**中北大学软件学院**

实训说明书

实训名称: 面向对象与数据库课程设计

题目名称：

三号楷体

专 业：

软件工程

班 级：

小组成员

学号： 三号楷体 姓名：三号楷体成绩：

学号： 姓名： 成绩：

学号： 姓名： 成绩：

学号： 姓名： 成绩：

学号： 姓名： 成绩：

学号： 姓名： 成绩：

指导教师：

三号楷体

2024 年 1 月

任务分工情况说明

|  |  |  |
| --- | --- | --- |
|  | 姓名 | 分工 |
| 组长 | 宋体小四 | 表格内填写字体为宋体小四 |
| 组员 |  |  |
| 组员 |  |  |
| 组员 |  |  |
| 组员 |  |  |
| 组员 |  |  |

目录

[1.引言 4](#_Toc155304483)

[1.1 编写目的 4](#_Toc155304484)

[1.2 项目说明 4](#_Toc155304485)

[1.3 项目背景 5](#_Toc155304486)

[1.4 项目定义 5](#_Toc155304487)

[2. 任务概述 5](#_Toc155304488)

[2.1 项目现状 5](#_Toc155304489)

[2.2 项目目标 5](#_Toc155304490)

[2.3 项目任务 5](#_Toc155304491)

[3. 需求分析 6](#_Toc155304492)

[3.1 用户需求分析 6](#_Toc155304493)

[3.2 业务需求分析 6](#_Toc155304494)

[3.3 功能需求分析 6](#_Toc155304495)

[4.详细设计 7](#_Toc155304496)

[4.1 数据连接层 7](#_Toc155304497)

[4.1.1 数据库连接配置 7](#_Toc155304498)

[4.1.2 单元工具类 10](#_Toc155304499)

[4.2 数据交互层 12](#_Toc155304500)

[4.2.1 用户数据交互 13](#_Toc155304501)

[4.2.2 商品数据交互 16](#_Toc155304502)

[4.3 业务逻辑层 20](#_Toc155304503)

[4.3.1 用户业务逻辑层 20](#_Toc155304504)

[4.3.2 商品业务逻辑层 23](#_Toc155304505)

[4.4 界面层 27](#_Toc155304506)

[4.4.1 界面层主要代码 27](#_Toc155304507)

[4.4.2 注册和登录界面 28](#_Toc155304508)

[4.4.3 普通用户界面 32](#_Toc155304509)

[4.4.4 管理员界面 40](#_Toc155304510)

1.引言

本文档旨在明确农副产品交易管理系统的需求，为系统的设计和开发提供准确的指导。农副产品交易管理系统是一个基于Java开发的数据库应用系统，旨在改善传统农副产品交易方式，提供一个便捷的平台，使用户能够方便地浏览、购买农副产品，管理员能够管理产品信息和用户信息。传统的农副产品交易过程存在信息不对称和交易效率低下的问题，通过开发农副产品交易管理系统，可以提高交易效率、降低信息不对称的风险，并提供良好的用户体验。

本项目的目标是设计和开发一个农副产品交易管理系统，实现农副产品的录入、修改、删除、查询和购买等功能，同时提供销量统计和用户管理功能。系统将采用图形用户界面，使用Java进行开发，数据将使用数据库进行持久存储。用户需求分析部分明确了用户对系统的需求和期望，包括方便浏览产品信息、快速找到感兴趣的产品以及修改个人信息和查看交易记录。业务需求分析部分明确了系统应支持农副产品品类和产品信息的录入、修改和删除操作，用户账户管理和个人信息修改功能，以及用户购买产品和管理员进行销量统计等业务需求。功能需求分析部分详细描述了农副产品管理、用户管理、产品浏览和购买、管理员管理、销量统计和查询等功能。

通过本文档的需求分析，可以为后续的系统设计和开发工作提供准确的指导。农副产品交易管理系统的开发将按照用户需求、业务需求和功能需求进行，确保系统具有良好的用户体验、高效的交易流程和可靠的数据管理。最终的系统将满足用户方便浏览和购买农副产品的需求，同时提供管理员对产品和用户的管理和销量统计功能。

1.1 编写目的

本文档旨在明确农副产品交易管理系统的需求，为系统的设计和开发提供准确的指导。

1.2 项目说明

农副产品交易管理系统是一个基于Java开发的数据库应用系统，旨在改善传统农副产品交易方式，提供一个便捷的平台，使用户能够方便地浏览、购买农副产品，管理员能够管理产品信息和用户信息。

1.3 项目背景

传统的农副产品交易过程存在信息不对称和交易效率低下的问题。通过开发农副产品交易管理系统，可以提高交易效率、降低信息不对称的风险，并提供良好的用户体验。

1.4 项目定义

农副产品交易管理系统旨在实现农副产品的录入、修改、删除、查询和购买等功能，同时提供销量统计和用户管理功能。系统将采用图形用户界面，使用Java进行开发，数据将使用数据库进行持久存储。

2. 任务概述

2.1 项目现状

目前，农副产品交易主要依赖传统的线下交易方式，存在信息不对称和交易效率低下的问题。用户和管理员需要花费大量时间和精力进行产品搜索、交易和管理。

2.2 项目目标

本项目的目标是设计和开发一个农副产品交易管理系统，提供用户友好的界面和便捷的操作方式，实现以下目标：

* 提供农副产品品类和产品信息的录入、修改、删除功能。
* 实现用户账户管理和个人信息修改功能。
* 提供产品浏览和购买功能，方便用户进行交易。
* 提供管理员账户管理和销量统计功能，方便管理员进行产品和用户管理。

