TECHNOLOGY

Caltech | Center for Technology & Management Education

Develop Java Backend for End User Web App

Powerd by simplilearn

TECHNOLOGY

Caltech | Center for Technology & Management Education

DAO Design Pattern

Powerd by simplilearn

You Already Know

Before we begin, let's recall what we have covered till now:

MongoDB

Recap

Maven Project for Backend

- Created a Maven project with archetype as web app in Eclipse EE

Developed POJO Classes

- Created various classes for the Admin and End UserProjects
- Developed POJO with constructors, getters, setters and toString

Project Configuration

- Configured MySQL, Servlet, JSP and Apache Tomcat Web Server

Build and Execute

- Built and executed the Maven Web App Project
- Packaged the Web Project as war file

A Day in the Life of a Full Stack Developer

As a full stack web developer, our key role is to develop both client and server software.

- Angular and Node can be used to build front end of the web page.

- Spring Boot, Java, and MySQL/MongoDB can be used to build at the back end.

A Day in the Life of a Full Stack Developer

Now, bob needs to develop the design pattern DAO in a generic manner. So, bob brainstorms a bit and finds a solution.

Let me use java, OOPS and JDBC to develop the design pattern DAO in a generic manner.

In this lesson, we will learn the java, OOPS, and JDBC skills for created Maven project and develop the design pattern DAO in a generic manner. Further, we will also implement CRUD operations with JDBC for various models and help bob to complete his task effectively and quickly.

Learning Objectives

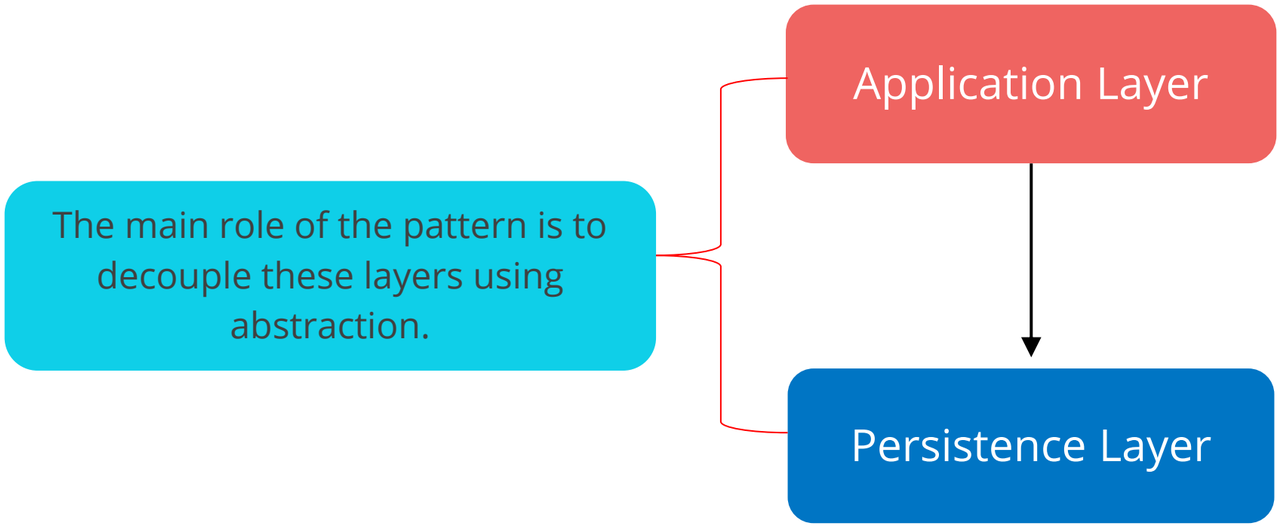
By the end of this lesson, you will be able to:

- Implement DAO Design Pattern in a Generic Manner
- Connect with MySQL using JDBC
- Create CRUD Operations for the Admin Models
- DeCouple the Model from Persistence Layer

Develop Generic DAO Design Pattern for the End User Web App

Data Access Object

The Data Access Object (DAO) is a structural design pattern.



The DAO APIs will hide all the complexity for CRUD operations, and hence, both layers work in isolation.

Create DAO Interface: DAO.java

DAO was created under the packeage com.example.estore.dao
Using the generic concept of Java i.e. <T> for Type.

```
package com.example.estore.dao;
import java.util.List;

// Generic Interface for CRUD Operations
public interface DAO<T> {

        T get(long id);

        List<T> getAll();

        void save(T object);

        void update(T object);

        void delete(long id);

}
```

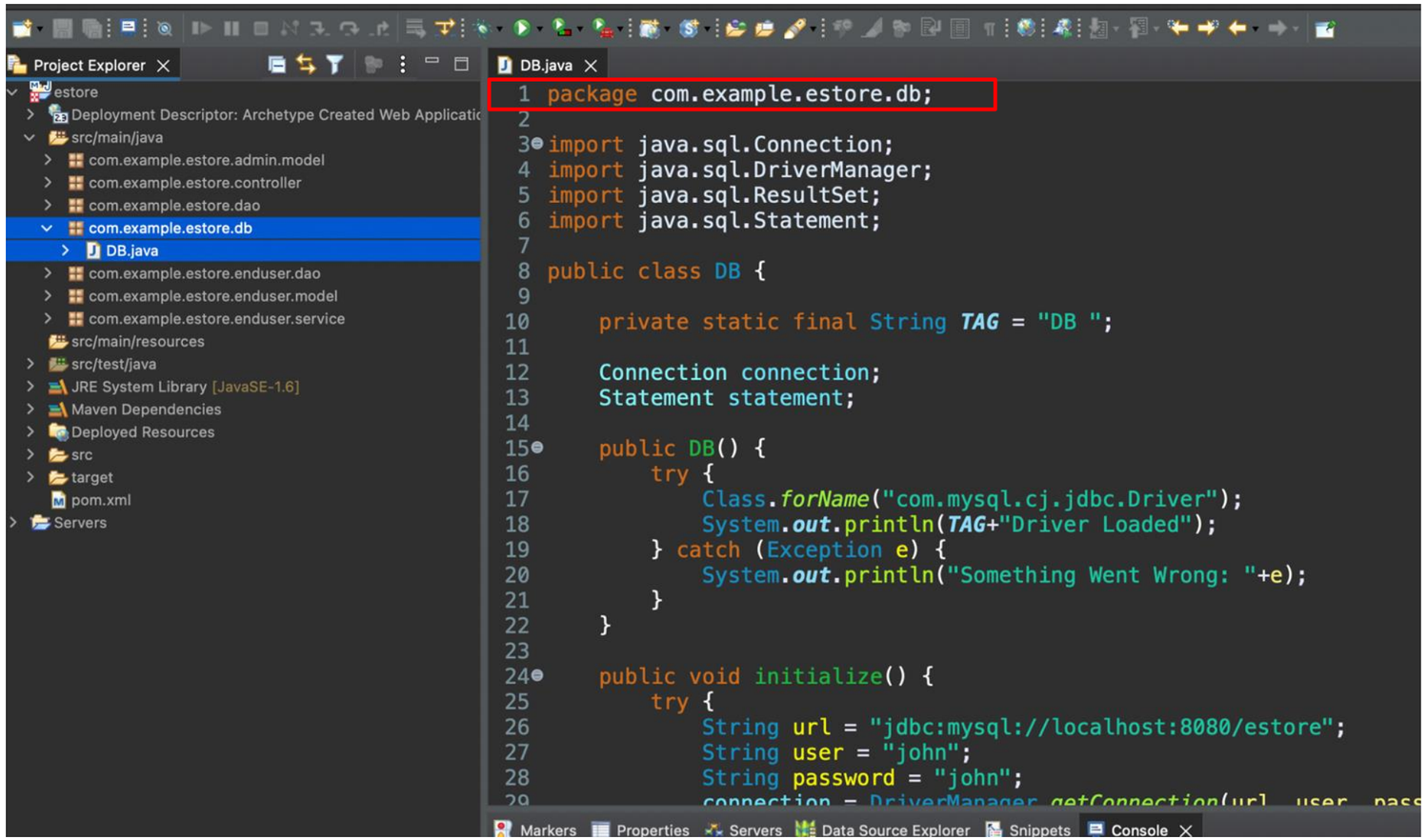
Create DAO Interface: DAO.java

Review of methods created inside the packeage com.example.estore.dao

Method	Use
get method	Returns the object based on id as input
getAll method	Returns list of all the objects and can serve as cache for various use cases
savemethod	Saves the object passed as input
update method	Updates the object passed as input
delete method	Deletes the object based on id as input

