "A wonderful learning experience");
51     //operations.upsertOffersToCustomers();
52     operations.updateCustomer("fionna@example.com", "promoCode", "JUMBO");
53
54 }
55

```

6.3 Save the file and run the code



The screenshot shows the Eclipse IDE interface with the title bar "eclipse-workspace - Mo... Data | Cloud: MongoDB ...". The menu bar includes File, Edit, Source, Refactor, Navigate, Search, Project, Run, Window, Help. The toolbar has various icons for file operations. The main editor window displays Java code for MongoDB CRUD operations:

```
1 Applications : eclipse-workspace - Mo... Data | Cloud: MongoDB ...
2 eclipse-workspace - MongoDBCRUDOperations/src/main/java/com/example/mongodbcrudoperations/App.java - Eclipse IDE
3
4 File Edit Source Refactor Navigate Search Project Run Window Help
5
6 DBOperations.java App.java X
7
8 18 System.out.println("MongoDB CRUD Operations App");
9 19 DBOperations operations = new DBOperations();
10 20
11 21 /*Customer customer = new Customer("John Watson", "+91 999999 1111", "john@example.com", 98.4f, new Date(), new Date());
12 22 System.out.println("Customer Details: ");
13 23 System.out.println(customer);
14 24 operations.insertCustomer(customer);*/
15 25
16 26
17 27 /*List<Customer> customers = new ArrayList<Customer>();
18 28 customers.add(new Customer("Fionna", "+91 999999 2222", "fionna@example.com", 98.6f, new Date(), new Date()));
19 29 customers.add(new Customer("Mike", "+91 999999 3333", "mike@example.com", 98.7f, new Date(), new Date()));
20 30 customers.add(new Customer("Anna", "+91 999999 4444", "anna@example.com", 98.2f, new Date(), new Date()));
21 31
22 32 operations.insertCustomers(customers);*/
23 33
24 34 List<Customer> customers = operations.getAllCustomers();
25 35 /*for(Customer customer : customers) {
26 36     System.out.println(customer);
27 37 }
28 38 customers.forEach(customer -> {
29 39     System.out.println(customer);
30 40 });
31 41 */
32 42 /*System.out.println("-----");
33 43 System.out.println("Fetching customer with email: fionna@example.com");
34 44 Customer customer = operations.getCustomerByEmail("fionna@example.com");
35 45 System.out.println(customer);*/
36 46
37 47 //operations.updateCustomer("fionna@example.com", "points", 100);
38 48 //operations.updateCustomer("fionna@example.com", "phone", "+91 99999 12345");
39 49 //operations.updateCustomer("fionna@example.com", "address", "2144 B20, ABC, Bangalore");
40 50 //operations.updateCustomer("leo@example.com", "feedback", "A wonderful learning experience");
41 51 //operations.insertOffersToCustomers();
42 52 operations.updateCustomer("fionna@example.com", "promoCode", "JUMBO");
43 53
44 54 }
45 55 }
```

The code implements MongoDB CRUD operations for a Customer entity. It includes methods for inserting, updating, and fetching customers by email. The code is annotated with line numbers from 1 to 55.

You can see the output as the **Updated Version of Document** is along with other details.

The screenshot shows the Eclipse IDE interface with the following details:

- Title Bar:** Practice Labs - DEFAULT
- Toolbar:** File Edit Source Refactor Navigate Search Project Run Window Help
- Left Sidebar:** Applications - eclipse-workspace - Mo... Data | Cloud MongoDb ...
- Code Editor:** DBOperations.java (selected) and App.java
- Output Console:**
 - Terminated: App (3) [Java Application] /usr/eclipse/plugins/org.eclipse.justj.openjdk.hotspot/jre/bin/java -Dfile.encoding=UTF-8 com.mongodb.diagnostics.logging.Logners should use WARNING: SLF4J not found on the classpath. Logging is disabled for the 'com.mongodb' package.
 - [DBOperations] Connection Created
 - [DBOperations] Database Selected as eStore
 - [DBOperations] Collection from eStore selected as customers
 - Customer [name=Anna, phone=>91 999999 4444, email=anna@example.com, tempId=61d41c988d05da6c1fa283e9]
 - Customer [name=Fionna, phone=>91 999999 12345, email=fionna@example.com, tempId=61d41c988d05da6c1fa283e9]
 - Customer [name=John Watson, phone=>91 999999 11111, email=john@example.com, tempId=61d41c988d05da6c1fa283e9]
 - Customer [name=Mike, phone=>91 999999 33333, email=mike@example.com, tempId=61d41c988d05da6c1fa283e9]
- Right Side:** Problems, Javadoc, Declaration, Console

This is all about the update operations. You can perform an update on a single document using the **UpdateOne** function and an **upsert** operation which is a combination of update and insert. The **updateMany** operation can be used with some filters or conditions.

By following these steps, you have successfully worked with the CRUD operations and updated the documents for existing and non-existing customers.