**Case Study on**

**Ecom Application**

**Name : J316 Sherin Sandra J**

**Date : 11 / 04 / 2025**

**Trainer : Mr. Madhu Kalla**

**Table of Contents:**

|  |  |
| --- | --- |
| **Topic** | **Page** |
| 1. Introduction | **1** |
| 1. Problem Statement | **1** |
| 1. Directory Structure | **2** |
| 1. Database Schema | **3** |
| 1. SQL Queries | **4** |
| 1. Java Implementation | **9** |
| 1. Output | **42** |
| 1. Unit Testing | **46** |

**Introduction:**

This document provides an overview of the E-Commerce Application Case Study, including the problem statement, solution approach, database schema, and key SQL queries. It covers the Java implementation of the project, detailing the system's design and coding. Additionally, the document explains the unit testing process using JUnit to ensure the reliability and functionality of the system. The aim is to present the full development cycle, from database design to testing, ensuring the system's correctness and effectiveness.

**Problem Statement:**

Online shopping has become a necessity nowadays. It is the responsibility of the Business owners to provide a seamless experience to their customers where they can browse, add items to cart, place and track orders.

Many small businesses face these challenges:

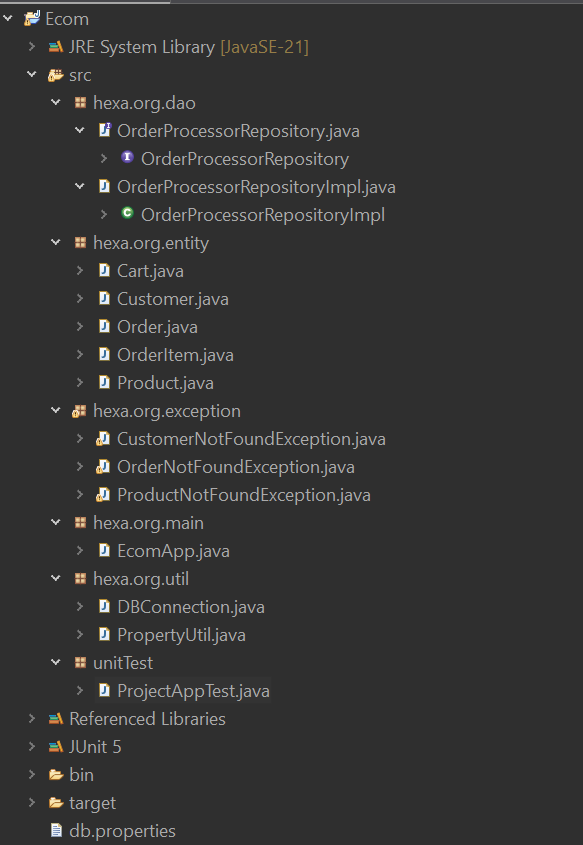
* Managing customer data efficiently.
* Handling a large number of products.
* Tracking shopping carts and orders.

**Goal :**

The goal of this project is to develop an E-commerce system using SQL, control flow, loops, arrays, collections, and exception handling. It follows object-oriented principles to model real-world entities and ensures code re-usability. Custom exceptions are thrown and handled, and unit testing is implemented to verify system functionality.

**Solution Approach:**

The application will follow a structured **Directory structure** as below:



**Database Schema Design:**

**Overview:**

The E-Com Application database is designed to manage customers, products, carts, orders, and order details effectively. It ensures relationship between tables using primary and foreign keys, allowing interaction between different table columns.

Create database ecomproject;



Use ecomproject;



**The database consists of five main tables:**

1. Customers

2. Products

3. Cart

4. Orders

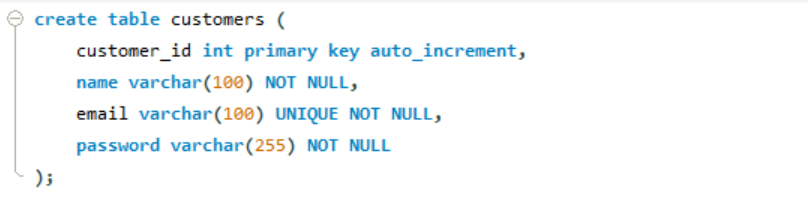
5. Order\_items

Each table has a specific purpose in making the Ecom application work efficiently.

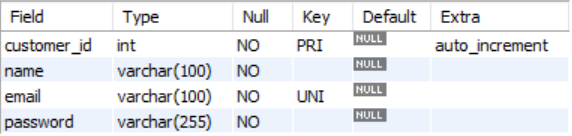
**SQL Queries:**

Tables:

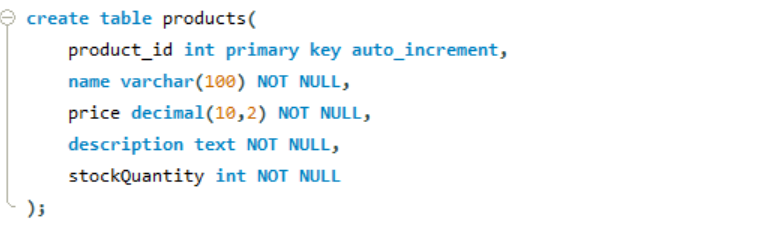
1. The customers table is used to store customer information including their names, emails and passwords. All these fields are given the NOT NULL constraint, as they are required for the system to function properly. The id field is set to primary key with auto\_increment to ensure each record has unique value and increments automatically.



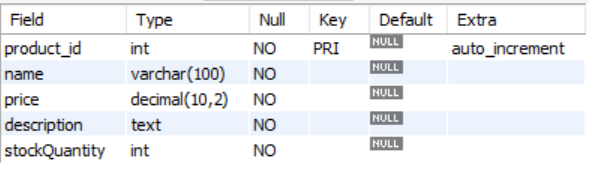
desc customers;



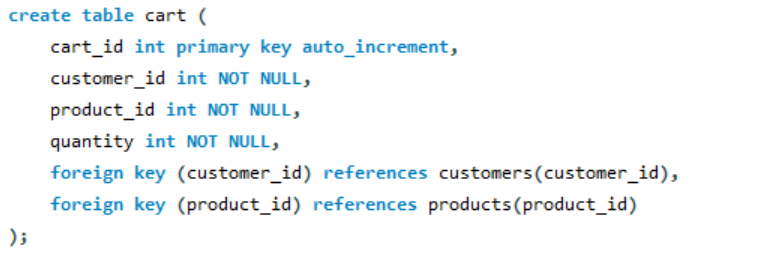
1. The products table stores product details such as name, price, description and stock quantity. All these fields are given the NOT NULL constraint, as they are required for the system to function properly. The id field is set to primary key with auto\_increment to ensure each record has unique value and increments automatically.The price field is given decimal constraint to accommodate decimal upto two places.



