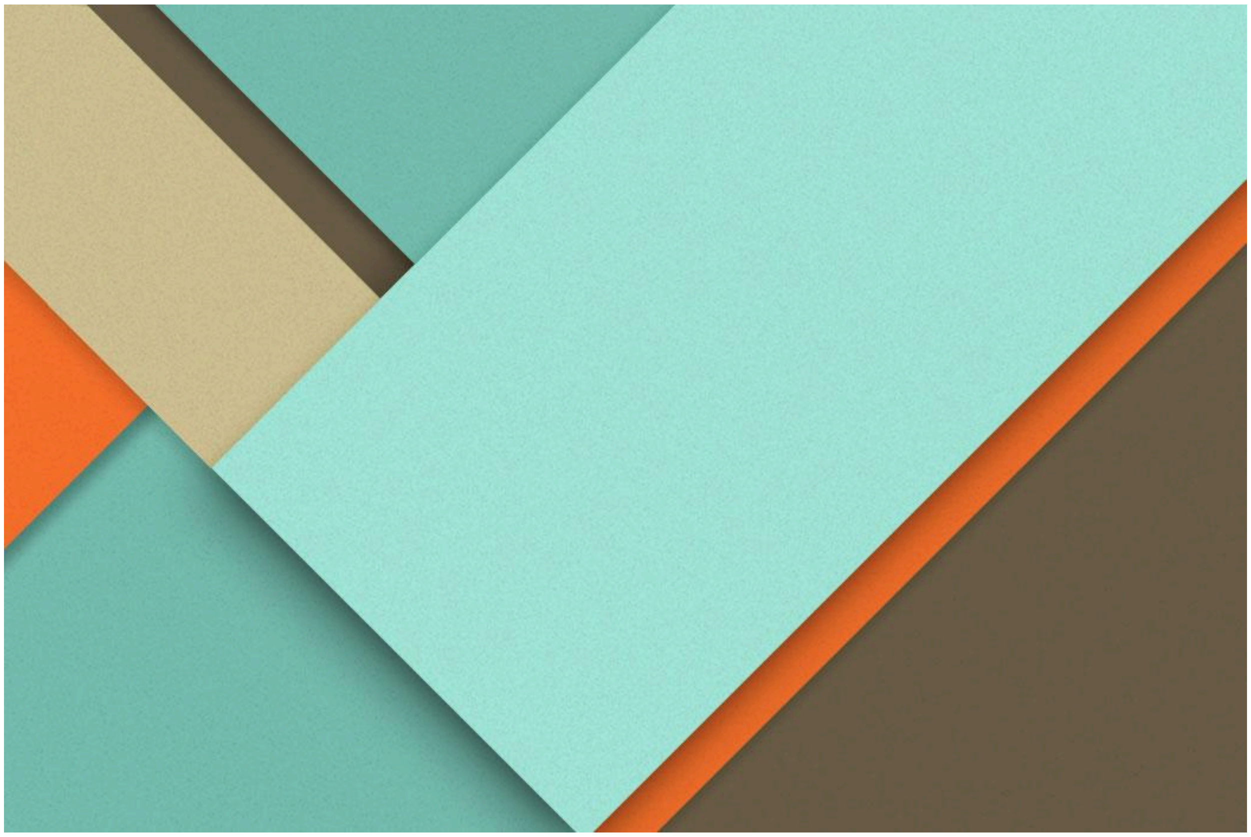


eCommerce Application Documentation

27-09-2024

—

Anjali Chauhan,
Mthree

Table of Contents

1. Introduction

The eCommerce application allows users to browse products, place orders, and manage inventory efficiently using a caching mechanism for performance optimization. It uses Java for the backend and MySQL as the database.