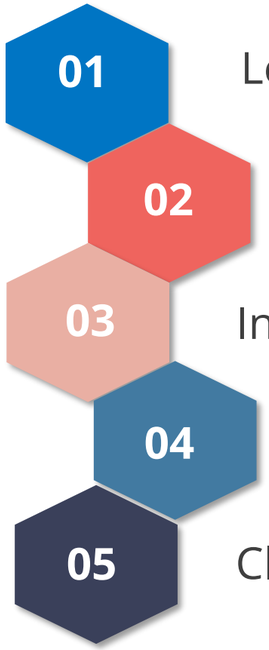
DB.java

Review of previously created class DB in the admin module



DB.java

Few implementation that were included are as follows:



Load the Driver

Create Connection to DB

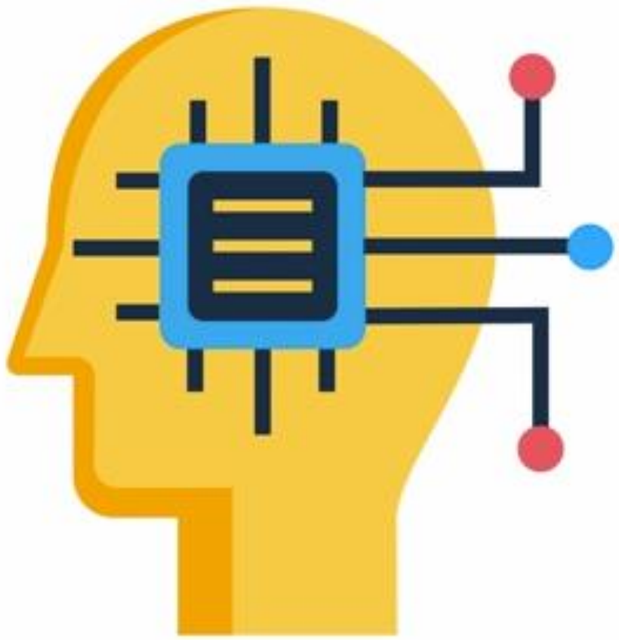
Initialize Statement

Execute SQL for update and query operations

Close the connection to release memory resources

DB.java: With Singleton Design Pattern

Singleton Design Pattern is used for better memory management.



DB.java: With Singleton Design Pattern

Reference Code for DB.java after singleton:

DB.java

```java

public class DB {

    private static final String TAG = "DB ";
    private static DB db = new DB();


    public static DB getDB() {
        return db;
    }


    Connection connection;
    Statement statement;

    private DB() {
        try {
            Class.forName("com.mysql.cj.jdbc.Driver");
            System.out.println(TAG+"Driver Loaded");
            initialize();
        } catch (Exception e) {
            System.out.println("Something Went Wrong: "+e);
        }
    }

    public void initialize() {
        try {
            String url = "jdbc:mysql://localhost:8080/estore";
            String user = "john";
            String password = "john";
            connection = DriverManager.getConnection(url, user, password);
            System.out.println(TAG+"Connection Created");
            statement = connection.createStatement();
            System.out.println(TAG+"Statement Created");
        } catch (Exception e) {
            System.out.println("Something Went Wrong: "+e);
        }
    }
```