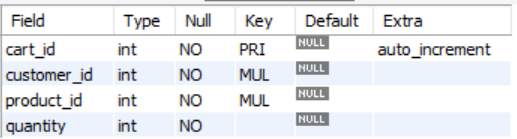
desc products;



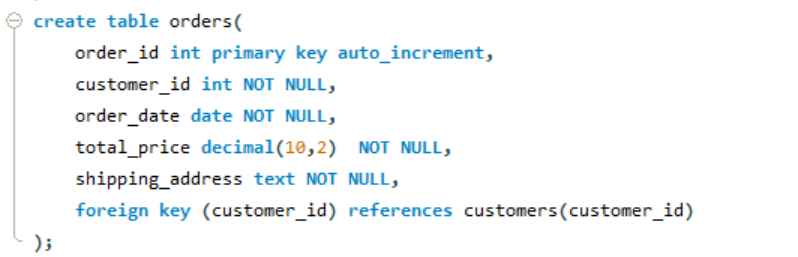
1. The cart table is used to track items added to the shopping cart by the customers. It also includes NOT NULL constraint to ensure all necessary fields have values. Foreign key references for customer\_id and product\_id are added, linking the table to the cutomer and product table respectively.



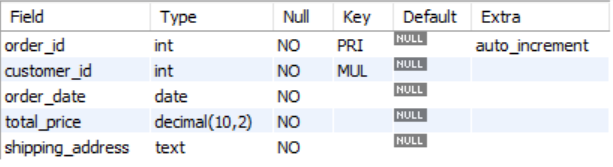
desc cart;



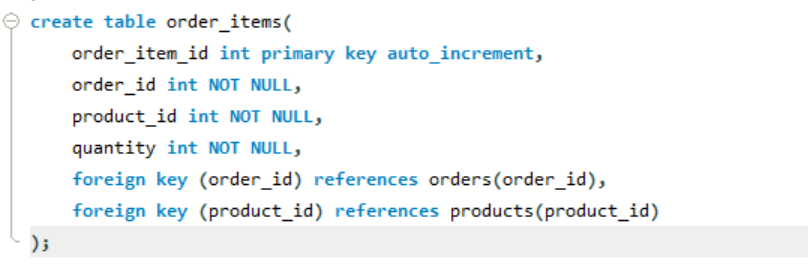
1. The orders table stores customer orders, including total price and shipping details. The order\_id field is set to primary key so that it has unique values and auto\_increment to increment automatically. It also includes NOT NULL constraint to ensure all necessary fields have values. order\_date field is provided with the DATE constraint to store date of the order. Foreign key references for customer\_id is added linking the customer table.



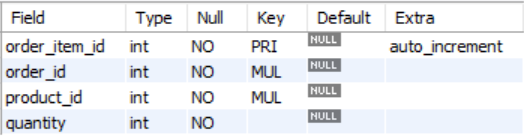
desc orders;



1. The order\_items table stores details of items purchased in each order. All these fields are given the NOT NULL constraint, as they are required for the system to function properly. The order\_item\_id field is set to primary key with auto\_increment to ensure each record has unique value and increments automatically. Foreign key references for order\_id and product\_id are added, linking the table to the orders and product table respectively.



desc order\_items;



**In the Eclipse IDE, following was performed:**

1. **Creating the model/entity classes corresponding to the schema within package entity with variables declared private, constructors(default and parametrized) and getters,setters )**

**Product.java**

|  |
| --- |
| package hexa.org.entity;  public class Product {  private int productId;  private String name;  private double price;  private String description;  private int stockQuantity;      public Product() {  super();  }  public Product(int productId, String name, double price, String description, int stockQuantity) {  super();  this.productId = productId;  this.name = name;  this.price = price;  this.description = description;  this.stockQuantity = stockQuantity;  }  public int getProductId() {  return productId;  }  public void setProductId(int productId) {  this.productId = productId;  }  public String getName() {  return name;  }  public void setName(String name) {  this.name = name;  }  public double getPrice() {  return price;  }  public void setPrice(double price) {  this.price = price;  }  public String getDescription() {  return description;  }  public void setDescription(String description) {  this.description = description;  }  public int getStockQuantity() {  return stockQuantity;  }  public void setStockQuantity(int stockQuantity) {  this.stockQuantity = stockQuantity;  }  @Override  public String toString() {  return "Product [productId=" + productId + ", name=" + name + ", price=" + price + ", description="  + description + ", stockQuantity=" + stockQuantity + "]";  }    } |

**Customer.java**

|  |
| --- |
| package hexa.org.entity;  public class Customer {  private int customerId;  private String name;  private String email;  private String password;    public Customer() {  super();  }  public Customer(int customerId, String name, String email, String password) {  super();  this.customerId = customerId;  this.name = name;  this.email = email;  this.password = password;  }  public int getCustomerId() {  return customerId;  }  public void setCustomerId(int customerId) {  this.customerId = customerId;  }  public String getName() {  return name;  }  public void setName(String name) {  this.name = name;  }  public String getEmail() {  return email;  }  public void setEmail(String email) {  this.email = email;  }  public String getPassword() {  return password;  }  public void setPassword(String password) {  this.password = password;  }  @Override  public String toString() {  return "Customer [customerId=" + customerId + ", name=" + name + ", email=" + email + ", password=" + password  + "]";  }  } |

**Cart.java**

|  |
| --- |
| package hexa.org.entity;  public class Cart {  private int cartId;  private int customerId;  private int productId;  private int quantity;    public Cart() {  super();  }  public Cart(int cartId, int customerId, int productId, int quantity) {  super();  this.cartId = cartId;  this.customerId = customerId;  this.productId = productId;  this.quantity = quantity;  }  public int getCartId() {  return cartId;  }  public void setCartId(int cartId) {  this.cartId = cartId;  }  public int getCustomerId() {  return customerId;  }  public void setCustomerId(int customerId) {  this.customerId = customerId;  }  public int getProductId() {  return productId;  }  public void setProductId(int productId) {  this.productId = productId;  }  public int getQuantity() {  return quantity;  }  public void setQuantity(int quantity) {  this.quantity = quantity;  }  @Override  public String toString() {  return "Cart [cartId=" + cartId + ", customerId=" + customerId + ", productId=" + productId + ", quantity="  + quantity + "]";  }  } |