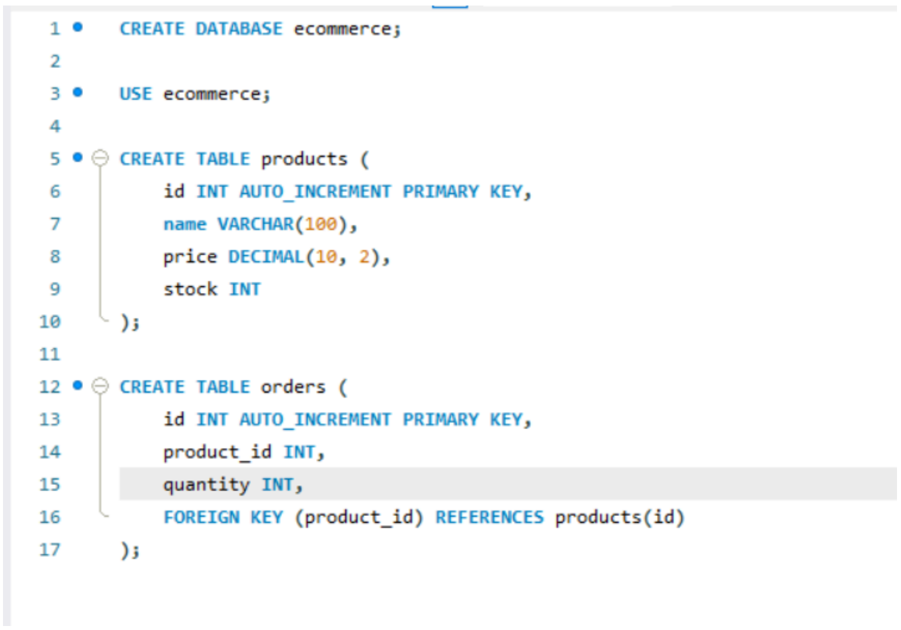
2. Architecture

The architecture of the eCommerce application is modular, consisting of the following components:

- **Database Layer**: Handles all database operations using JDBC.
- **Business Logic Layer**: Contains the main logic for retrieving products and placing orders.
- **Cache Layer**: Implements a simple in-memory cache for products to improve performance.
- **User Interface**: Provides a console-based interaction for users.

3. Database Schema

SQL Commands to Create Database and Tables



```sql
CREATE DATABASE ecommerce;

USE ecommerce;

CREATE TABLE products (
    id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(100),
    price DECIMAL(10, 2),
    stock INT
);

CREATE TABLE orders (
    id INT AUTO_INCREMENT PRIMARY KEY,
    product_id INT,
    quantity INT,
    FOREIGN KEY (product_id) REFERENCES products(id)
);
```

Table Descriptions

- **products**: Stores product details including ID, name, price, and stock quantity.
- **orders**: Stores order details including the product ID and quantity ordered.

4. Functionality

I. Core Features

View Products: Display a list of products available in the database.

Place Order: Allow users to place orders for specific products and update stock accordingly.

In-Memory Caching: Use a cache to store product information for quick retrieval, reducing database queries.

II. Workflow

The user runs the application and views the list of products.

The user enters a product ID and the desired quantity to place an order.

The application checks the cache for product details; if not found, it retrieves the data from the database.