TECHNOLOGY

Caltech | Center for Technology & Management Education

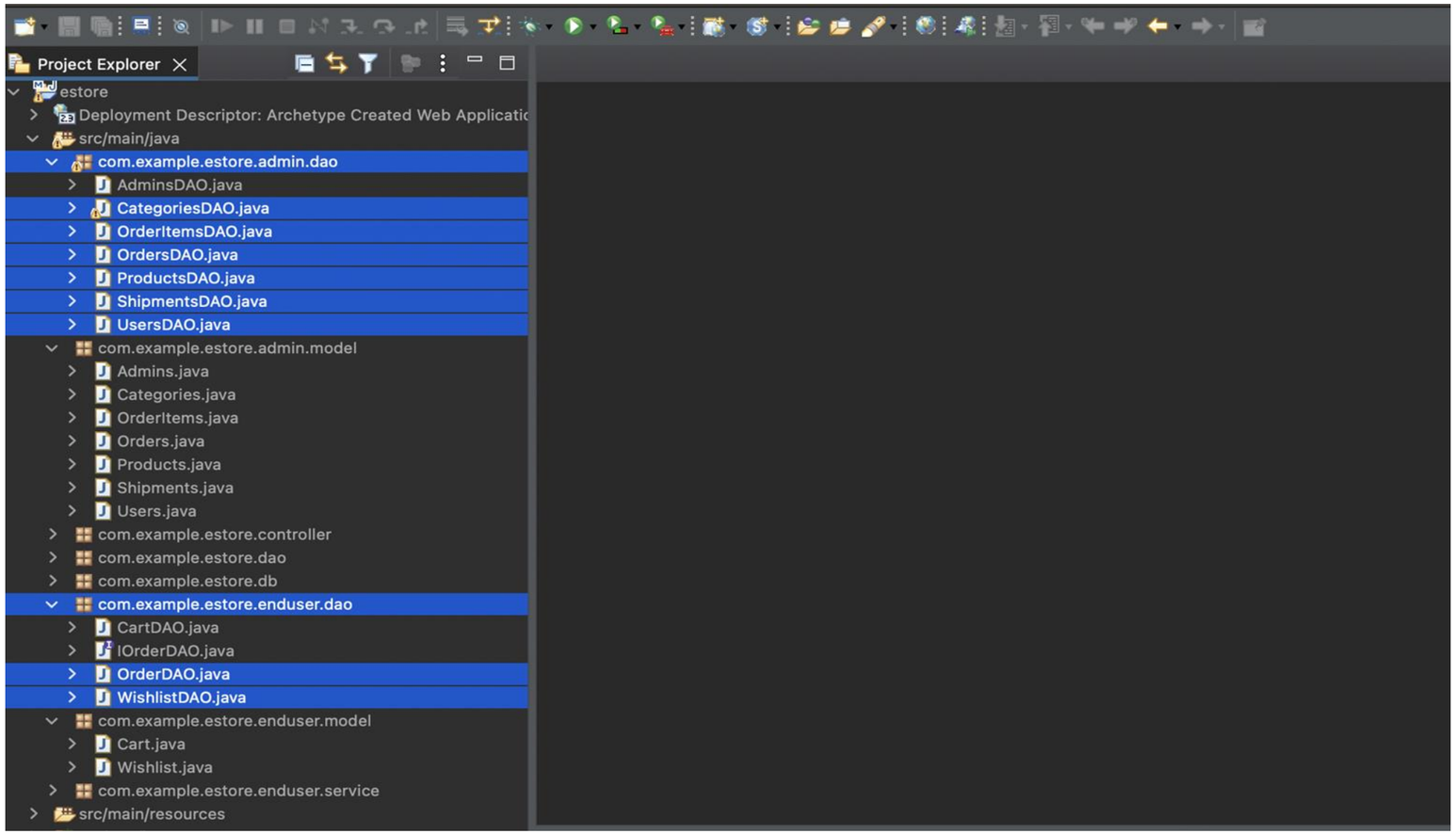
Implement CRUD Operations

Powerd by simplilearn

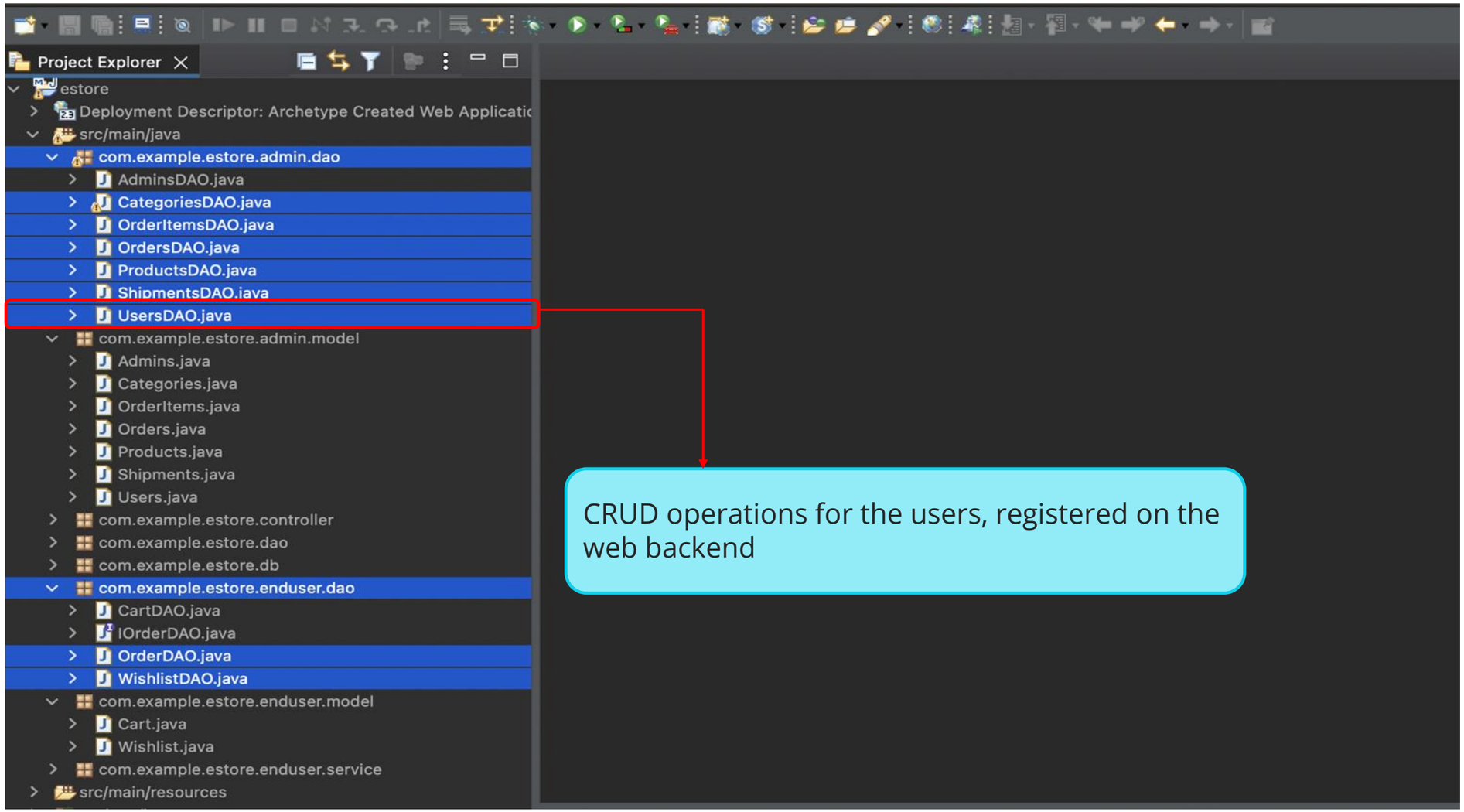
CRUD Operations

Create Classes Implementing DAO for the End User Web App

Create various classes implementing DAO interface to perform CRUD Operations for the End User Web App.

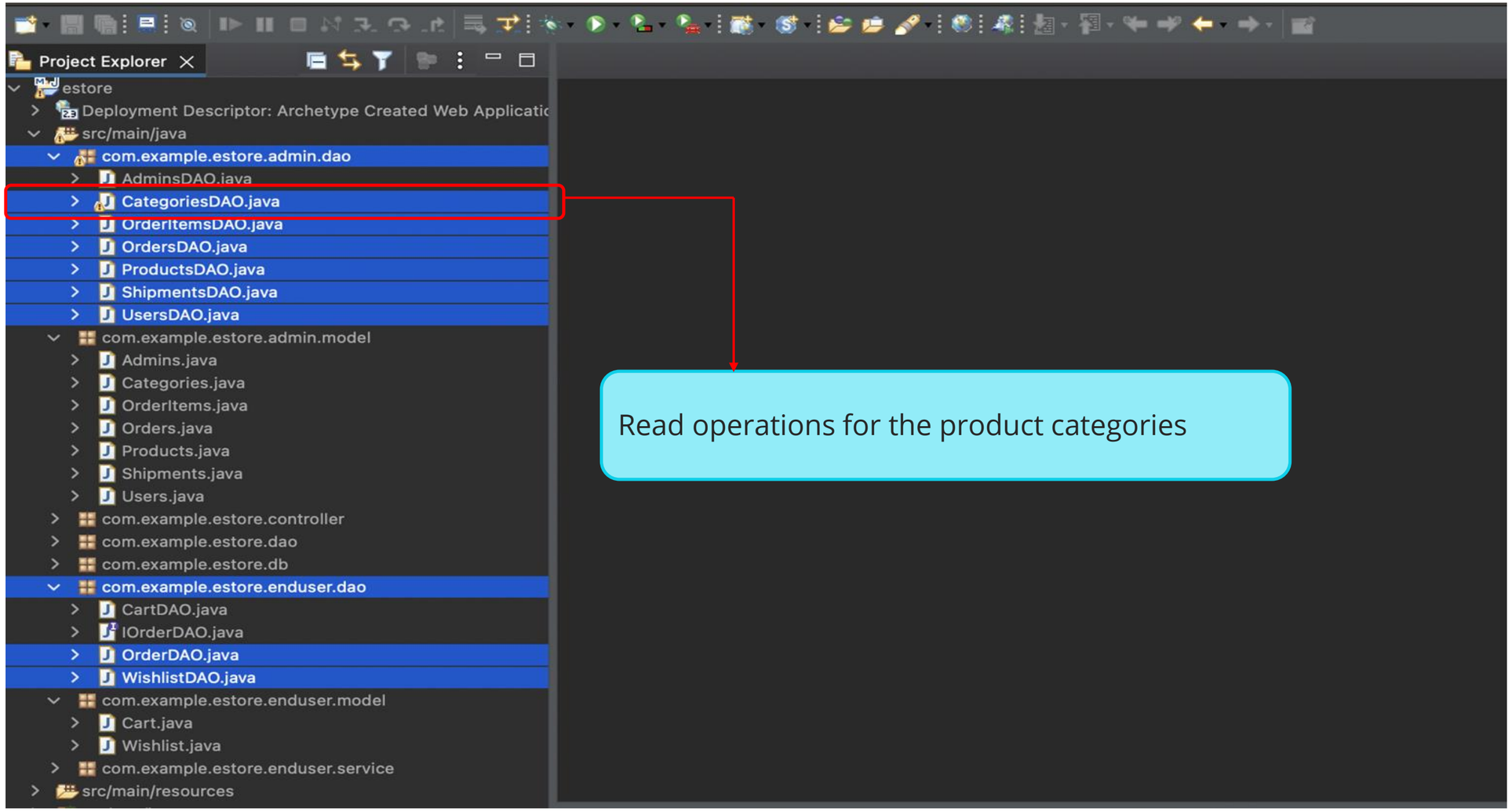


Create Classes Implementing DAO for the End User Web App



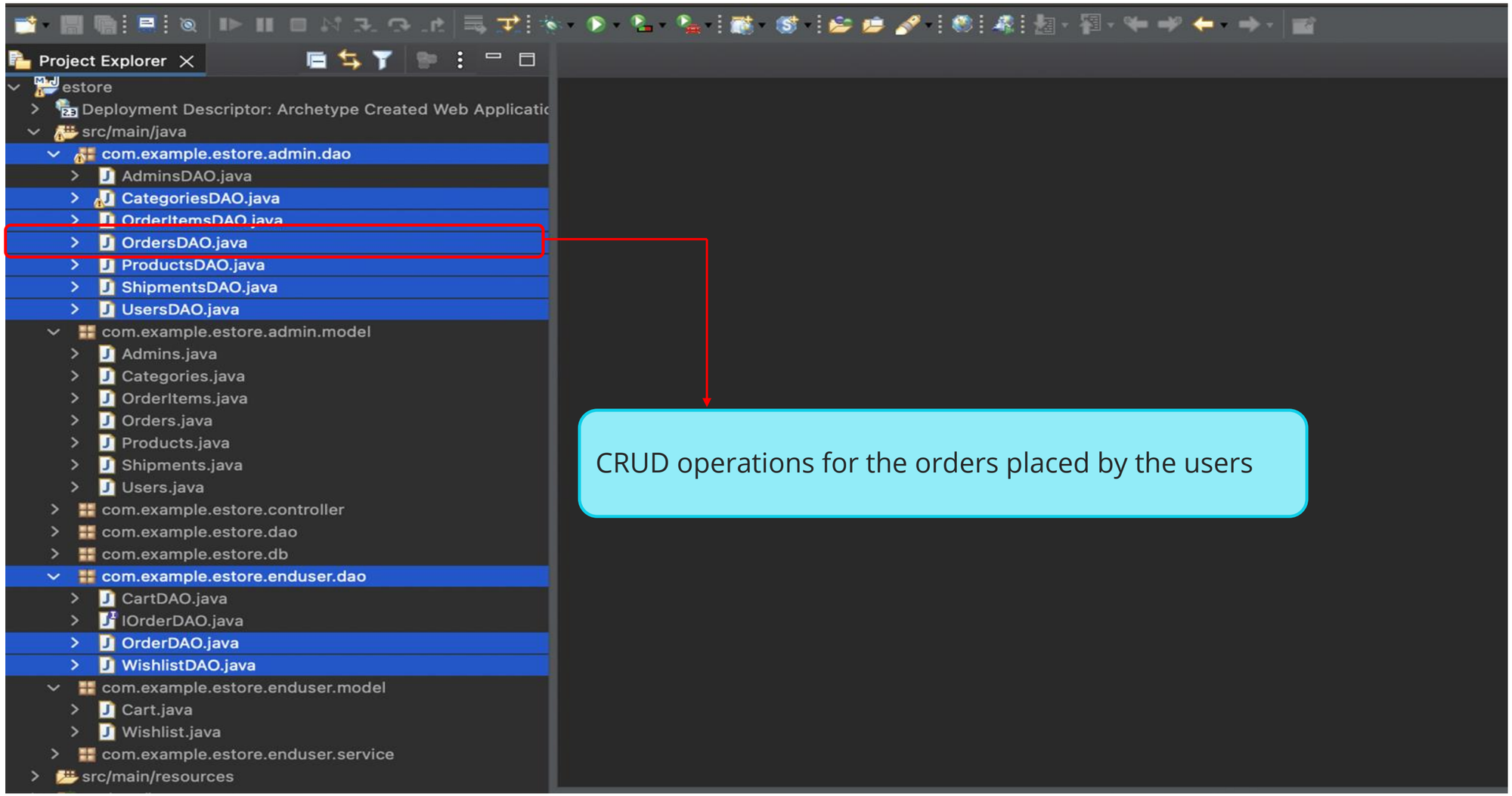
CRUD operations for the users, registered on the web backend