**Order.java**

|  |
| --- |
| package hexa.org.entity;  import java.util.Date;  public class Order {  private int orderId;  private int customerId;  private Date orderDate;  private double totalPrice;  private String shippingAddress;    public Order() {  super();  }  public Order(int orderId, int customerId, Date orderDate, double totalPrice, String shippingAddress) {  super();  this.orderId = orderId;  this.customerId = customerId;  this.orderDate = orderDate;  this.totalPrice = totalPrice;  this.shippingAddress = shippingAddress;  }  public int getOrderId() {  return orderId;  }  public void setOrderId(int orderId) {  this.orderId = orderId;  }  public int getCustomerId() {  return customerId;  }  public void setCustomerId(int customerId) {  this.customerId = customerId;  }  public Date getOrderDate() {  return orderDate;  }  public void setOrderDate(Date orderDate) {  this.orderDate = orderDate;  }  public double getTotalPrice() {  return totalPrice;  }  public void setTotalPrice(double totalPrice) {  this.totalPrice = totalPrice;  }  public String getShippingAddress() {  return shippingAddress;  }  public void setShippingAddress(String shippingAddress) {  this.shippingAddress = shippingAddress;  }  @Override  public String toString() {  return "Order [orderId=" + orderId + ", customerId=" + customerId + ", orderDate=" + orderDate + ", totalPrice="  + totalPrice + ", shippingAddress=" + shippingAddress + "]";  }    } |

**OrderItem.java**

|  |
| --- |
| package hexa.org.entity;  public class OrderItem {  private int orderItemId;  private int orderId;  private int productId;  private int quantity;    public OrderItem() {  super();  }  public OrderItem(int orderItemId, int orderId, int productId, int quantity) {  super();  this.orderItemId = orderItemId;  this.orderId = orderId;  this.productId = productId;  this.quantity = quantity;  }  public int getOrderItemId() {  return orderItemId;  }  public void setOrderItemId(int orderItemId) {  this.orderItemId = orderItemId;  }  public int getOrderId() {  return orderId;  }  public void setOrderId(int orderId) {  this.orderId = orderId;  }  public int getProductId() {  return productId;  }  public void setProductId(int productId) {  this.productId = productId;  }  public int getQuantity() {  return quantity;  }  public void setQuantity(int quantity) {  this.quantity = quantity;  }    @Override  public String toString() {  return "OrderItem [orderItemId=" + orderItemId + ", orderId=" + orderId + ", productId=" + productId  + ", quantity=" + quantity + "]";  }    } |

1. **Creating a Service Provider Interface/Abstract class:**

Keep the interfaces and implementation classes in package dao

Define an OrderProcessorRepository interface/abstract class with methods for

1. createProduct()

2. createCustomer()

3. deleteProduct()

4. deleteCustomer(customerId)

5. addToCart()

6. removeFromCart()

7. getAllFromCart(Customer customer)

8. placeOrder(Customer customer, List<Map<Product,quantity>>, string Address)

9. getOrdersByCustomer()

**OrderProcessorRepository.java**

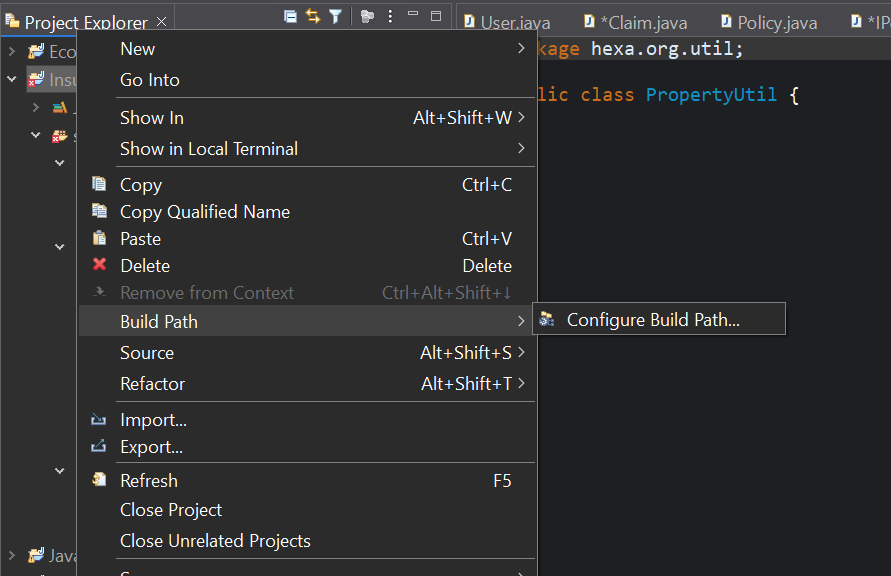
|  |
| --- |
| package hexa.org.dao;  import java.util.List;  import java.util.Map;  import hexa.org.entity.Customer;  import hexa.org.entity.Product;  import hexa.org.exception.CustomerNotFoundException;  import hexa.org.exception.OrderNotFoundException;  import hexa.org.exception.ProductNotFoundException;  public interface OrderProcessorRepository {  boolean createProduct(Product product);  boolean createCustomer(Customer customer);  boolean deleteProduct(int productId) throws ProductNotFoundException;  boolean deleteCustomer(int customerId) throws CustomerNotFoundException;  boolean addToCart(Customer customer, Product product, int quantity);  boolean removeFromCart(Customer customer, Product product);  List<Product> getAllFromCart(Customer customer) throws CustomerNotFoundException;  boolean placeOrder(Customer customer, List<Map<Product, Integer>> productList, String shippingAddress) throws CustomerNotFoundException, ProductNotFoundException;  List<Map<Product, Integer>> getOrdersByCustomer(int customerId) throws CustomerNotFoundException, OrderNotFoundException;  } |