2.3 项目任务

* 进行用户需求分析：明确用户对系统的需求和期望。
* 进行业务需求分析：了解农副产品交易的流程和规则，明确系统的业务需求。
* 进行功能需求分析：从用户和业务需求出发，明确系统的具体功能和操作方式。
* 进行数据库设计：设计合理的数据库结构，确保数据的存储和管理。
* 进行系统详细设计：包括界面设计和功能模块设计，确保系统的易用性和扩展性。
* 进行系统编码实现：按照软件工程的思想进行系统开发，保证代码的质量和可维护性。
* 进行系统测试和调试：确保系统的稳定性和功能完备性。

3. 需求分析

3.1 用户需求分析

* 用户希望能够方便地浏览农副产品的信息，包括品类、名称、价格等。
* 用户希望能够快速找到感兴趣的产品，并进行购买操作。
* 用户希望能够修改个人信息和查看交易记录。

3.2 业务需求分析

* 系统应支持农副产品品类和产品信息的录入、修改和删除操作，确保产品信息的准确性和及时性。
* 系统应支持用户账户管理和个人信息修改功能，保护用户隐私和权益。
* 系统应支持用户购买产品和管理员进行销量统计，提供交易的便利性和数据的分析性。
* 系统应支持按品类、名称或价格等进行查询，方便用户查找感兴趣的产品。

3.3 功能需求分析

1. 农副产品管理功能：
   1. 系统管理员能够录入、修改和删除农副产品的品类和产品信息。
2. 用户管理功能：
   1. 用户能够注册新账户，并提供必要的个人信息。
   2. 用户能够登录系统，并能够修改个人信息。
3. 产品浏览和购买功能：
   1. 用户能够浏览系统中的农副产品信息，包括品类、名称、价格等。
   2. 用户能够选择感兴趣的产品并进行购买操作。
4. 管理员管理功能：
   1. 管理员能够登录系统，并能够修改个人信息。
   2. 管理员能够管理用户信息，包括查看用户列表、修改用户信息和删除用户账户。
5. 销量统计功能：
   1. 管理员能够进行销量统计，包括按品类、产品名称等进行统计。
6. 查询功能：
   1. 系统应支持按品类、名称或价格等进行查询，方便用户查找感兴趣的产品。

4.详细设计

4.1 数据连接层

该层主要完成与数据库的连接以及数据库的配置和sql语句的执行

4.1.1 数据库连接配置

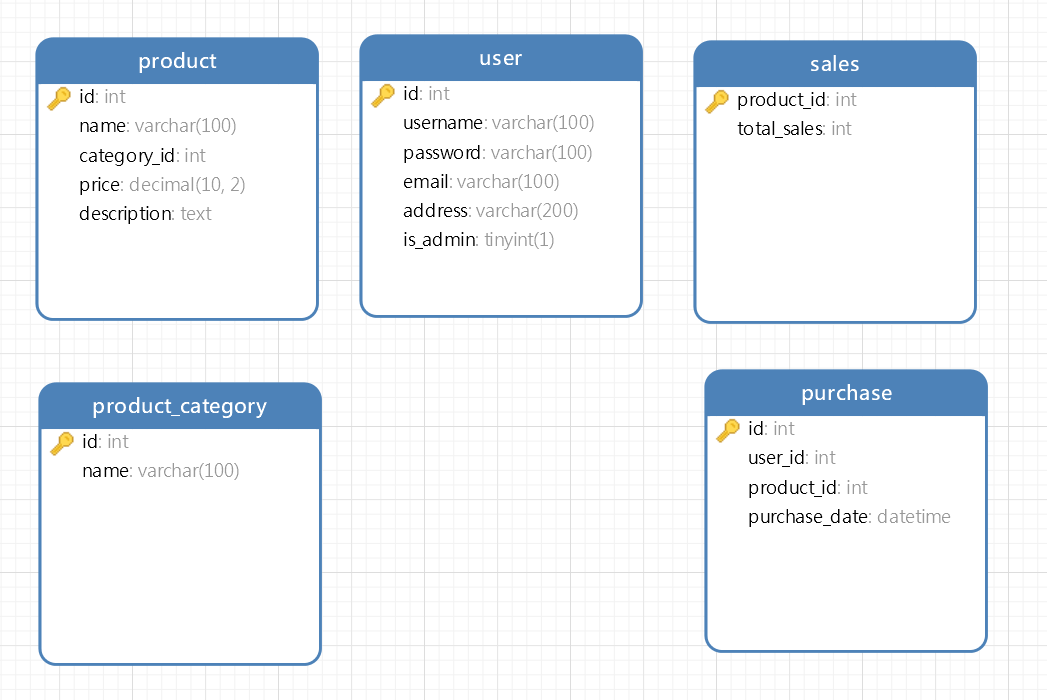
使用Mysql进行数据的持久化存储，共分为五个表，五个表分别存放了用户表、商品表、分类表、分类商品关系表、用户购买关系表，数据库结构如图4.1所示：

图 4.1 数据库结构图

详细sql语句如下：

*-- 创建数据库*DROP DATABASE IF EXISTS agricultural;  
CREATE DATABASE agricultural CHARACTER SET utf8mb4 COLLATE utf8mb4\_unicode\_ci;  
*-- 选择数据库*USE agricultural;  
*-- 创建表：农副产品品类*CREATE TABLE product\_category  
(  
 id INT PRIMARY KEY AUTO\_INCREMENT,  
 name VARCHAR(100) NOT NULL  
);  
  
*-- 创建表：农副产品信息*CREATE TABLE product  
(  
 id INT PRIMARY KEY AUTO\_INCREMENT,  
 name VARCHAR(100) NOT NULL,  
 category\_id INT,  
 price DECIMAL(10, 2),  
 description TEXT  
);  
  
*-- 创建表：用户*CREATE TABLE user  
(  
 id INT PRIMARY KEY AUTO\_INCREMENT,  
 username VARCHAR(100) NOT NULL,  
 password VARCHAR(100) NOT NULL,  
 email VARCHAR(100) NOT NULL,  
 address VARCHAR(200),  
 is\_admin BOOLEAN DEFAULT false  
);  
  