Create Classes Implementing DAO for the End User Web App



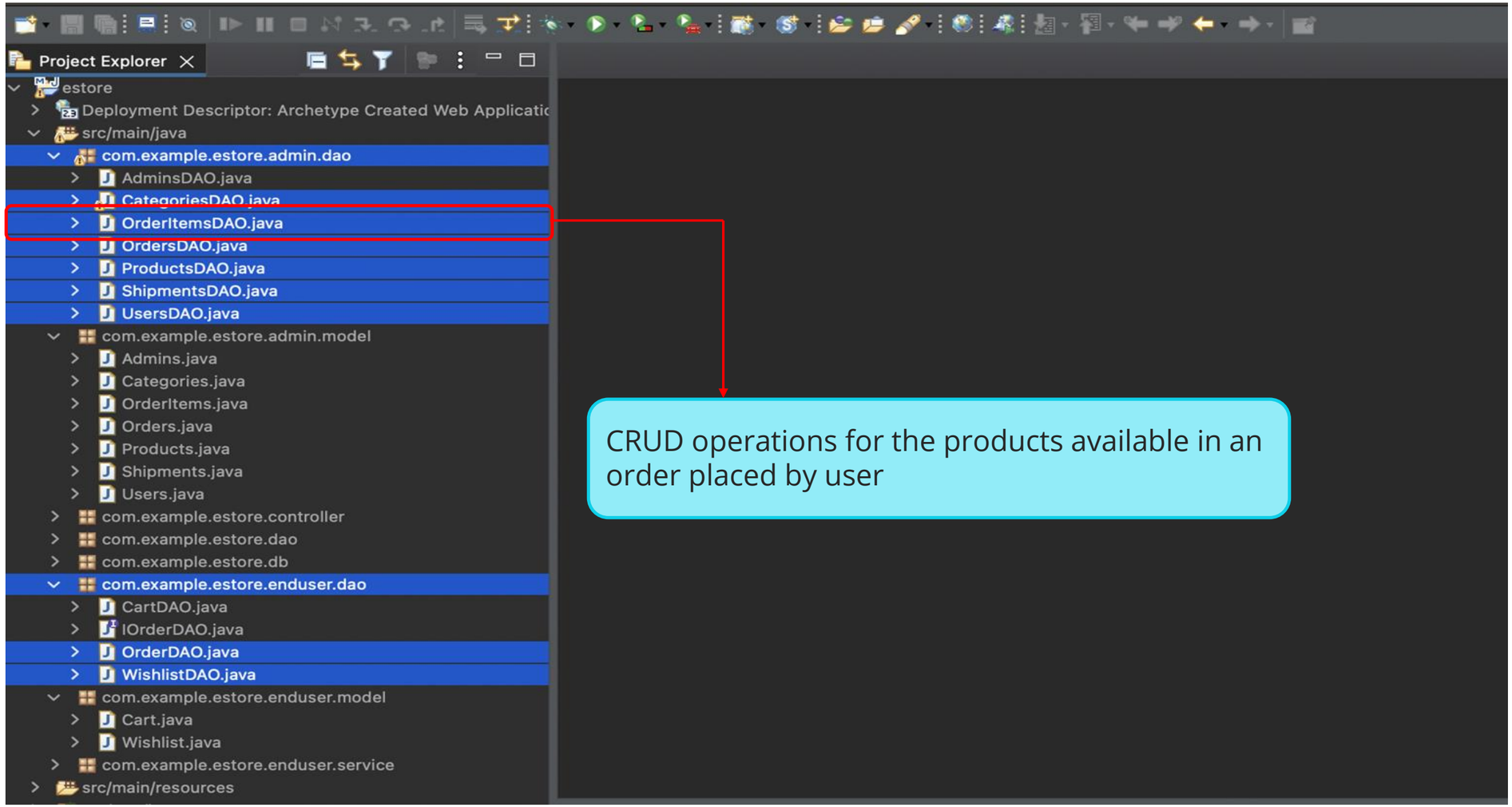
Read operations for the product categories

Create Classes Implementing DAO for the End User Web App



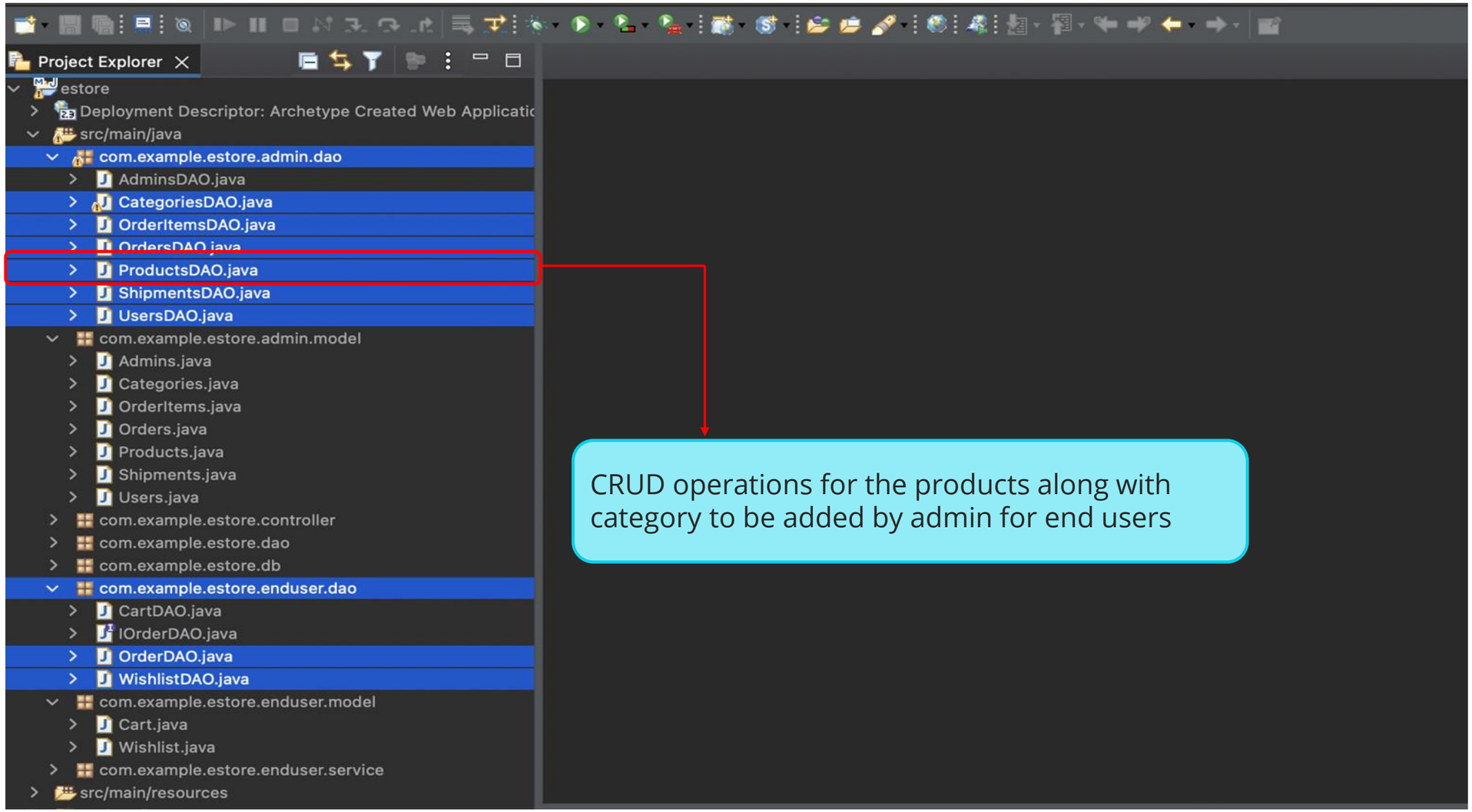
CRUD operations for the orders placed by the users