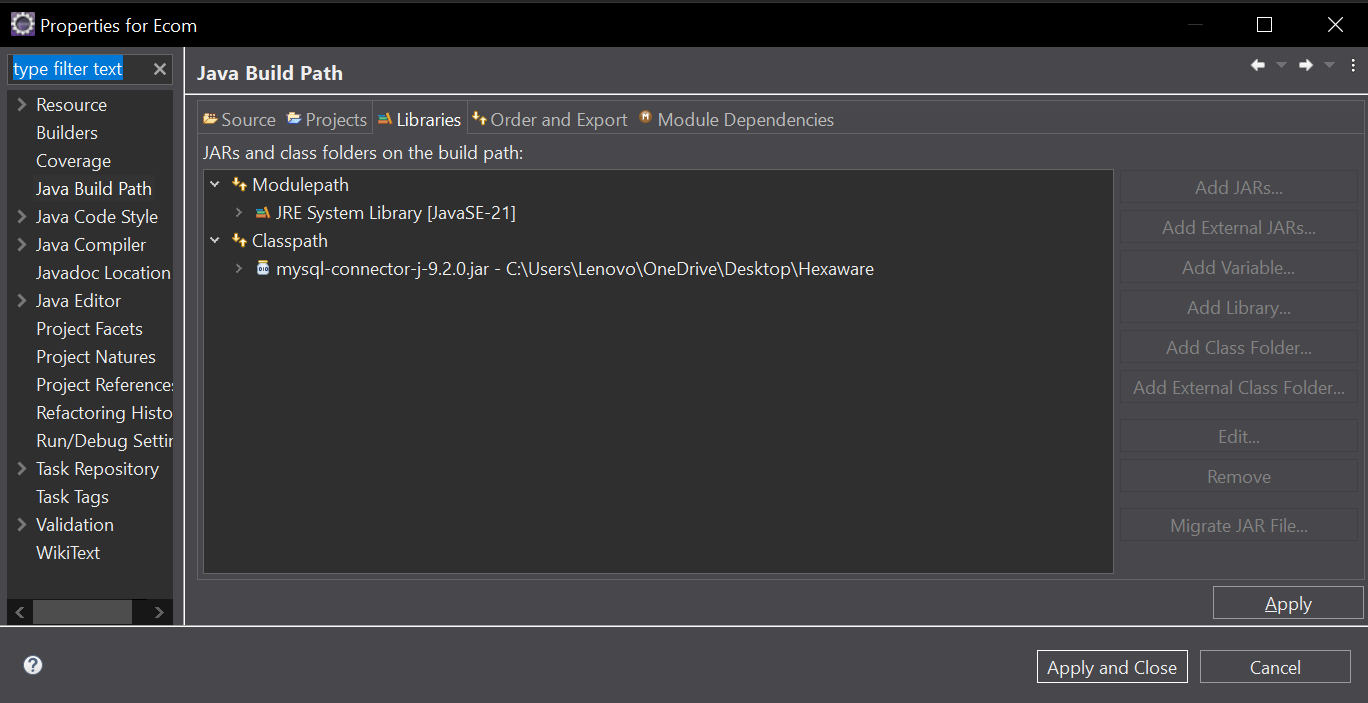
1. **Implementing the above interface in a class called OrderProcessorRepositoryImpl in package dao**