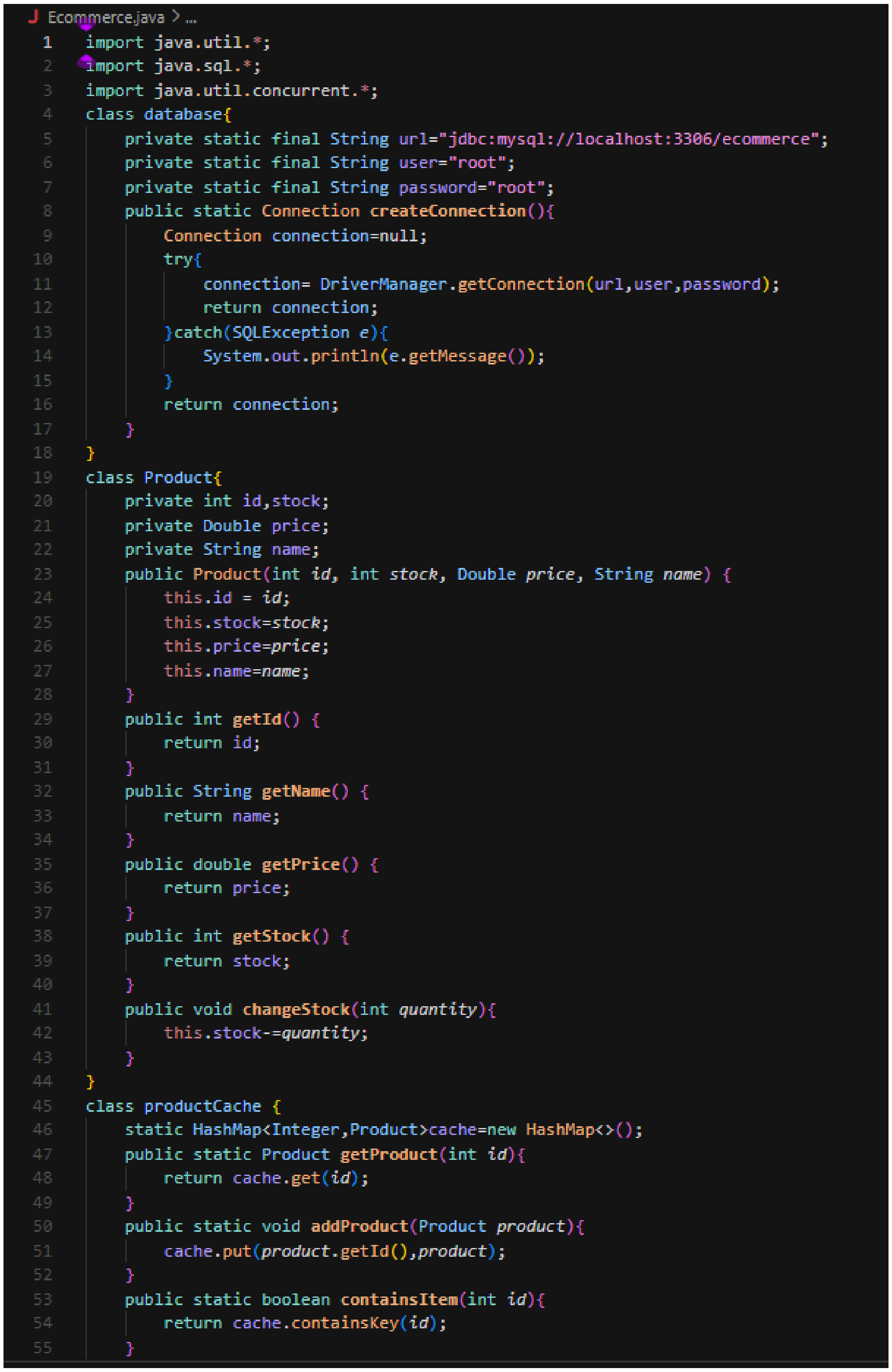
The application updates the stock in the database and confirms the order.

5. Code Structures

Classes Overview

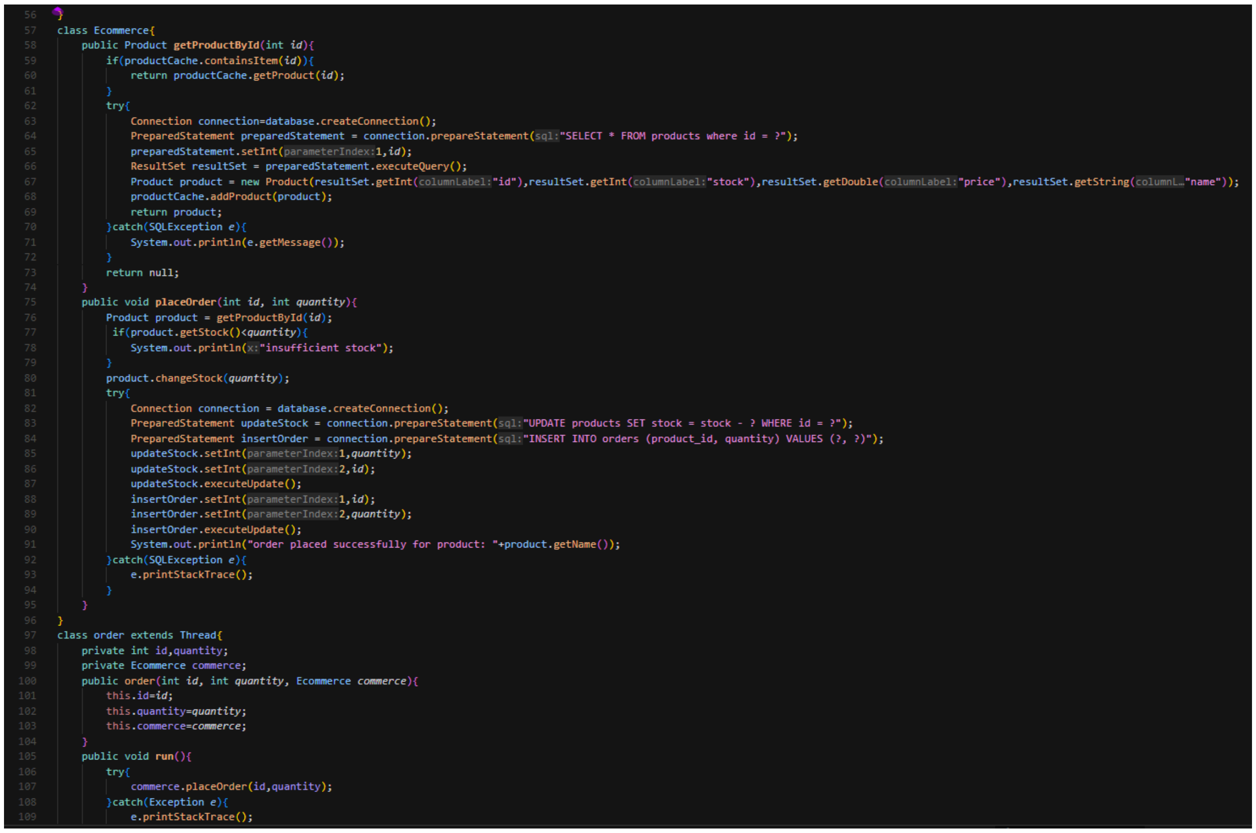
- **`database`**: Contains methods to establish a connection to the MySQL database.
- **`Product`**: Represents a product entity with attributes such as ID, stock, price, and name.
- **`productCache`**: Implements a simple caching mechanism for storing and retrieving products.
- **`Ecommerce`**: Main business logic class that handles product retrieval and order placement.
- **`order`**: Extends `Thread` to manage order placement concurrently.
- **`eCommerce`**: Main class containing the `main` method to execute the application.