Create Classes Implementing DAO for the End User Web App



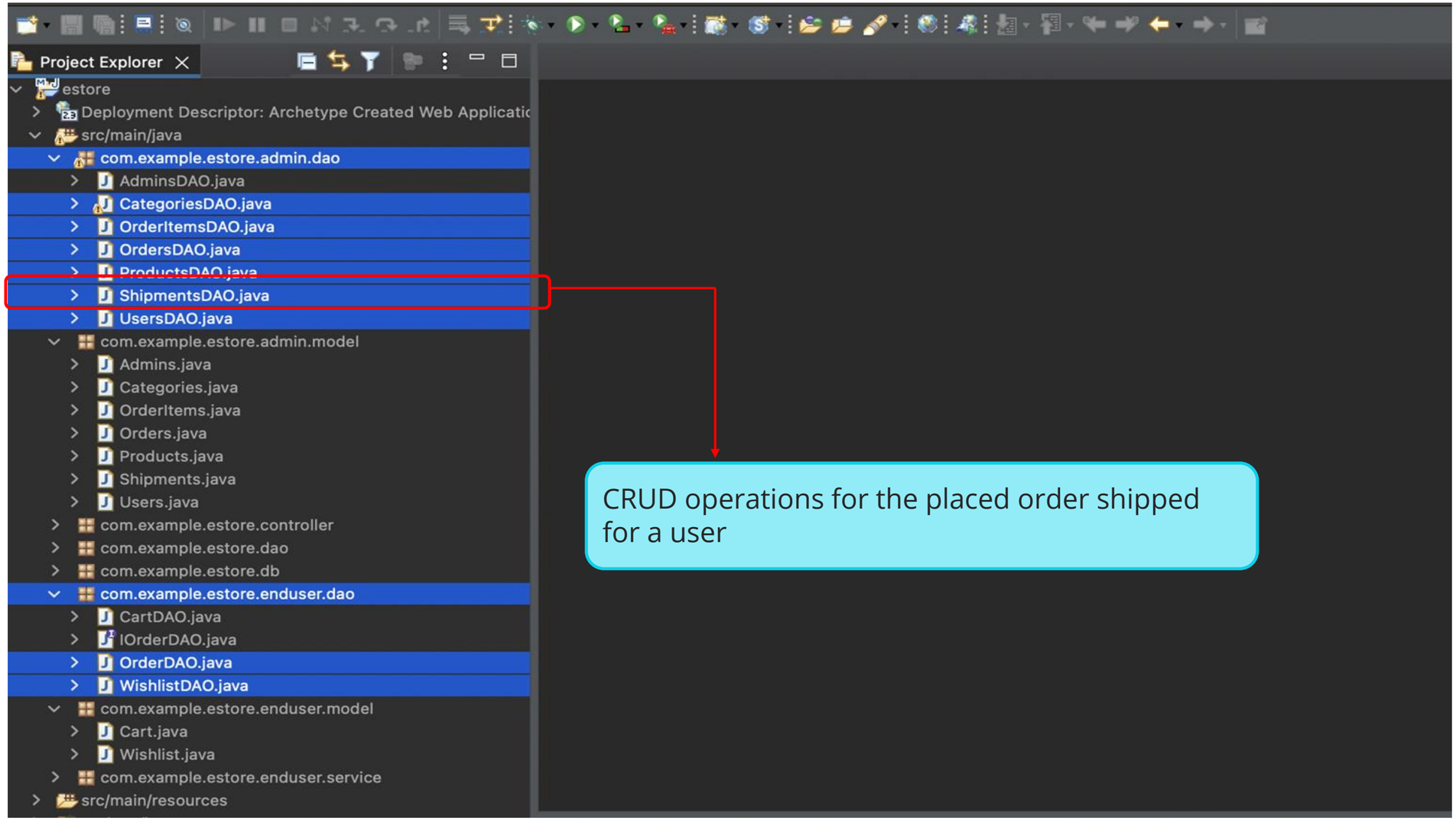
CRUD operations for the products available in an order placed by user

Create Classes Implementing DAO for the End User Web App



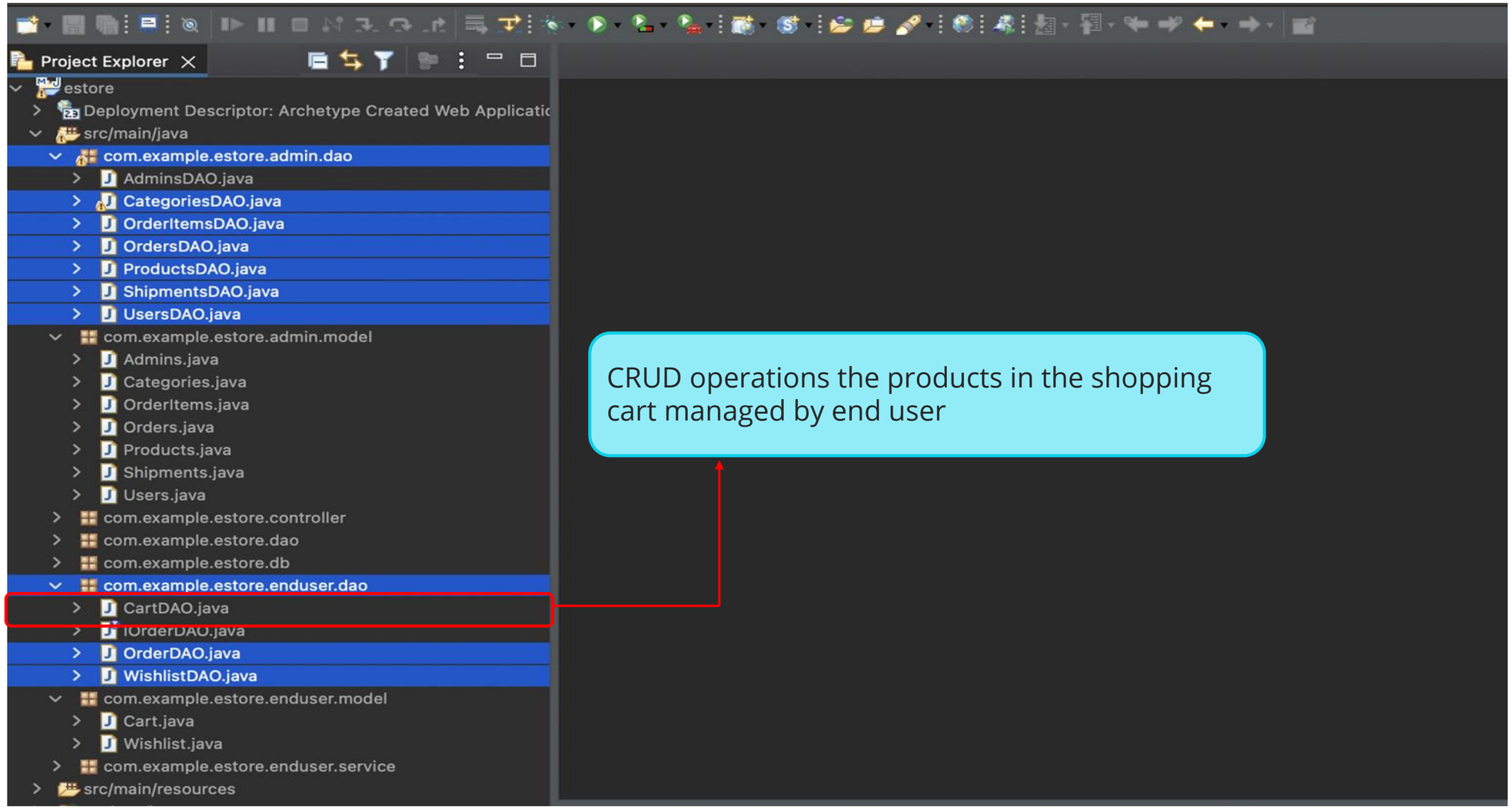
CRUD operations for the products along with category to be added by admin for end users