**OrderProcessorRepositoryImpl .java**

|  |
| --- |
| package hexa.org.dao;  import java.sql.Connection;  import java.sql.Date;  import java.sql.PreparedStatement;  import java.sql.ResultSet;  import java.sql.SQLException;  import java.sql.Statement;  import java.util.ArrayList;  import java.util.HashMap;  import java.util.List;  import java.util.Map;  import hexa.org.entity.Customer;  import hexa.org.entity.Product;  import hexa.org.exception.CustomerNotFoundException;  import hexa.org.exception.OrderNotFoundException;  import hexa.org.exception.ProductNotFoundException;  import hexa.org.util.DBConnection;  public class OrderProcessorRepositoryImpl implements OrderProcessorRepository {  public OrderProcessorRepositoryImpl() {  super();  }  @Override  public boolean createProduct(Product product) {  boolean flag=false;  try {  Connection con = DBConnection.getConnection();  PreparedStatement ps = con.prepareStatement("INSERT INTO products (name, price, description, stockQuantity) VALUES (?, ?, ?, ?)");  ps.setString(1, product.getName());  ps.setDouble(2, product.getPrice());  ps.setString(3, product.getDescription());  ps.setInt(4, product.getStockQuantity());    ps.executeUpdate();  flag=true;  }catch(SQLException se) {  System.out.println("Error While Inserting Product...");  se.printStackTrace();  }  return flag;  }  @Override  public boolean createCustomer(Customer customer) {  boolean flag = false;  try {  Connection con=DBConnection.getConnection();  PreparedStatement ps = con.prepareStatement("INSERT INTO customers (name, email, password) VALUES (?, ?, ?)");  ps.setString(1, customer.getName());  ps.setString(2, customer.getEmail());  ps.setString(3, customer.getPassword());    ps.executeUpdate();  flag=true;  }catch(SQLException se) {  System.out.println("Error While Inserting Customer...");  se.printStackTrace();  }  return flag;  }  @Override  public boolean deleteProduct(int productId) throws ProductNotFoundException {  boolean flag = false;  try {  Connection con = DBConnection.getConnection();  PreparedStatement check = con.prepareStatement("select \* from products where product\_id = ?");  check.setInt(1, productId);  ResultSet rs = check.executeQuery();  if (!rs.next()) {  throw new ProductNotFoundException("Product not found...");  }  PreparedStatement ps = con.prepareStatement("delete from products where product\_id = ?");  ps.setInt(1, productId);  int rows = ps.executeUpdate();  if (rows > 0) {  flag = true;  }  } catch (SQLException se) {  System.out.println("Error while deleting product...");  se.printStackTrace();  }  return flag;  }  @Override  public boolean deleteCustomer(int customerId) throws CustomerNotFoundException {  boolean flag = false;  try {  Connection con = DBConnection.getConnection();  PreparedStatement check = con.prepareStatement("select \* from customers where customer\_id = ?");  check.setInt(1, customerId);  ResultSet rs = check.executeQuery();  if (!rs.next()) {  throw new CustomerNotFoundException("Customer not found...");  }  PreparedStatement ps = con.prepareStatement("delete from customers where customer\_id = ?");  ps.setInt(1, customerId);  int rows = ps.executeUpdate();  if (rows > 0) {  flag = true;  }  } catch (SQLException se) {  System.out.println("Error while deleting customer...");  se.printStackTrace();  }  return flag;  }  @Override  public boolean addToCart(Customer customer, Product product, int quantity) {  boolean flag=false;  try {  Connection con=DBConnection.getConnection();  PreparedStatement ps = con.prepareStatement("INSERT INTO cart (customer\_id, product\_id, quantity) VALUES (?, ?, ?)");  ps.setInt(1,customer.getCustomerId());  ps.setInt(2, product.getProductId());  ps.setInt(3, quantity);  ps.executeUpdate();  flag=true;  }catch(SQLException se) {  System.out.println("Error While Adding To Cart...");  se.printStackTrace();  }  return flag;  }  @Override  public boolean removeFromCart(Customer customer, Product product) {  boolean flag=false;  try {  Connection con=DBConnection.getConnection();  PreparedStatement ps=con.prepareStatement("DELETE FROM cart WHERE customer\_id=? AND product\_id=?");  ps.setInt(1, customer.getCustomerId());  ps.setInt(2, product.getProductId());  int rows = ps.executeUpdate();  if (rows > 0) {  flag = true;  }  }catch(SQLException se) {  System.out.println("Error While Removing From Cart...");  se.printStackTrace();  }  return flag;  }  @Override  public List<Product> getAllFromCart(Customer customer) throws CustomerNotFoundException {  List<Product> productList = new ArrayList<>();  try {  Connection con=DBConnection.getConnection();    PreparedStatement check = con.prepareStatement("select \* from customers where customer\_id = ?");  check.setInt(1, customer.getCustomerId());  ResultSet rs = check.executeQuery();  if (!rs.next()) {  throw new CustomerNotFoundException("Customer not found...");  }  PreparedStatement ps=con.prepareStatement("select p.\* from products p join cart c on p.product\_id=c.product\_id where c.customer\_id=?");  ps.setInt(1,customer.getCustomerId());  ResultSet cartrs=ps.executeQuery();  while(cartrs.next()) {  Product cartItem =new Product();  cartItem.setProductId(cartrs.getInt("product\_id"));  cartItem.setName(cartrs.getString("name"));  cartItem.setPrice(cartrs.getDouble("price"));  cartItem.setDescription(cartrs.getString("description"));  cartItem.setStockQuantity(cartrs.getInt("stockQuantity"));  productList.add(cartItem);  }  }catch (SQLException se) {  System.out.println("Error While Fetching Cart Items...");  se.printStackTrace();  }  return productList;  }    @Override  public boolean placeOrder(Customer customer, List<Map<Product, Integer>> productList, String shippingAddress) throws CustomerNotFoundException, ProductNotFoundException {  boolean flag = false;  double totalPrice = 0.0;  Date orderDate = new Date(System.currentTimeMillis());  try {  Connection con = DBConnection.getConnection();  PreparedStatement checkCustomer = con.prepareStatement("select \* from customers where customer\_id = ?");  checkCustomer.setInt(1, customer.getCustomerId());  ResultSet rc = checkCustomer.executeQuery();  if (!rc.next()) {  throw new CustomerNotFoundException("Customer not found.");  }  for (Map<Product, Integer> map : productList) {  for (Map.Entry<Product, Integer> entry : map.entrySet()) {  Product p = entry.getKey();  int quantity = entry.getValue();  totalPrice += p.getPrice() \* quantity;  PreparedStatement checkProduct = con.prepareStatement("select \* from products where product\_id = ?");  checkProduct.setInt(1, p.getProductId());  ResultSet rp = checkProduct.executeQuery();  if (!rp.next()) {  throw new ProductNotFoundException("Product with ID " + p.getProductId() + " not found.");  }  }  }  PreparedStatement ps = con.prepareStatement("insert into orders (customer\_id, order\_date, total\_price, shipping\_address) values (?, ?, ?, ?)", Statement.RETURN\_GENERATED\_KEYS);  ps.setInt(1, customer.getCustomerId());  ps.setDate(2, orderDate);  ps.setDouble(3, totalPrice);  ps.setString(4, shippingAddress);  ps.executeUpdate();  ResultSet rs = ps.getGeneratedKeys();  int orderId = 0;  if (rs.next()) {  orderId = rs.getInt(1);  }  PreparedStatement pstmt = con.prepareStatement("insert into order\_items (order\_id, product\_id, quantity) values (?, ?, ?)");  for (Map<Product, Integer> map : productList) {  for (Map.Entry<Product, Integer> entry : map.entrySet()) {  Product p = entry.getKey();  int quantity = entry.getValue();  pstmt.setInt(1, orderId);  pstmt.setInt(2, p.getProductId());  pstmt.setInt(3, quantity);  pstmt.executeUpdate();  }  }  PreparedStatement clearCart = con.prepareStatement("delete from cart where customer\_id = ?");  clearCart.setInt(1, customer.getCustomerId());  clearCart.executeUpdate();  flag = true;  } catch (SQLException se) {  System.out.println("Error while placing order...");  se.printStackTrace();  }  return flag;  }    @Override  public List<Map<Product, Integer>> getOrdersByCustomer(int customerId) throws CustomerNotFoundException, OrderNotFoundException {  List<Map<Product, Integer>> orderList = new ArrayList<>();  try {  Connection con = DBConnection.getConnection();    PreparedStatement checkCustomer = con.prepareStatement("select \* from customers where customer\_id = ?");  checkCustomer.setInt(1, customerId);  ResultSet rc = checkCustomer.executeQuery();  if (!rc.next()) {  throw new CustomerNotFoundException("Customer not found...");  }  PreparedStatement orderStmt = con.prepareStatement("select order\_id from orders where customer\_id = ?");  orderStmt.setInt(1, customerId);  ResultSet orderRs = orderStmt.executeQuery();    if (!orderRs.isBeforeFirst()) {  throw new OrderNotFoundException("No orders found for customer...");  }  while (orderRs.next()) {  int orderId = orderRs.getInt("order\_id");  PreparedStatement itemStmt = con.prepareStatement("select product\_id, quantity from order\_items where order\_id = ?");  itemStmt.setInt(1, orderId);  ResultSet itemRs = itemStmt.executeQuery();  Map<Product, Integer> orderMap = new HashMap<>();  while (itemRs.next()) {  int productId = itemRs.getInt("product\_id");  int quantity = itemRs.getInt("quantity");  PreparedStatement productStmt = con.prepareStatement("select \* from products where product\_id = ?");  productStmt.setInt(1, productId);  ResultSet productRs = productStmt.executeQuery();  if (productRs.next()) {  Product product = new Product();  product.setProductId(productRs.getInt("product\_id"));  product.setName(productRs.getString("name"));  product.setPrice(productRs.getDouble("price"));  product.setDescription(productRs.getString("description"));  product.setStockQuantity(productRs.getInt("stockQuantity"));  orderMap.put(product, quantity);  }  }  orderList.add(orderMap);  }  } catch (SQLException se) {  System.out.println("Error fetching orders...");  se.printStackTrace();  }  return orderList;  }  } |

1. **To enable Java to communicate with the MySQL database via JDBC, the MySQL JDBC driver (mysql-connector-j-9.2.0.jar) must be added to the classpath.**





1. **Connecting the application to the SQL database:**

Writing code to establish a connection to your SQL database.

Create a utility class DBConnection in a package util with a static variable connection of Type Connection and a static method getConnection() which returns connection.Connection properties supplied in the connection string should be read from a property file.

Create a utility class PropertyUtil which contains a static method named getPropertyString() which reads a property file containing connection details like hostname, dbname, username, password, port number and returns a connection string.