Code Snippets

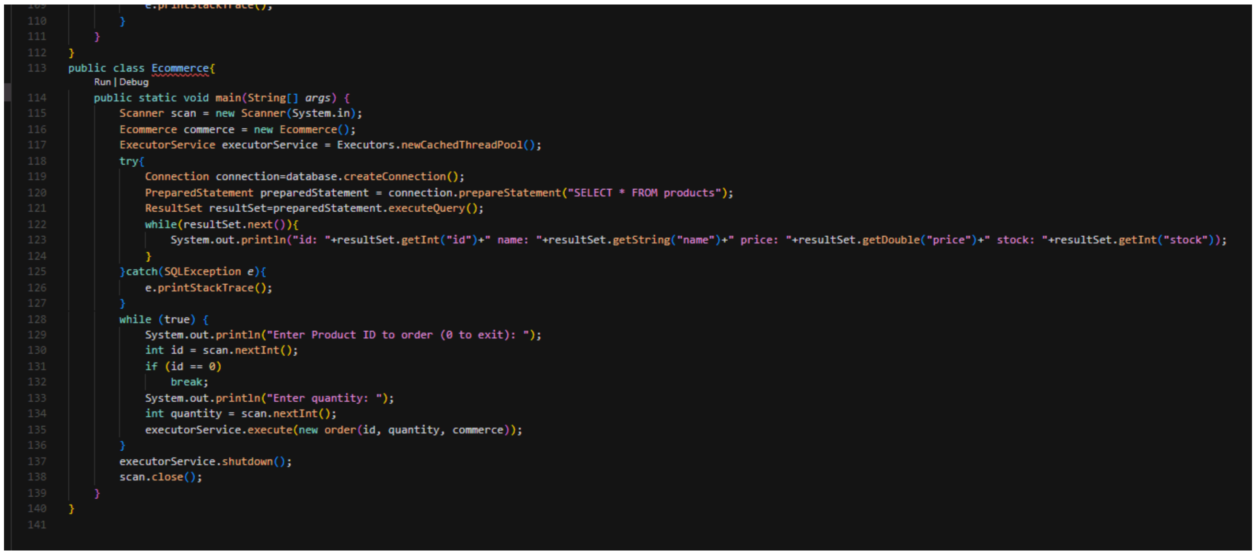


J Ecommerce.java > ...

```java
import java.util.*;
import java.sql.*;
import java.util.concurrent.*;
class database{
    private static final String url="jdbc:mysql://localhost:3306/ecommerce";
    private static final String user="root";
    private static final String password="root";
    public static Connection createConnection(){
        Connection connection=null;
        try{
            connection= DriverManager.getConnection(url,user,password);
            return connection;
        }catch(SQLException e){
            System.out.println(e.getMessage());
        }
        return connection;
    }
}
class Product{
    private int id,stock;
    private Double price;
    private String name;
    public Product(int id, int stock, Double price, String name) {
        this.id = id;
        this.stock=stock;
        this.price=price;
        this.name=name;
    }
    public int getId() {
        return id;
    }
    public String getName() {
        return name;
    }
    public double getPrice() {
        return price;
    }
    public int getStock() {
        return stock;
    }
    public void changeStock(int quantity){
        this.stock-=quantity;
    }
}
class productCache {
    static HashMap<Integer,Product>cache=new HashMap<>();
    public static Product getProduct(int id){
        return cache.get(id);
    }
    public static void addProduct(Product product){
        cache.put(product.getId(),product);
    }
    public static boolean containsItem(int id){
        return cache.containsKey(id);
    }
```



```java
}
class Ecommerce{
    public Product getProductById(int id){
        if(productCache.containsItem(id)){
            return productCache.getProduct(id);
        }
        try{
            Connection connection=database.createConnection();
            PreparedStatement preparedStatement = connection.prepareStatement(sql:"SELECT * FROM products where id = ?");
            preparedStatement.setInt(parameterIndex:1,id);
            ResultSet resultSet = preparedStatement.executeQuery();
            Product product = new Product(resultSet.getInt(columnLabel:"id"),resultSet.getInt(columnLabel:"stock"),resultSet.getDouble(columnLabel:"price"),resultSet.getString(columnL…"name"));
            productCache.addProduct(product);
            return product;
        }catch(SQLException e){
            System.out.println(e.getMessage());
        }
        return null;
    }
    public void placeOrder(int id, int quantity){
        Product product = getProductById(id);
         if(product.getStock()<quantity){
            System.out.println(x:"insufficient stock");
        }
        product.changeStock(quantity);
        try{
            Connection connection = database.createConnection();
            PreparedStatement updateStock = connection.prepareStatement(sql:"UPDATE products SET stock = stock - ? WHERE id = ?");
            PreparedStatement insertOrder = connection.prepareStatement(sql:"INSERT INTO orders (product_id, quantity) VALUES (?, ?)");
            updateStock.setInt(parameterIndex:1,quantity);
            updateStock.setInt(parameterIndex:2,id);
            updateStock.executeUpdate();
            insertOrder.setInt(parameterIndex:1,id);
            insertOrder.setInt(parameterIndex:2,quantity);
            insertOrder.executeUpdate();
            System.out.println("order placed successfully for product: "+product.getName());
        }catch(SQLException e){
            e.printStackTrace();
        }
    }
}
class order extends Thread{
    private int id,quantity;
    private Ecommerce commerce;
    public order(int id, int quantity, Ecommerce commerce){
        this.id=id;
        this.quantity=quantity;
        this.commerce=commerce;
    }
    public void run(){
        try{
            commerce.placeOrder(id,quantity);
        }catch(Exception e){
            e.printStackTrace();
```



```java
        }
    }
}
public class Ecommerce{
    Run | Debug
    public static void main(String[] args) {
        Scanner scan = new Scanner(System.in);
        Ecommerce commerce = new Ecommerce();
        ExecutorService executorService = Executors.newCachedThreadPool();
        try{
            Connection connection=database.createConnection();
            PreparedStatement preparedStatement = connection.prepareStatement("SELECT * FROM products");
            ResultSet resultSet=preparedStatement.executeQuery();
            while(resultSet.next()){
                System.out.println("id: "+resultSet.getInt("id")+" name: "+resultSet.getString("name")+" price: "+resultSet.getDouble("price")+" stock: "+resultSet.getInt("stock"));
            }
        }catch(SQLException e){
            e.printStackTrace();
        }
        while (true) {
            System.out.println("Enter Product ID to order (0 to exit): ");
            int id = scan.nextInt();
            if (id == 0)
                break;
            System.out.println("Enter quantity: ");
            int quantity = scan.nextInt();
            executorService.execute(new order(id, quantity, commerce));
        }
        executorService.shutdown();
        scan.close();
    }
}
```

6. Future Enhancements

Web Interface: Develop a web-based frontend using frameworks like Spring Boot or JavaServer Faces (JSF).

User Authentication: Implement user registration and login functionalities.

Advanced Inventory Management: Add features for product addition, deletion, and updating details.

Payment Gateway Integration: Allow users to make payments directly through the application.

7. Conclusion

The eCommerce application is a basic yet functional system demonstrating core concepts in Java programming, JDBC, caching mechanisms, and concurrency. The modular architecture enables easy expansion and integration of additional features.