Create Classes Implementing DAO for the End User Web App

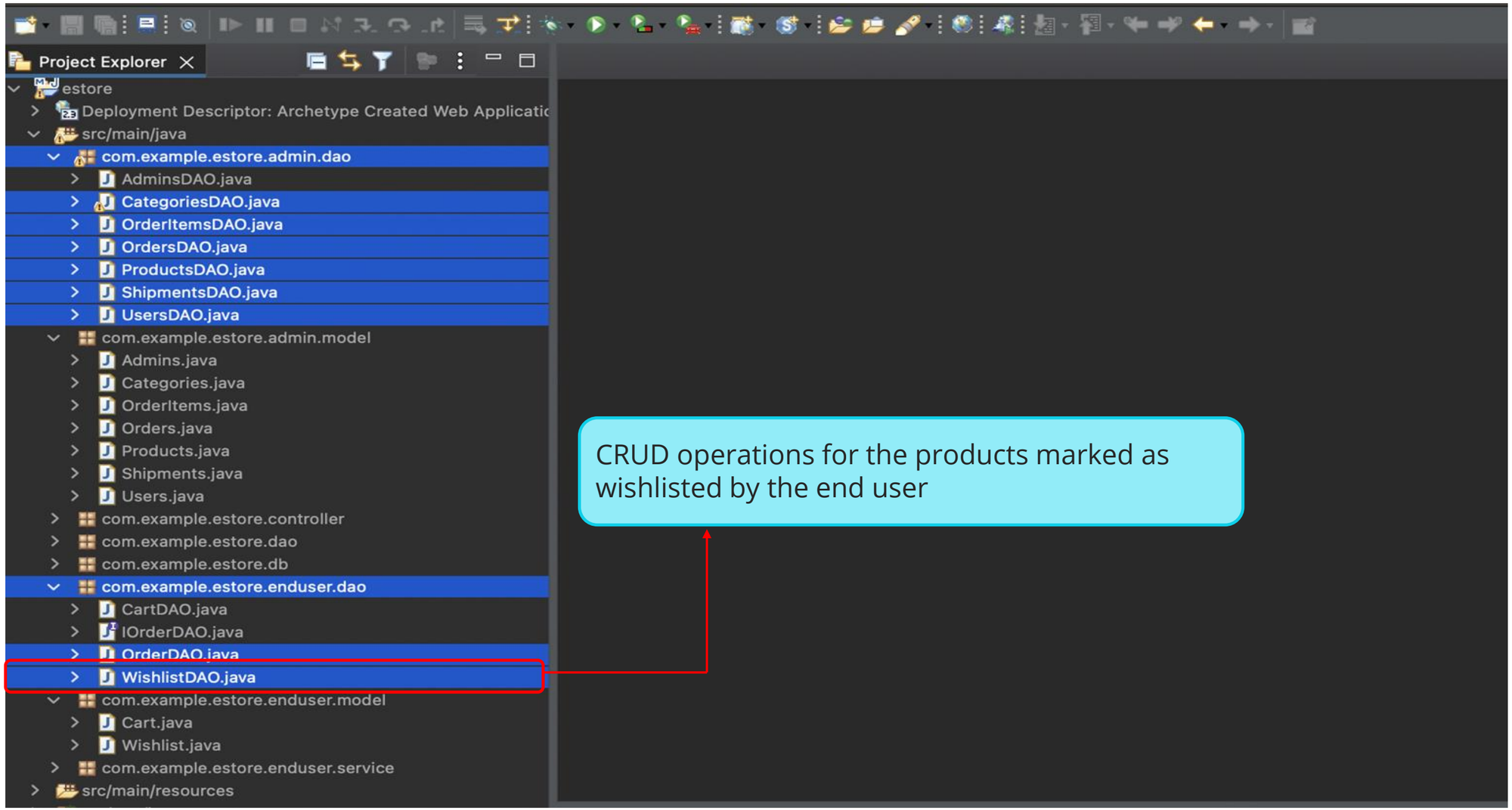


CRUD operations for the placed order shipped for a user

Create Classes Implementing DAO for the End User Web App



Create Classes Implementing DAO for the End User Web App



CRUD operations for the products marked as wishlisted by the end user

UsersDAO.java: Implementation of DAO Methods

In order to perform CRUD operations for the user, implement DAO methods.

User Table

User Model

In the usersDAO class, initialize the DB to make connection and execute SQL Statements.

UsersDAO.java: Implementation of DAO Methods

CRUD operations use DB.java methods internally to:

executeUpdate for insert, update and delete of the User record

executeQuery to fetch the details of a User based on ID or fetch the details of all the users as listed

UsersDAO.java: Implementation of DAO Methods

```
package com.example.estore.admin.dao;

public class usersDAO implements DAO<users>{

      DB db = DB.getDB();

      @Override
      public users get(long id) {
            // TODO Auto:generated method stub
            return null;
      }

      @Override
      public List<users> getAll() {
            // TODO Auto:generated method stub
            return null;
      }
```

UsersDAO.java: Implementation of DAO Methods

```
        @Override
        public void save(users object) {
              // TODO Auto:generated method stub
        }

        @Override
        public void update(users object) {
              // TODO Auto:generated method stub
        }

        @Override
        public void delete(long id) {
              // TODO Auto:generated method stub

        }

  }
```

UserDAO.java: Login Method for Authentication

In order to register or login, the user can create separate business methods for register and login.

For the same, create an additional method, other than CRUD Operations.

UserDAO.java: Login Method for Authentication

```
public users login(users object) {

            users User= new users();

            try {
                  String sql = "select * from users
where email = '"+object.getEmail()+"' and password =
'"+object.getPassword()+"'";
                  ResultSet set =
db.executeQuery(sql);
                  if(set.next()) {

      object.setuserId(set.getInt("userId"));

      object.setFullName(set.getString("fullName"));

      object.setEmail(set.getString("email"));
                        String date =
```

UserDAO.java: Login Method for Authentication

```
set.getString("addedOn");
                        SimpleDateFormat format =
new SimpleDateFormat("YYYY:MM:DD");
                        Date addedOn =
format.parse(date);
                        object.setAddedOn(addedOn);
                  }
            }catch(Exception e) {
                  System.out.println("Something went
wrong: "+e);
            }

            return user;
      }
```

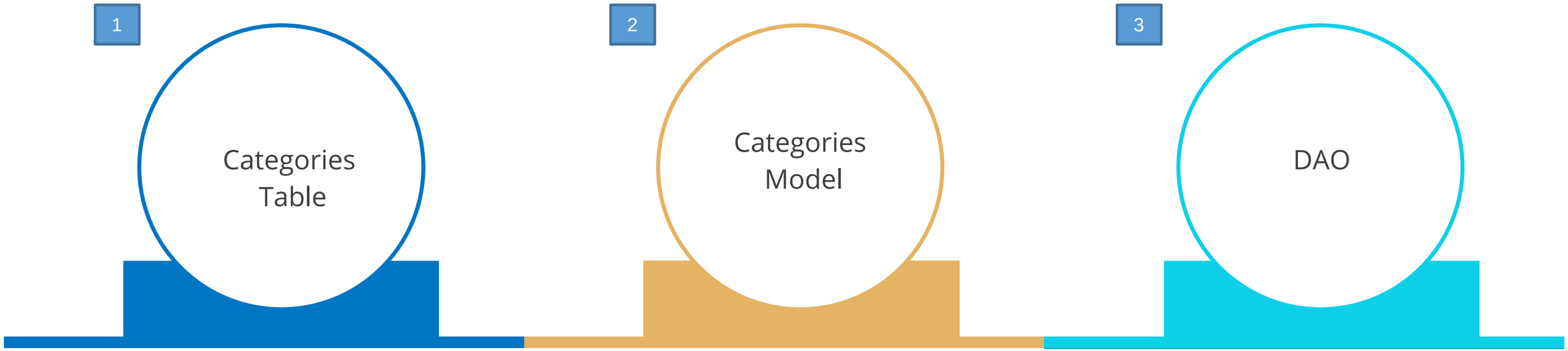
UserDAO.java: Login Method for Authentication

Register method goes something like this:

```
public String registeruser(users object) {

            users User= new users();

            // DB Code goes here

            return "Thank you for Registering";

      }
```