**DBConnection.java**

|  |
| --- |
| package hexa.org.util;  import java.io.IOException;  import java.sql.Connection;  import java.sql.DriverManager;  import java.sql.SQLException;  public class DBConnection {    private static final String fileName="db.properties";  public static Connection getConnection() {  Connection con=null;  String connString=null;  try {  connString=PropertyUtil.getPropertyString(fileName);  }catch(IOException e) {  System.out.println("Connection String Creation Failed...");  e.printStackTrace();  }  if(connString!=null) {  try {  con=DriverManager.getConnection(connString);  }catch(SQLException e) {  System.out.println("Error While Establishing DBConnection...");  e.printStackTrace();  }  }  return con;  }  } |

**PropertyUtil.java**

|  |
| --- |
| package hexa.org.util;  import java.io.FileInputStream;  import java.io.IOException;  import java.util.Properties;  public class PropertyUtil {  public static String getPropertyString(String fileName)throws IOException{  String connStr=null;  Properties props=new Properties();  FileInputStream fis=new FileInputStream(fileName);  props.load(fis);  String user=props.getProperty("username");  String password=props.getProperty("password");  String protocol=props.getProperty("protocol");  String system=props.getProperty("system");  String database=props.getProperty("dbname");  String port=props.getProperty("port");  connStr=protocol+"//"+system+":"+port+"/"+database+"?user="+user+"&password="+password;  return connStr;  }  } |

**db.properties**

|  |
| --- |
| protocol=jdbc:mysql:  system=localhost  port=3306  dbname=ecomproject  username=root  password=\*\*\*\*\*\*\*\* |

1. **Creating the exceptions in package exception and creating the following custom exceptions and throw them in methods whenever needed. Handle all the exceptions in main method,**

**• CustomerNotFoundException:** throw this exception when user enters an invalid customer id which doesn’t exist in db

**• ProductNotFoundException:** throw this exception when user enters an invalid product id which doesn’t exist in db

**• OrderNotFoundException:** throw this exception when user enters an invalid order id which doesn’t exist in db

|  |
| --- |
| **CustomerNotFoundException.java**  package hexa.org.exception;  public class CustomerNotFoundException extends Exception{  public CustomerNotFoundException (String message) {  super(message);  }  }  **OrderNotFoundException.java**  package hexa.org.exception;  public class OrderNotFoundException extends Exception{  public OrderNotFoundException (String message) {  super(message);  }  }  **ProductNotFoundException.java**  package hexa.org.exception;  public class ProductNotFoundException extends Exception{  public ProductNotFoundException (String message) {  super(message);  }  } |

1. **Creating a class named EcomApp with main method in app Trigger all the methods in service implementation class by user choose operation from the following menu.**

1. Register Customer.

2. Create Product.

3. Delete Product.

4. Add to cart.

5. View cart.

6. Place order.

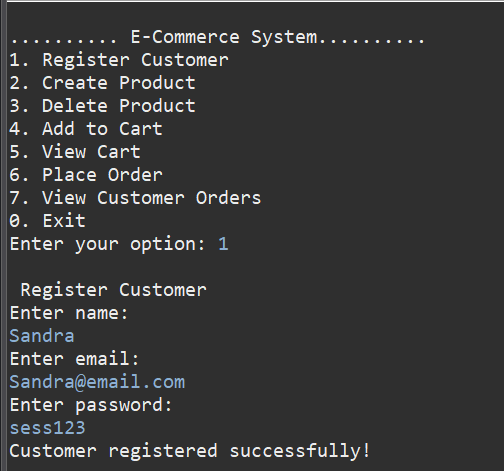
7. View Customer Order

**EcomApp.java**

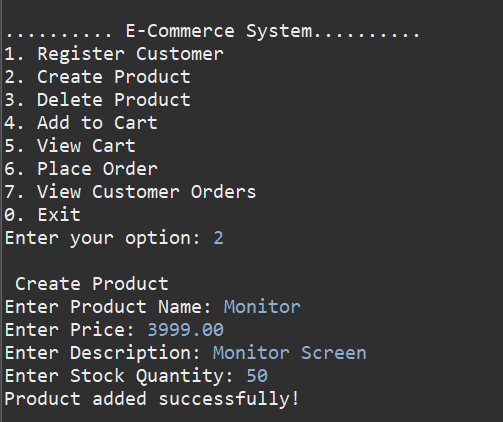
|  |
| --- |
| package hexa.org.main;  import java.util.\*;  import hexa.org.dao.\*;  import hexa.org.entity.\*;  import hexa.org.exception.\*;  public class EcomApp {  public static void main(String[] args) {  Scanner sc = new Scanner(System.in);  OrderProcessorRepository o = new OrderProcessorRepositoryImpl();    while (true) {  System.out.println("\n.......... E-Commerce System..........");  System.out.println("1. Register Customer");  System.out.println("2. Create Product");  System.out.println("3. Delete Product");  System.out.println("4. Add to Cart");  System.out.println("5. View Cart");  System.out.println("6. Place Order");  System.out.println("7. View Customer Orders");  System.out.println("0. Exit");  System.out.print("Enter your option: ");  int option = sc.nextInt();  sc.nextLine();  switch (option) {  case 1:  System.out.println("\n Register Customer");  System.out.println("Enter name: ");  String name = sc.nextLine();  System.out.println("Enter email: ");  String email = sc.nextLine();  System.out.println("Enter password: ");  String password = sc.nextLine();    Customer newCustomer = new Customer();  newCustomer.setName(name);  newCustomer.setEmail(email);  newCustomer.setPassword(password);    boolean isRegistered = o.createCustomer(newCustomer);    if(isRegistered) {  System.out.println("Customer registered successfully!");  }else {  System.out.println("Failed to register customer.");  }  break;    case 2:  System.out.println("\n Create Product");  System.out.print("Enter Product Name: ");  String pname = sc.nextLine();  System.out.print("Enter Price: ");  double price = sc.nextDouble();  sc.nextLine();  System.out.print("Enter Description: ");  String desc = sc.nextLine();  