*-- 创建表：购买记录*CREATE TABLE purchase  
(  
 id INT PRIMARY KEY AUTO\_INCREMENT,  
 user\_id INT,  
 product\_id INT,  
 purchase\_date DATETIME  
);  
  
*-- 创建表：销量统计*CREATE TABLE sales  
(  
 product\_id INT PRIMARY KEY AUTO\_INCREMENT,  
 total\_sales INT  
);  
  
*-- 插入示例数据：农副产品品类*INSERT INTO product\_category (name)  
VALUES ('水果'),  
 ('蔬菜'),  
 ('畜禽产品');  
  
*-- 插入示例数据：农副产品信息*INSERT INTO product (name, category\_id, price, description)  
VALUES ('苹果', 1, 2.50, '新鲜红富士苹果'),  
 ('香蕉', 1, 1.80, '香甜可口的香蕉'),  
 ('西红柿', 2, 1.20, '新鲜番茄'),  
 ('黄瓜', 2, 0.80, '绿色有机黄瓜'),  
 ('鸡蛋', 3, 0.30, '农家散养鸡蛋'),  
 ('牛肉', 3, 8.50, '优质牛肉');  
  
*-- 插入示例数据：用户*INSERT INTO user (username, password, email, address, is\_admin)  
VALUES ('user1', 'password1', 'user1@example.com', '地址1', false),  
 ('user2', 'password2', 'user2@example.com', '地址2', false),  
 ('admin', 'adminpassword', 'admin@example.com', '管理员地址', true);  
  
*-- 插入示例数据：购买记录*INSERT INTO purchase (user\_id, product\_id, purchase\_date)  
VALUES (1, 1, '2023-12-27 23:45:37'),  
 (1, 2, '2023-12-27 23:45:37'),  
 (2, 3, '2023-12-27 23:45:37');  
  
*-- 插入示例数据：销量统计*INSERT INTO sales (total\_sales)  
VALUES (0),  
 (0),  
 (0),  
 (0),  
 (0),  
 (0);

为了方便数据库的配置，在类DatabaseConfig中使用静态常量分别定义了数据库链接地址、数据库用户名和数据库密码。

package com.agricultural.config;  
  
public class DatabaseConfig {  
 *// 数据库连接信息* public static final String *URL* = "jdbc:mysql://localhost:3306/agricultural";  
 *// 数据库用户名* public static final String *USERNAME* = "root";  
 *// 数据库密码* public static final String *PASSWORD* = "root";  
}

4.1.2 单元工具类

为了方便数据库的链接和sql的执行，我们编写了类DatabaseUtil，在类DatabaseUtil中我们定义了四个执行自定义sql的静态方法，在每次执行都可以返回相应的ResultSet内容，DatabaseUtil的核心代码如下：

package com.agricultural.util;  
import com.agricultural.config.DatabaseConfig;  
import java.sql.\*;  
public class DatabaseUtil {  
 private static Connection *cn* = null;  
 static {  
 try {  
 *cn* = DriverManager.*getConnection*(DatabaseConfig.*URL*, DatabaseConfig.*USERNAME*, DatabaseConfig.*PASSWORD*);  
 } catch (Exception e) {  
 System.*out*.println("数据库链接失败，请检查数据库链接配置");  
 }  
 }  
  
 */\*\*  
 \* 获取数据库连接池  
 \*  
 \* @return 数据库连接  
 \*/* public static Connection classDbInit() {  
 return *cn*;  
 }  
  
 */\*\*  
 \* 执行查询sql语句  
 \*  
 \* @param sql 执行的sql语句  
 \* @param params 传入的参数  
 \* @return 执行结果  
 \*/* public static ResultSet executeQuery(String sql, Object... params) {  
 PreparedStatement pstm = null;  
 try {  
 pstm = *cn*.prepareStatement(sql);  
 *// 设置占位符参数* for (int i = 0; i < params.length; i++) {  
 pstm.setObject(i + 1, params[i]);  
 }  
 pstm.executeQuery();  
 *// 执行查询* return pstm.executeQuery();  
 } catch (Exception e) {  
 e.printStackTrace();  
 return null; *// 或者根据需要抛出自定义异常* }  
 }  
  
 */\*\*  
 \* 执行更新sql语句  
 \*  
 \* @param sql 传入的sql语句  
 \* @param params 传入的参数  
 \* @return 执行结果  
 \*/* public static Integer executeUpdate(String sql, Object... params) {  
 PreparedStatement pstm = null;  
 try {  
 pstm = *cn*.prepareStatement(sql);  
 for (int i = 0; i < params.length; i++) {  
 pstm.setObject(i + 1, params[i]);  
 }  
 return pstm.executeUpdate();  
 } catch (Exception e) {  
 e.printStackTrace();  
 return -1;  
 } finally {  
 try {  
 if (pstm != null) {  
 pstm.close();  
 }  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
 }  
 }  
  
 */\*\*  
 \* 执行单句sql  
 \*  
 \* @param sql 传入的sql语句  
 \* @return 执行结果  
 \*/* public static ResultSet singleSentenceSql(String sql) {  
 Statement stm = null;  
 try {  
 stm = *cn*.createStatement();  
 return stm.executeQuery(sql);  
 } catch (Exception e) {  
 e.printStackTrace();  
 return null;  
 }  
 }  
  
 */\*\*  
 \* 关闭数据库链接  
 \*/* public static void closeDb() {  
 try {  
 *cn*.close();  
 } catch (SQLException e) {  
 throw new RuntimeException(e);  
 }  
 }  
}

4.2 数据交互层

该层主要实现了代码和数据库的交互，主要完成对数据库的读写操作。

4.2.1 用户数据交互

* 添加用户

public static int addUser(User usr);