CategoriesDAO.java: Implementation of DAO Methods

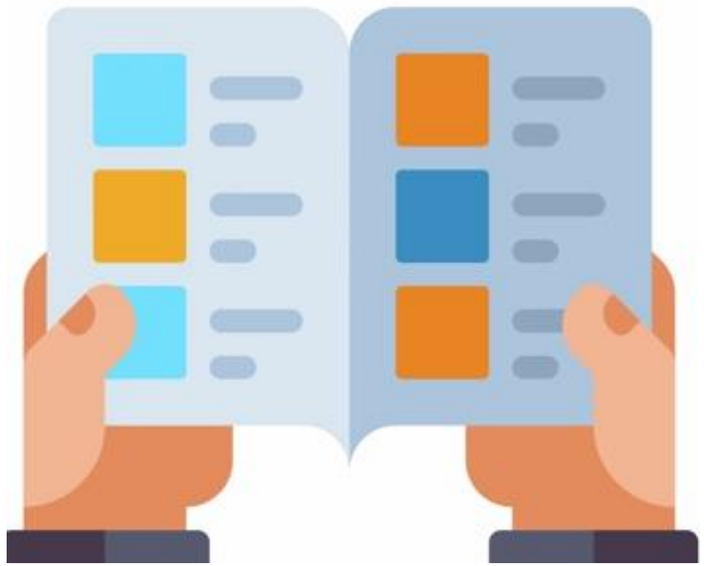
In order to perform CRUD operations for the user:



CategoriesDAO.java: Implementation of DAO Methods

Implement the methods from the DAO to perform CRUD operations and use DB.java methods internally to:

executeQuery to fetch the details of product categories



CategoriesDAO.java: Implementation of DAO Methods

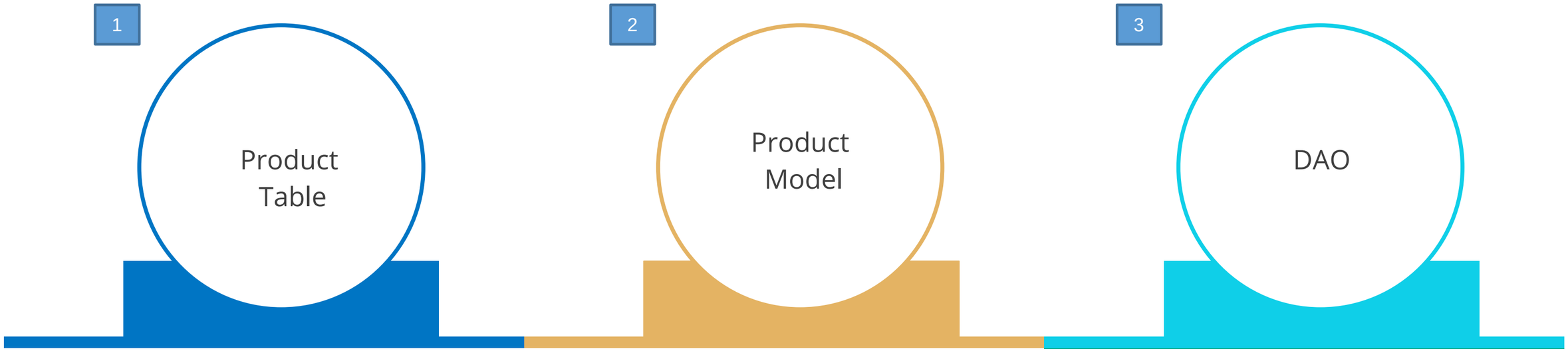
```
package com.example.estore.admin.dao;

public class CategoriesDAO implements
DAO<Categories>{

      DB db = DB.getDB();

      @Override
      public Categories get(long id) {
            // TODO Auto:generated method stub
            return null;
      }

      @Override
      public List<Categories> getAll() {
            // TODO Auto:generated method stub
            return null;
      }
```

CategoriesDAO.java: Implementation of DAO Methods

```
        @Override
         public void save(Categories object) {
               // TODO Auto:generated method stub
         }

    }

         @Override
         public void update(Categories object) {
               // TODO Auto:generated method stub
         }
         @Override
         public void delete(long id) {
               // TODO Auto:generated method stub

         }

    }
```

ProductsDAO.java: Implementation of DAO Methods

In order to list the products to the end user:



ProductsDAO.java: Implementation of DAO Methods

Implement the methods from the DAO to perform CRUD operations and use DB.java methods internally to:

executeQuery to fetch the details of products to be displayed on web pages

ProductsDAO.java: Implementation of DAO Methods

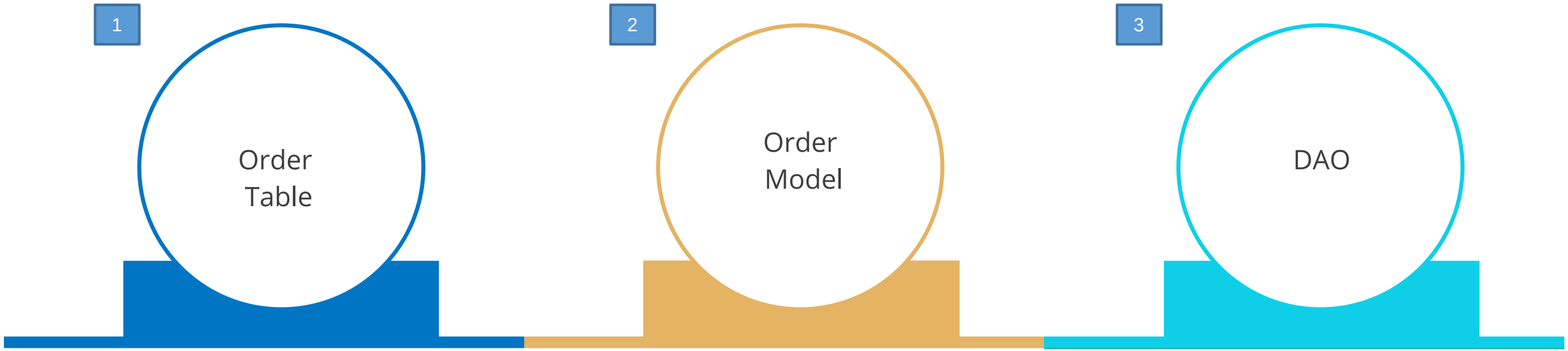
```
package com.example.estore.admin.dao;


public class ProductsDAO implements DAO<Products>{

      DB db = DB.getDB();

      @Override
      public Products get(long id) {
            // TODO Auto:generated method stub
            return null;
      }

      @Override
      public List<Products> getAll() {
            // TODO Auto:generated method stub
            return null;
      }
```

ProductsDAO.java: Implementation of DAO Methods

```
@Override
      public void save(Products object) {
            // TODO Auto:generated method stub
      }

package com.example.estore.admin.dao;

      @Override
      public void update(Products object) {
            // TODO Auto:generated method stub
      }

      @Override
      public void delete(long id) {
            // TODO Auto:generated method stub

      }
}
```

OrdersDAO.java: Implementation of DAO Methods

In Order to list and create the orders by End user:



OrdersDAO.java: Implementation of DAO Methods

Implement the methods from the DAO to perform CRUD operations and use DB.java methods internally to:

- executeUpdate for insert, update, and delete of the order record for the user
- executeQuery to fetch the details of orders to be displayed on web pages

OrdersDAO.java: Implementation of DAO Methods

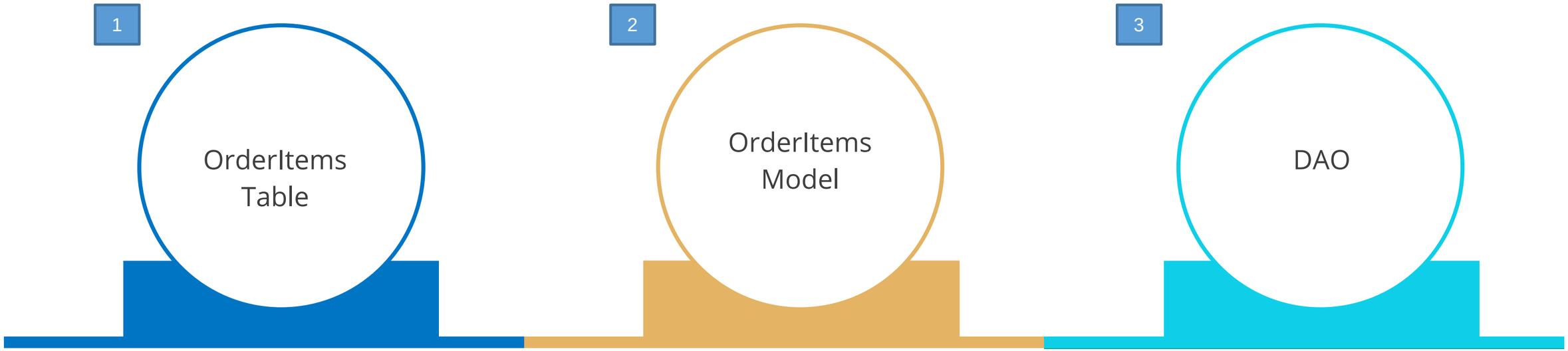
```
package com.example.estore.admin.dao;

public class OrdersDAO implements DAO<Orders>{

      DB db = DB.getDB();

      @Override
      public Orders get(long id) {
            // TODO Auto:generated method stub
            return null;
      }

      @Override
      public List<Orders> getAll() {
            // TODO Auto:generated method stub
            return null;
      }
```