System.out.print("Enter Stock Quantity: ");  int stock = sc.nextInt();  sc.nextLine();    Product product = new Product();  product.setName(pname);  product.setPrice(price);  product.setDescription(desc);  product.setStockQuantity(stock);  boolean isProductCreated = o.createProduct(product);  if (isProductCreated) {  System.out.println("Product added successfully!");  } else {  System.out.println("Failed to add product.");  }  break;  case 3:  System.out.println("\n Delete Product");  System.out.print("Enter Product ID to delete: ");  int productIdToDelete = sc.nextInt();  sc.nextLine();  try {  boolean isDeleted = o.deleteProduct(productIdToDelete);  if (isDeleted) {  System.out.println("Product deleted successfully!");  } else {  System.out.println("Failed to delete product.");  }  } catch (ProductNotFoundException e) {  System.out.println(e.getMessage());  }  break;  case 4:  System.out.println("\n Add to Cart");  System.out.print("Enter Customer ID: ");  int customerIdToAdd = sc.nextInt();  System.out.print("Enter Product ID: ");  int productIdToAdd = sc.nextInt();  System.out.print("Enter Quantity: ");  int quantityToAdd = sc.nextInt();  sc.nextLine();  Customer customerToAdd = new Customer();  customerToAdd.setCustomerId(customerIdToAdd);  Product productToAdd = new Product();  productToAdd.setProductId(productIdToAdd);  boolean isAdded = o.addToCart(customerToAdd, productToAdd, quantityToAdd);  if (isAdded) {  System.out.println("Product added to cart successfully!");  } else {  System.out.println("Failed to add product to cart.");  }  break;  case 5:  System.out.println("\n View Cart");  System.out.print("Enter Customer ID to view cart: ");  int customerIdToView = sc.nextInt();  sc.nextLine();  Customer customerToView = new Customer();  customerToView.setCustomerId(customerIdToView);  try {  List<Product> cartProducts = o.getAllFromCart(customerToView);  if (cartProducts.isEmpty()) {  System.out.println("Cart is empty.");  } else {  System.out.println("\nProducts in your Cart:");  for (Product cartProduct : cartProducts) {  System.out.println("Product ID: " + cartProduct.getProductId() +  ", Name: " + cartProduct.getName() +  ", Price: " + cartProduct.getPrice() +  ", Stock Quantity: " + cartProduct.getStockQuantity());  }  }  } catch (CustomerNotFoundException e) {  System.out.println(e.getMessage());  }  break;  case 6:  System.out.println("\n Place Order");  System.out.print("Enter Customer ID: ");  int customerIdForOrder = sc.nextInt();  sc.nextLine();  Customer customerForOrder = new Customer();  customerForOrder.setCustomerId(customerIdForOrder);  List<Product> cartProductsForOrder = new ArrayList<>();  try {  cartProductsForOrder = o.getAllFromCart(customerForOrder);  } catch (CustomerNotFoundException e) {  System.out.println(e.getMessage());  break;  }  if (cartProductsForOrder.isEmpty()) {  System.out.println("Cart is empty, cannot place order.");  break;  }  System.out.println("Products in your cart:");  Map<Product, Integer> selectedProducts = new HashMap<>();  for (Product orderProduct : cartProductsForOrder) {  System.out.println("Product ID: " + orderProduct.getProductId() +  ", Name: " + orderProduct.getName() +  ", Price: " + orderProduct.getPrice());  }  System.out.print("Enter Shipping Address: ");  String shippingAddressForOrder = sc.nextLine();  for (Product orderProduct : cartProductsForOrder) {  System.out.print("Enter quantity for " + orderProduct.getName() + ": ");  int quantity = sc.nextInt();  sc.nextLine();  selectedProducts.put(orderProduct, quantity);  }  try {  boolean isOrderPlaced = o.placeOrder(customerForOrder, List.of(selectedProducts), shippingAddressForOrder);  if (isOrderPlaced) {  System.out.println("Order placed successfully!");  } else {  System.out.println("Failed to place order.");  }  } catch (CustomerNotFoundException | ProductNotFoundException e) {  System.out.println(e.getMessage());  }  break;    case 7:  System.out.println("\n View Customer Orders");  System.out.print("Enter Customer ID: ");  int customerIdForOrders = sc.nextInt();  sc.nextLine();  try {  List<Map<Product, Integer>> customerOrders = o.getOrdersByCustomer(customerIdForOrders);  if (customerOrders.isEmpty()) {  System.out.println("No orders found for this customer.");  } else {  System.out.println("\nOrders for Customer ID: " + customerIdForOrders);  for (Map<Product, Integer> order : customerOrders) {  for (Map.Entry<Product, Integer> entry : order.entrySet()) {  Product product1 = entry.getKey();  int quantity = entry.getValue();  System.out.println("Product: " + product1.getName() + ", Quantity: " + quantity);  }  }  }  } catch (CustomerNotFoundException e) {  System.out.println(e.getMessage());  } catch (OrderNotFoundException e) {  System.out.println(e.getMessage());  }  break;  case 0:  System.out.println("Exited...");  sc.close();  return;  default:  System.out.println("Invalid choice");  }  }  }  } |