该方法通过传入一个User对象来进行对user表添加用户的操作，返回一个int类型的值来显示在数据库中影响的函数以便判断插入用户是否成功。

* 删除用户

public static int deleteUser(int id);

通过传入用户的id来对user表进行删除的操作，返回int类型的值来显示在数据库中影响的行数以便判断删除用户是否成功。

* 修改用户

public static int updateUser(User usr);

通过传入用户对象来对user表进行编辑的操作，返回int类型的值来显示在数据库中影响的行数以便判断修改用户是否成功。

* 查询用户

public static ResultSet getUser(int id);

通过传入用户id来对user表进行查询的操作，返回ResultSet结果集。

* 获取用户列表

public static ResultSet getUserList();

使用此方法可以返回包含所有用户信息的Result结果集。

* 用户登录

public static ResultSet login(String username, String password);

通过传入用户名和密码来实现登录，返回登录操作数据库的结果集。

* 检查用户名是否存在

public static ResultSet checkUser(String username);

传入用户名，检查数据库中是否存在相同用户名，并返回查询操作数据库的结果集。

* 类UserDB的核心代码如下：

package com.agricultural.dbchange;  
import com.agricultural.bean.User;  
import com.agricultural.util.DatabaseUtil;  
import java.sql.ResultSet;  
public class UserDB {  
 */\*\*  
 \* 新增用户  
 \*  
 \* @param usr 用户  
 \* @return int 影响行数  
 \*/* public static int addUser(User usr) {  
 String sql = "insert into user (username, password, email, address, is\_admin) values (?, ?, ?, ?, ?)";  
 return DatabaseUtil.*executeUpdate*(sql, usr.getUsername(), usr.getPassword(), usr.getEmail(), usr.getAddress(), usr.getIs\_admin());  
 }  
  
 */\*\*  
 \* 删除用户  
 \*  
 \* @param id 用户id  
 \* @return int 影响行数  
 \*/* public static int deleteUser(int id) {  
 String sql = "delete from user where id = ?";  
 return DatabaseUtil.*executeUpdate*(sql, id);  
 }  
  
 */\*\*  
 \* 修改用户  
 \*  
 \* @param usr 用户  
 \* @return int 影响行数  
 \*/* public static int updateUser(User usr) {  
 String sql = "update user set username = ?, password = ?, email = ?, address = ?, is\_admin = ? where id = ?";  
 return DatabaseUtil.*executeUpdate*(sql, usr.getUsername(), usr.getPassword(), usr.getEmail(), usr.getAddress(), usr.getIs\_admin(), usr.getId());  
 }  
  
 */\*\*  
 \* 查询用户  
 \*  
 \* @param id 用户id  
 \* @return ResultSet 查询结果  
 \*/* public static ResultSet getUser(int id) {  
 String sql = "select \* from user where id = ?";  
 return DatabaseUtil.*executeQuery*(sql, id);  
 }  
  
 */\*\*  
 \* 获取用户列表  
 \* @return ResultSet 查询结果  
 \*/* public static ResultSet getUserList() {  
 String sql = "select \* from user";  
 return DatabaseUtil.*executeQuery*(sql);  
 }  
  
 */\*\*  
 \* 用户登录方法  
 \* @param username 用户名  
 \* @param password 密码  
 \* @return ResultSet 查询结果，包括用户id，用户名，密码，邮箱，地址，是否为管理R  
 \*/* public static ResultSet login(String username, String password) {  
 String sql = "select \* from user where username = ? and password = ?";  
 return DatabaseUtil.*executeQuery*(sql, username, password);  
 }  
  
 */\*\*  
 \* 检查用户名是否存在  
 \* @param username 用户名  
 \* @return ResultSet 查询结果  
 \*/* public static ResultSet checkUser(String username){  
 String sql = "select \* from user where username = ?";  
 return DatabaseUtil.*executeQuery*(sql,username);  
 }  
}

4.2.2 商品数据交互

* 添加商品

public static int addProduct(Product product);

该方法通过传入一个Product对象来进行对product表添加商品的操作，返回一个int类型的值来显示在数据库中影响的函数以便判断添加商品是否成功。

* 删除商品

public static int deleteProduct (int id);

通过传入商品的id来对product表进行删除的操作，返回int类型的值来显示在数据库中影响的行数以便判断删除商品是否成功。

* 修改商品

public static int updateProduct (User usr);

通过传入商品对象来对product表进行编辑的操作，返回int类型的值来显示在数据库中影响的行数以便判断修改商品是否成功。

* 查询商品

public static ResultSet getProduct (int id);

通过传入商品id来对product表进行查询的操作，返回ResultSet结果集。

* 获取商品列表

public static ResultSet getProductList();

调用此方法可以返回一个包含所有商品的结果集。

* 用户购买商品

public static int buyProduct(int userId,int productId);

通过传入用户id和所购买的商品id操作sales表和purchase表进行销量统计和购买操作，返回int类型操作数据库影响的行数。

* 查询某个用户的购买记录

public static ResultSet getPurchaseListByUserId(int userId);

通过传入用户id来查询传入的用户id下购买了哪些商品，返回一个包含用户购买信息的结果集。

* 查询某个分类下的商品

public static ResultSet searchByCategory(int categoryId);

传入分类id，返回包含此分类下的所有商品的结果集。

* 通过商品名称搜索

public static ResultSet searchByName(String name);

传入搜索的商品名称，返回包含符合条件的商品的结果集。

* 通过价格筛选商品

public static ResultSet searchByPrice(double price);