OrdersDAO.java: Implementation of DAO Methods

```
        @Override
        public void save(Orders object) {
              // TODO Auto:generated method stub

        }

        @Override
        public void update(Orders object) {
              // TODO Auto-generated method stub

        }

        @Override
        public void delete(long id) {
              // TODO Auto-generated method stub
        }
  }
```

OrderItemsDAO.java: Implementation of DAO Methods

In order to perform CRUD operations for the orderitems and details of the orderitems:



OrderItemsDAO.java: Implementation of DAO Methods

Implement the methods from the DAO to perform CRUD operations and use DB.java methods internally to:

- executeUpdate for insert, update, and delete of the OrderItems record
- execute Query to fetch the details of a Userbased on ID or fetch the details of all the OrderItems as listed

OrderItemsDAO.java: Implementation of DAO Methods

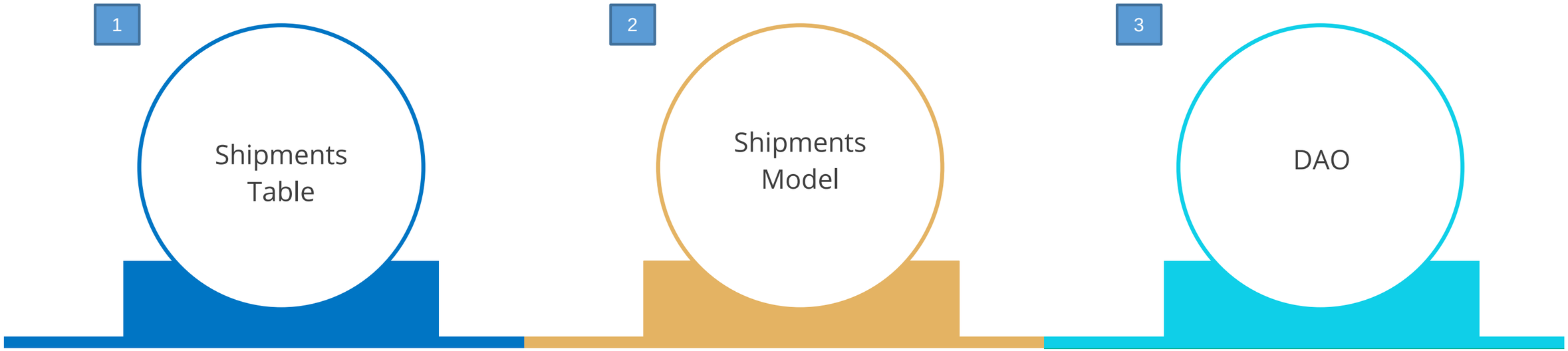
```
package com.example.estore.admin.dao;

public class OrderItemsDAO implements
DAO<OrderItems>{

      DB db = DB.getDB();

      @Override
      public OrderItems get(long id) {
            // TODO Auto:generated method stub
            return null;
      }

      @Override
      public List<OrderItems> getAll() {
            // TODO Auto:generated method stub
            return null;
      }
```

OrderItemsDAO.java: Implementation of DAO Methods

```
 @Override
public void save(OrderItems object) {
      // TODO Auto:generated method stub

}

@Override
public void update(OrderItems object) {
      // TODO Auto:generated method stub

}

@Override
public void delete(long id) {
      // TODO Auto:generated method stub

}

}
```

ShipmentsDAO.java: Implementation of DAO Methods

In order to know the status and details of the shipments:



In the ShipmentsDAO class, initialize the DB so as to make a connection and execute SQL Statements.

ShipmentsDAO.java: Implementation of DAO Methods

Implement the methods from the DAO to perform CRUD operations and use DB.java methods internally to:

execute Query to fetch the details of Shipment for a user

ShipmentsDAO.java: Implementation of DAO Methods

```
package com.example.estore.admin.dao;


public class ShipmentsDAO implements DAO<Shipments>{

      DB db = DB.getDB();

      @Override
      public Shipments get(long id) {
            // TODO Auto:generated method stub
            return null;
      }

      @Override
      public List<Shipments> getAll() {
            // TODO Auto:generated method stub
            return null;
      }
```

ShipmentsDAO.java: Implementation of DAO Methods

```
        @Override
         public void save(Shipments object) {
               // TODO Auto:generated method stub

         }

        @Override
         public void update(Shipments object) {
               // TODO Auto:generated method stub

         }

         @Override
         public void delete(long id) {
               // TODO Auto:generated method stub

         }
}
```

CartDAO.java: Implementation of DAO Methods

In order to perform CRUD operations for the shopping cart and details of the products:

Cart Table

Cart Model

In the CartDAO class, initialize the DB to make a connection and execute SQL Statements.

CartDAO.java: Implementation of DAO Methods

Implement the methods from the DAO to perform CRUD operations and use DB.Java methods internally to:

- Executeupdate for insert, update ,and delete of the products record in the shopping cart
- Execute query to fetch the details of a user's shopping cart

CartDAO.java: Implementation of DAO Methods

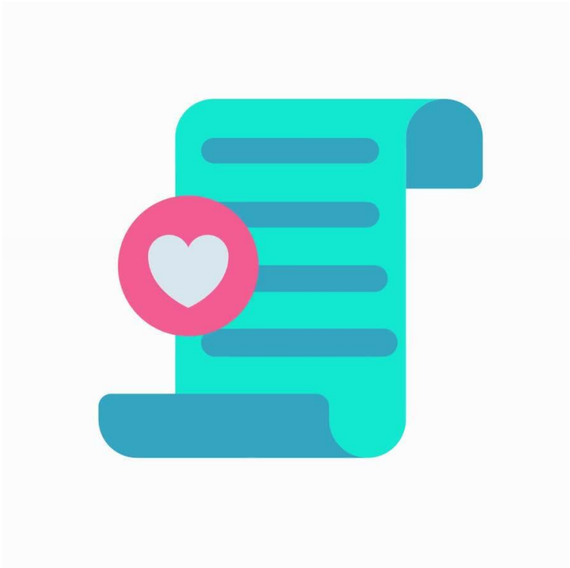
```
import com.example.estore.dao.DAO;
import com.example.estore.enduser.model.Cart;

public class CartDAO implements DAO<Cart>{

      @Override
      public Cart get(long id) {
            // TODO Auto:generated method stub
            return null;
      }
      @Override
      public List<Cart> getAll() {
            // TODO Auto:generated method stub
            return null;
      }
      @Override
      public void save(Cart object) {
            // TODO Auto:generated method stub
      }
```

CartDAO.java: Implementation of DAO Methods

```
        @Override
        public void update(Cart object) {
              // TODO Auto:generated method stub

        }

        @Override
        public void delete(long id) {
              // TODO Auto:generated method stub

        }

}
```

WishlistDAO.java: Implementation of DAO Methods

In order to perform CRUD operations for the wishlisted products, the user must implement DAO methods.



In the WishlistDAO class, initialize the DB to make a connection and execute SQL Statements.

WishlistDAO.java: Implementation of DAO Methods

Implement the methods from the DAO to perform CRUD operations and use DB.java methods internally to:

- executeUpdate for insert, update ,and delete of the Wishlist record
- execute Query to fetch the details of wishlisted product by end user

WishlistDAO.java: Implementation of DAO Methods

```
package com.example.estore.enduser.dao;

import com.example.estore.dao.DAO;
import com.example.estore.enduser.model.Wishlist;

public class WishlistDAO implements DAO<Wishlist>{

      @Override
      public Wishlist get(long id) {
            // TODO Auto:generated method stub
            return null;
      }

      @Override
      public List<Wishlist> getAll() {
            // TODO Auto:generated method stub
            return null;
      }
```

WishlistDAO.java: Implementation of DAO Methods

```
        @Override
        public void save(Wishlist object) {
              // TODO Auto:generated method stub

        }

        @Override
        public void update(Wishlist object) {
              // TODO Auto:generated method stub

        }

        @Override
        public void delete(long id) {
              // TODO Auto:generated method stub

        }
}
```

Key Takeaways

- DAO design pattern are implemented generically.
- Singleton Design Pattern are implemented for DB.
- CRUD operations are implemented for various models in End-User Web app.
- CRUD operations are tested for various models in End-User Web app.

Before the Next Class

Since you have successfully completed this session. Before next discussion you should go through:

- JUit
- Spring Boot

What’s Next?

Now we have finished our Classes and Design Pattern for the Backend Project with End Usermodule. In our next live session, we will :

- Explore how to create Servlets
- See how to use JDBC with Servlets
- Perform CRUD Operations with DB
- Work with Design Patterns