1. **OUTPUT**

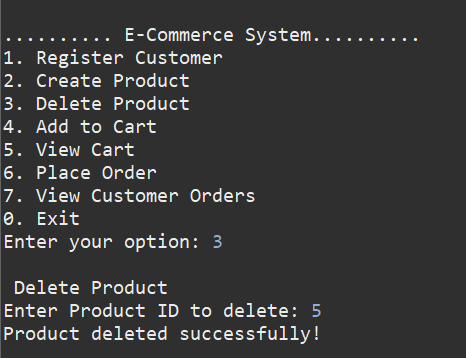
**When option is: 1**



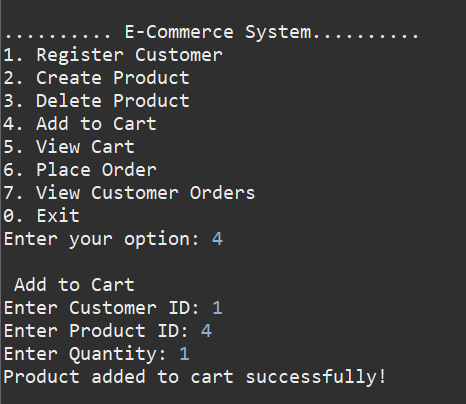
**When option is: 2**



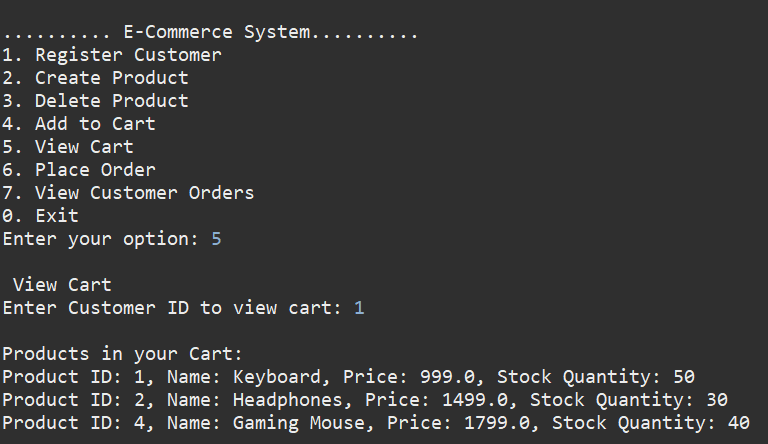
**When option is: 3**



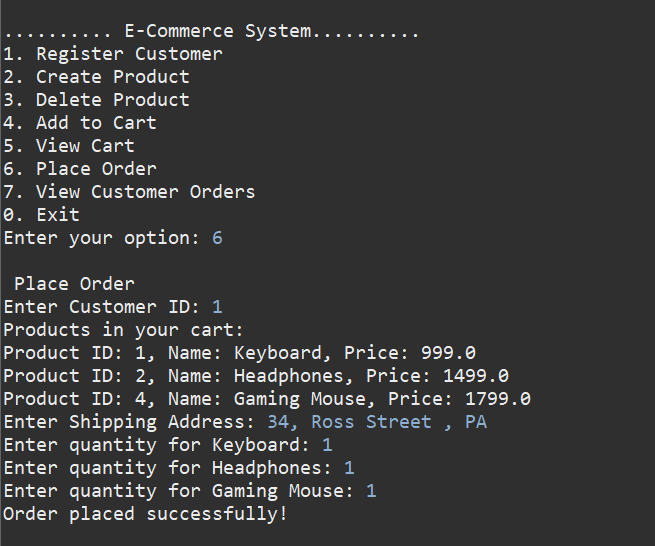
**When option is: 4**



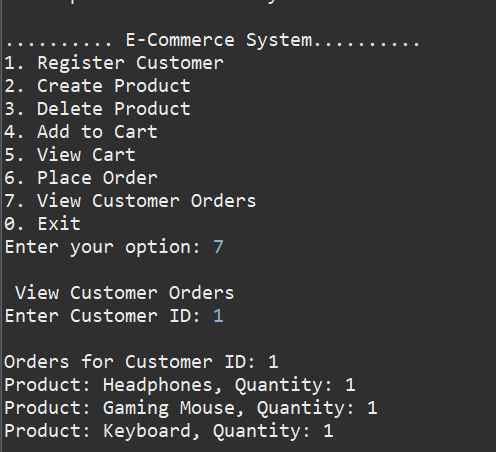
**When option is: 5**



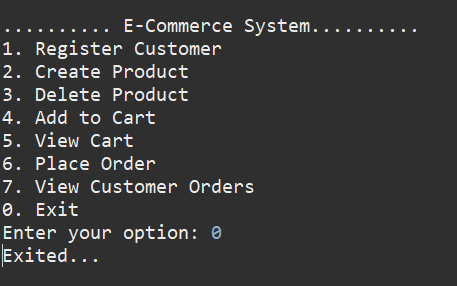
**When option is: 6**



**When option is: 7**



**When option is: 0**



**Unit Testing**

1. **Creating Unit test cases for Ecommerce System are essential to ensure the correctness and reliability of the system.**

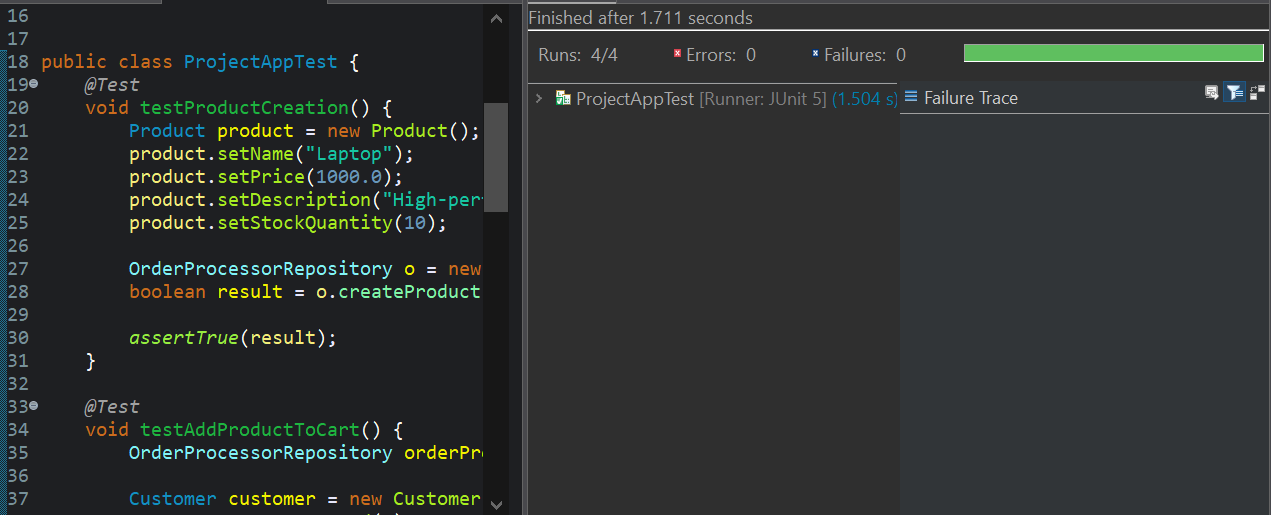
**• Write test case to test Product created successfully or not.**

**• Write test case to test product is added to cart successfully or not.**

**• Write test case to test product is ordered successfully or not.**

**• Write test case to test exception is thrown correctly or not when customer not found in database.**

|  |
| --- |
| package unitTest;  import org.junit.jupiter.api.Test;  import hexa.org.entity.Customer;  import hexa.org.entity.Product;  import hexa.org.exception.CustomerNotFoundException;  import hexa.org.exception.ProductNotFoundException;  import hexa.org.dao.OrderProcessorRepository;  import hexa.org.dao.OrderProcessorRepositoryImpl;  import static org.junit.jupiter.api.Assertions.\*;  import java.util.HashMap;  import java.util.List;  import java.util.Map;  public class ProjectAppTest {  @Test  void testProductCreation() {  Product product = new Product();  product.setName("Laptop");  product.setPrice(1000.0);  product.setDescription("High-performance laptop");  product.setStockQuantity(10);  OrderProcessorRepository o = new OrderProcessorRepositoryImpl();  boolean result = o.createProduct(product);  assertTrue(result);  }  @Test  void testAddProductToCart() {  OrderProcessorRepository orderProcessor = new OrderProcessorRepositoryImpl();  Customer customer = new Customer();  customer.setCustomerId(1);  Product product = new Product();  product.setProductId(2);  product.setName("Laptop");  product.setPrice(1000.0);  product.setDescription("High-performance laptop");  product.setStockQuantity(10);  boolean isAdded = orderProcessor.addToCart(customer, product, 2);  assertTrue(isAdded, "Product should be added to the cart successfully.");  }  @Test  void testPlaceOrder() throws CustomerNotFoundException, ProductNotFoundException {  OrderProcessorRepository orderProcessor = new OrderProcessorRepositoryImpl();  Customer customer = new Customer();  customer.setCustomerId(1);  Product product = new Product();  product.setProductId(2);  product.setName("Laptop");  product.setPrice(1000.0);  product.setDescription("High-performance laptop");  product.setStockQuantity(10);  Map<Product, Integer> orderProducts = new HashMap<>();  orderProducts.put(product, 2);  boolean isOrderPlaced = orderProcessor.placeOrder(customer, List.of(orderProducts), "123 Main Street");  assertTrue(isOrderPlaced, "Order should be placed successfully.");  }  @Test  void testCustomerNotFoundException() {  OrderProcessorRepository orderProcessor = new OrderProcessorRepositoryImpl();  Customer customer = new Customer();  customer.setCustomerId(9999);  try {  orderProcessor.getAllFromCart(customer);  fail("Expected CustomerNotFoundException to be thrown");  } catch (CustomerNotFoundException e) {  System.out.println("Exception thrown as expected: " + e.getMessage());  }  }  } |



GitHub Link: <https://github.com/sherinsandra03/Ecom>