传入double类型的价格，返回包含小于传入值的商品信息的结果集。

* 类ProductDB的核心代码如下：

package com.agricultural.dbchange;  
import com.agricultural.bean.Product;  
import com.agricultural.util.DatabaseUtil;  
import java.sql.ResultSet;  
public class ProductDB {  
 */\*\*  
 \* 新增产品信息  
 \*  
 \* @param product 产品对象  
 \* @return int 查询影响的行数  
 \*/* public static int addProduct(Product product) {  
 String sql = "INSERT INTO product (name, category\_id, price, description) VALUES (?, ?, ?, ?)";  
 SalesDB.*initSales*(product.getId());  
 return DatabaseUtil.*executeUpdate*(sql, product.getName(), product.getCategoryId(), product.getPrice(), product.getDescription());  
 }  
 */\*\*  
 \* 删除产品信息  
 \*  
 \* @param id 产品id  
 \* @return int 查询影响的行数  
 \*/* public static int deleteProduct(int id) {  
 String sql = "DELETE FROM product WHERE id = ?";  
 SalesDB.*deleteSales*(id);  
 return DatabaseUtil.*executeUpdate*(sql, id);  
 }  
  
 */\*\*  
 \* 修改产品信息  
 \*  
 \* @param product 产品对象  
 \* @return int 查询影响的行数  
 \*/* public static int updateProduct(Product product) {  
 String sql = "UPDATE product SET name = ?, category\_id = ?, price = ?, description = ? WHERE id = ?";  
 return DatabaseUtil.*executeUpdate*(sql, product.getName(), product.getCategoryId(), product.getPrice(), product.getDescription(), product.getId());  
 }  
  
 */\*\*  
 \* 查询产品信息  
 \*  
 \* @param id 产品id  
 \* @return ResultSet 查询结果，包含产品信息、分类信息等多个表  
 \*/* public static ResultSet getProduct(int id) {  
 String sql = "SELECT \* FROM product WHERE id = ?";  
 return DatabaseUtil.*executeQuery*(sql, id);  
 }  
  
 */\*\*  
 \* 查询产品列表  
 \*  
 \* @return ResultSet 查询结果  
 \*/* public static ResultSet getProductList() {  
 String sql = "SELECT \* FROM product";  
 return DatabaseUtil.*executeQuery*(sql);  
 }  
  
 */\*\*  
 \* 根据分类id获取某个分类的产品  
 \*  
 \* @param categoryId 分类id  
 \* @return ResultSet 查询结果  
 \*/* public static ResultSet getProductListByCategoryId(int categoryId) {  
 String sql = "SELECT \* FROM product WHERE category\_id = ?";  
 return DatabaseUtil.*executeQuery*(sql, categoryId);  
 }  
  
 */\*\*  
 \* 通过用户id和产品id购买商品，添加商品记录  
 \* @param userId 用户id  
 \* @param productId 产品id  
 \* @return int 影响的行数  
 \*/* public static int buyProduct(int userId,int productId) {  
 String sql = "INSERT INTO purchase (user\_id, product\_id, purchase\_date) VALUES (?, ?, CURRENT\_TIMESTAMP)";  
 *upSale*(productId);  
 return DatabaseUtil.*executeUpdate*(sql, userId, productId);  
 }  
  
 */\*\*  
 \* 传入产品id使产品销量+1  
 \* @param productId 产品id  
 \*/* public static void upSale(int productId){  
 String sql = "UPDATE sales SET total\_sales = total\_sales + 1 WHERE product\_id = ?";  
 DatabaseUtil.*executeUpdate*(sql,productId);  
 }  
  
 */\*\*  
 \* 查询某个用户的购买记录  
 \* @param userId 用户id  
 \* @return ResultSet 查询结果，包含用户id，购买日期，产品id，产品name  
 \*/* public static ResultSet getPurchaseListByUserId(int userId) {  
 String sql = "SELECT p.id, p.purchase\_date, pr.name, pr.price FROM purchase p JOIN product pr ON p.product\_id = pr.id WHERE p.user\_id = ?";  
 return DatabaseUtil.*executeQuery*(sql, userId);  
 }  
  
 */\*\*  
 \* 通过分类查id询产品  
 \* @param categoryId 分类id  
 \* @return ResultSet 查询结果，包含产品id，产品name  
 \*/* public static ResultSet searchByCategory(int categoryId){  
 String sql = "SELECT \* FROM product WHERE category\_id = ?";  
 return DatabaseUtil.*executeQuery*(sql,categoryId);  
 }  
  
 */\*\*  
 \* 通过名称查询产品  
 \* @param name 需要查询的名称  
 \* @return ResultSet 查询结果，包含产品id，产品name  
 \*/* public static ResultSet searchByName(String name){  
 name = "%" + name + "%";  
 String sql = "SELECT \* FROM product WHERE name LIKE ?";  
 return DatabaseUtil.*executeQuery*(sql,name);  
 }  
  
 */\*\*  
 \* 通过价格查询产品，查询低于传入价格的产品  
 \* @param price 产品价格  
 \* @return ResultSet 查询结果，包含产品id，产品name  
 \*/* public static ResultSet searchByPrice(double price){  
 String sql = "SELECT \* FROM product WHERE price <= ?";  
 return DatabaseUtil.*executeQuery*(sql,price);  
 }  
}

4.3 业务逻辑层

该层主要实现对数据交互层的进一步封装，使其返回所需要的类型。

4.3.1 用户业务逻辑层

* 登录方法

public static User login(String username, String password);

传入用户名和密码，通过和数据交互层的操作将结果集封装为返回登录的用户对象。

* 获取用户列表

public static List<User> getUserList();

返回通过数据交互层查询的用户列表，并将其封装为包含User类型的列表。

* 检查用户名是否符合要求

public static boolean checkUser(String username);

传入用户名，并通过数据交互层的查询来检查用户名是否存在，检查存在返回true，不存在返回false。

* 获取用户对象

public static User getUser(int id);

封装数据交互层返回的结果集为User对象。

* 类UserService的核心代码如下：

package com.agricultural.service;  
import com.agricultural.bean.User;  
import com.agricultural.dbchange.UserDB;  
import com.agricultural.util.DatabaseUtil;  
import java.sql.ResultSet;  
import java.util.ArrayList;  
import java.util.List;  
public class UserService {  
 */\*\*  
 \* 用户登录方法  
 \*  
 \* @param username 用户名  
 \* @param password 密码  
 \* @return User 用户  
 \*/* public static User login(String username, String password) {  
 try (ResultSet resultSet = UserDB.*login*(username,password)) {  
 if (resultSet != null && resultSet.next()) {  
 return new User(resultSet.getInt(1), resultSet.getString(2), resultSet.getString(3), resultSet.getString(4), resultSet.getString(5), resultSet.getString(6));  
 } else {  
 return null;  
 }  
 } catch (Exception e) {  
 e.printStackTrace();  
 return null;  
 }  
 }  
  
 */\*\*  
 \* 获取用户列表  
 \*  
 \* @return List 用户列表  
 \*/* public static List<User> getUserList() {  
 try (ResultSet userList = UserDB.*getUserList*()) {  
 List<User> users = new ArrayList<>();  
 while (userList.next()) {  
 User user = new User();  
 user.setId(userList.getInt(1));  
 user.setUsername(userList.getString(2));  
 user.setPassword(userList.getString(3));  
 user.setEmail(userList.getString(4));  
 user.setAddress(userList.getString(5));  
 user.setIs\_admin(userList.getString(6));  
 users.add(user);  
 }  
 return users;  
 } catch (Exception e) {  
 e.printStackTrace();  
 }  
 return null;  
 }  
  
 */\*\*  
 \* 查询用户名是否存在  
 \*  
 \* @param username 用户名  
 \* @return boolean 是否存在  
 \*/* public static boolean checkUser(String username) {  
 try (ResultSet resultSet = UserDB.*checkUser*(username)) {  
 return resultSet != null && resultSet.next();  
 } catch (Exception e) {  
 e.printStackTrace();  
 return false;  
 }  
 }  
  
 */\*\*  
 \* 查询用户  
 \*  
 \* @param id 用户id  
 \*/* public static User getUser(int id) {  
 try (ResultSet resultSet = UserDB.*getUser*(id)) {  
 if (resultSet != null && resultSet.next()) {  
 return new User(resultSet.getInt(1), resultSet.getString(2), resultSet.getString(3), resultSet.getString(4), resultSet.getString(5), resultSet.getString(6));  
 } else {  
 return null;  
 }  
 } catch (Exception e) {  
 e.printStackTrace();  
 return null;  
 }  
 }  
}

4.3.2 商品业务逻辑层

* 获取产品列表

public static List<Product> getProductList();

将数据交互层的结果集转换为包含Product类型的列表。

* 用户购买商品

public static int buyProduct(int userId, int productId);

传入用户id和商品id进行商品购买，返回操作影响的行数。

* 查询商品信息

public static Product getProduct(int id);

传入商品id，返回商品对象。

* 通过分类查询商品

public static List<Product> searchByCategory(int categoryId);

传入分类id，返回分类中包含的商品列表。

* 通过名称查询商品

public static List<Product> searchByName(String name);

传入搜索的商品名称，返回符合条件的商品列表。

* 通过价格查询商品

public static List<Product> searchByPrice(double price);

传入商品价格，返回低于传入商品价格的商品列表。

* 类ProductService的核心代码如下：

package com.agricultural.service;  
import com.agricultural.bean.Product;  
import com.agricultural.dbchange.ProductDB;  
import java.sql.ResultSet;  
import java.sql.SQLException;  
import java.util.ArrayList;  
import java.util.List;  
public class ProductService {  
 */\*\*  
 \* 获取产品列表  
 \*  
 \* @return List 产品列表  
 \*/* public static List<Product> getProductList() {  
 try (ResultSet resultSet = ProductDB.*getProductList*()) {  
 return *getProductsList*(resultSet);  
 } catch (Exception e) {  
 e.printStackTrace();  
 return null;  
 }  
 }  
 */\*\*  
 \* 通过用户id和产品id购买商品，添加商品记录  
 \* @param userId 用户id  
 \* @param productId 产品id  
 \* @return int 影响的行数  
 \*/* public static int buyProduct(int userId, int productId) {  
 return ProductDB.*buyProduct*(userId, productId);  
 }  
  
 */\*\*  
 \* 查询产品信息  
 \*  
 \* @param id 产品id  
 \* @return Product 产品对象  
 \*/* public static Product getProduct(int id) {  
 try (ResultSet resultSet = ProductDB.*getProduct*(id)) {  
 if (resultSet != null && resultSet.next()) {  
 return new Product(resultSet.getInt(1), resultSet.getString(2), resultSet.getInt(3), resultSet.getDouble(4), resultSet.getString(5));  
 } else {  
 return null;  
 }  
 } catch (Exception e) {  
 e.printStackTrace();  
 return null;  
 }  
 }  
  
 */\*\*  
 \* 通过分类查询产品  
 \*  
 \* @param categoryId 分类id  
 \* @return List 产品列表  
 \*/* public static List<Product> searchByCategory(int categoryId) {  
 try (ResultSet resultSet = ProductDB.*searchByCategory*(categoryId)) {  
 return *getProductsList*(resultSet);  
 } catch (Exception e) {  
 e.printStackTrace();  
 return null;  
 }  
 }  
  
 */\*\*  
 \* 通过名称查询产品  
 \*  
 \* @param name 产品名称  
 \* @return List 产品列表  
 \*/* public static List<Product> searchByName(String name) {  
 try (ResultSet resultSet = ProductDB.*searchByName*(name)) {  
 return *getProductsList*(resultSet);  
 } catch (Exception e) {  
 e.printStackTrace();  
 return null;  
 }  
 }  
  
 */\*\*  
 \* 提取resultSet中的List  
 \* @param resultSet 查询  
 \* @return List  
 \* @throws SQLException 抛出异常  
 \*/* private static List<Product> getProductsList(ResultSet resultSet) throws SQLException {  
 List<Product> products = new ArrayList<>();  
 while (resultSet.next()) {  
 Product product = new Product();  
 product.setId(resultSet.getInt(1));  
 product.setName(resultSet.getString(2));  
 product.setCategoryId(resultSet.getInt(3));  
 product.setPrice(resultSet.getDouble(4));  
 product.setDescription(resultSet.getString(5));  
 products.add(product);  
 }  
 return products;  
 }  
  
 */\*\*  
 \* 通过价格查询产品  
 \*  
 \* @param price 产品价格  
 \* @return List 产品列表  
 \*/* public static List<Product> searchByPrice(double price) {  
 try (ResultSet resultSet = ProductDB.*searchByPrice*(price)) {  
 return *getProductsList*(resultSet);  
 } catch (Exception e) {  
 e.printStackTrace();  
 return null;  
 }  
 }  
}

4.4 界面层

该层主要对数据进行一个可视化操作，使用户对数据交互层的逻辑清晰可见。

4.4.1 界面层主要代码

界面层主要分为四个模块，即：

* 成员变量：包含各种组件的属性设置和某些成员变量。
* 构造函数：在类被初始化时执行的方法。

构造函数的主要代码，以类ChangeProduct为例:

public ChangeProduct(Product changeProduct) {  
 this.product = changeProduct;  
 initCategoryList();  
 if (product != null) {  
 isChange = true;  
 inName.setText(product.getName());  
 inPrice.setText(String.*valueOf*(product.getPrice()));  
 inDescription.setText(product.getDescription());  
 inCategory.setSelectedIndex(product.getCategoryId() - 1);  
 } else {  
 isChange = false;  
 this.product = new Product();  
 }  
 initializeComponents();  
 submit.addActionListener(new ActionListener() {  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 *// 判断各个输入框是否为空* if (inName.getText().isEmpty() || inPrice.getText().isEmpty() || inDescription.getText().isEmpty()) {  
 JOptionPane.*showMessageDialog*(null, "请填写完整信息");  
 return;  
 }  
 product.setName(inName.getText());  
 product.setPrice(Double.*parseDouble*(inPrice.getText()));  
 product.setDescription(inDescription.getText());  
 product.setCategoryId(CategoryDB.*fromCategoryGetId*((String) inCategory.getSelectedItem()));  
 if (isChange) {  
 changeProduct();  
 } else {  
 addProduct();  
 }  
 }  
 });  
}

* 初始化方法：在构造函数中被调用，进行组件的初始化和事件设置。

初始化方法的主要代码：

private void initializeComponents() {  
 frame = new JFrame("ChangeProduct");  
 frame.setSize(500, 400);  
 frame.setLocationRelativeTo(null);  
 frame.setResizable(false);  
 frame.setContentPane(main);  
 frame.setDefaultCloseOperation(JFrame.*DISPOSE\_ON\_CLOSE*);  
 frame.setVisible(true);  
}

* UI生成器生成的代码：主要由form文件转换为可编译的java代码。

通过对各个模块的封装使其实现低耦合、高内聚的特点。

4.4.2 注册和登录界面

该页面包含三个窗体类：Login和SignIn运行结果如图4.2.，4.5。

**登录**

1. 用户在进行系统登录时，需要通过该界面输入帐号和密码，然后点击“登录”按钮进行登录，点击重置按钮可重置输入框中的内容，登录界面设计如图4.2所示。

图 4.2 登录界面设计

1. 用户在点击“登录”按钮之前必须输入用户名和密码，否则系统将会提示，提示信息如图4.3所示。

图 4.3 提示“账号或密码错误”

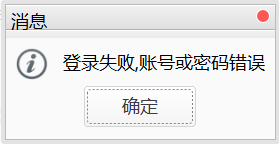
1. 获取用户在登录窗体输入的用户名和密码，将信息通过数据交互层同数据库交互，如果用户输入的用户名和密码可以和数据库中的用户名和密码相对应，界面就会根据身份跳转到主页面，并提示“登录成功”，如图4.4所示。

图 4.4 提示“登录成功”

1. 登录按钮事件处理的核心代码

loginButton.addActionListener(new ActionListener() {  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 *// 获取输入的用户名和密码* String username = inUsername.getText();  
 char[] pwd = inPassword.getPassword();  
 String password = new String(pwd);  
 *user* = UserService.*login*(username, password);  
 *//判断登录逻辑* if (*user* != null) {  
 JOptionPane.*showMessageDialog*(null, "登录成功");  
 *// 销毁当前页面* frame.dispose();  
 *// 跳转页面* UserIframe userIframe = new UserIframe();  
 } else {  
 inUsername.setText("");  
 inPassword.setText("");  
 JOptionPane.*showMessageDialog*(null, "登录失败,账号或密码错误");  
 }  
 }  
});

**注册**

1. 用户在进行注册账号时，需要通过该界面输入用户名、密码、确认密码、邮箱和地址，并将信息用User封装后提交，然后点击“提交”按钮进行注册注册界面设计如图4.5所示。

图 4.5 注册界面设计

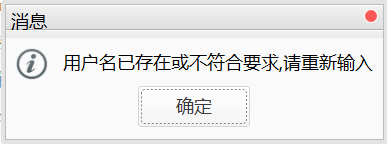
1. 用户在点击“提交”按钮之前必须输入必要的信息，否则系统将会提示，提示信息如图4.6所示。

图 4.6 提示“用户名已存在或不符合要求，请重新输入”

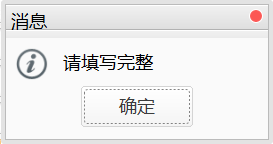
1. 用户名如果已经注册过，系统将会给出“用户名已存在或不符合要求，请重新输入”如图4.7所示。

图 4.7 提示“请填写完整”

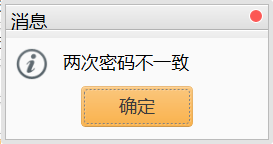
1. 如果用户输入的两次密码不一致，系统将会给出“两次密码不一致”的提示信息，如图4.8所示。

图 4.8 提示“两次密码不一致”

1. 如果用户输入的注册信息正确，系统就会将从界面层获取的信息封装为User，并通过数据交互层对数据库进行操作，给出“注册成功”的提示信息。
2. 注册按钮事件处理的核心代码：

submit.addActionListener(new ActionListener() {  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 if (inUsername.getText().isEmpty() || inPassword.getText().isEmpty() || inMail.getText().isEmpty() || inLocal.getText().isEmpty()) {  
 JOptionPane.*showMessageDialog*(null, "请填写完整");  
 } else if (!inPassword.getText().equals(inRepeatPassword.getText())) {  
 JOptionPane.*showMessageDialog*(null, "两次密码不一致");  
 } else if (UserService.*checkUser*(inUsername.getText())) {  
 JOptionPane.*showMessageDialog*(null, "用户名已存在或不符合要求,请重新输入");  
 } else {  
 User user = new User();  
 user.setUsername(inUsername.getText());  
 user.setPassword(inPassword.getText());  
 user.setEmail(inMail.getText());  
 user.setAddress(inLocal.getText());  
 user.setIs\_admin("0");  
 int res = UserDB.*addUser*(user);  
 if (res == 1) {  
 JOptionPane.*showMessageDialog*(null, "注册成功");  
 frame.dispose();  
 } else {  
 JOptionPane.*showMessageDialog*(null, "注册失败");  
 }  
 }  
 }  
});

4.4.3 普通用户界面

该界面包含四个窗体类：UserIframe、ChangeUserInfo、ShopList和PurchaseList，运行结果如图4.9，4.10，4.12，4.18。

图 4.9 用户登录主界面

**用户登录主界面**

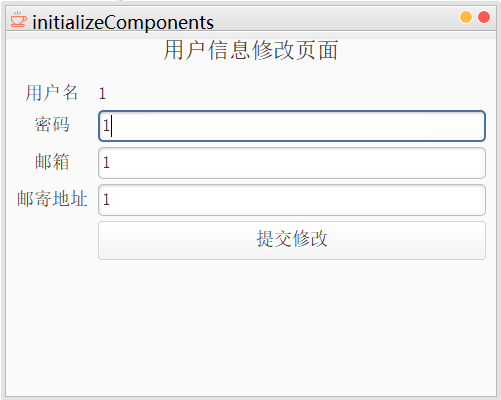
1. 普通用户登录后的主界面会显示四个选项，分别为“修改信息”“商品列表”“我的购买”和“返回登录”，通过登录页面传入的User对象，在登录后标题左方会显示用户身份，标题右方会显示“欢迎您”和用户的用户名，如图4.9所示。
2. 用户点击修改信息后会跳转到ChangeUserInfo界面，用户名不可修改，可修改密码，邮箱和邮件地址，如图4.10所示。

图 4.10 用户信息修改界面

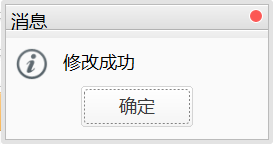
1. 在用户输入界面输入框的内容后，点击提交修改，数据交互层操作数据库，并提示“修改成功”如图4.11所示。

图 4.11 提示“修改成功”

1. 点击“提交修改”按钮事件处理的核心代码：

submit.addActionListener(new ActionListener() {  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 user.setUsername(inUserNameTextField.getText());  
 user.setPassword(inPassword.getText());  
 user.setEmail(inMail.getText());  
 user.setAddress(inLocal.getText());  
 user.setIs\_admin(inIsAdmin.isSelected() ? "1" : "0");  
 int res = UserDB.*updateUser*(user);  
 if (res == 1) {  
 JOptionPane.*showMessageDialog*(null, "修改成功");  
 } else {  
 JOptionPane.*showMessageDialog*(null, "修改失败");  
 }  
 }  
});

**商品列表界面**

1. 用户点击商品列表后跳转到ShopList界面，通过与数据交互层的操作，让用户可以直观的看到商品列表。

图 4.12 商品列表界面

1. 通过选择查询的条件即可在其后方展开一个列表，选择需要查询的内容，点击“搜索”按钮后通过与数据交互层的操作显示需要查询的内容。

图 4..13 商品列表选择菜单

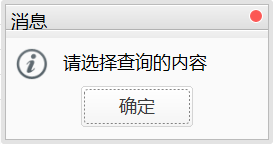
1. 如果在选择查询条件中没有选择内容，则提示“请选择查询的内容”，如图4.14所示。

图.4.14 提示“请输入查询的内容”

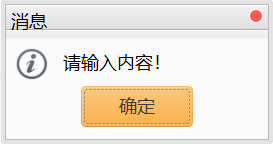
1. 同样的，在输入框中为空时则提示“请输入内容”，如图4.15所示。

图 4.15 提示“请输入内容”

1. 当用户选择商品后在页面底部会显示出商品的属性，时用户直观的查看商品的描述等，在用户点击购买按钮时，检查是否选择商品，如果选择，则数据交互层操作数据库，执行ProductService中的buy方法，实现购买的操作，在选择商品并点击购买按钮后提示“购买成功，谢谢惠顾”。如图4.16所示。

图 4.16 用户购买成功界面

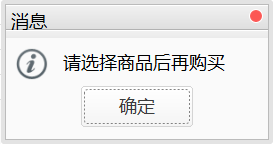
1. 如果用户没有选择商品就进行购买，则系统提示“请选择商品后购买”，如图4.17所示。

图 4.17 提示“请选择商品后再购买”

1. 点击“购买”按钮和“搜索”按钮事件处理的核心代码：

**购买按钮**

buy.addActionListener(new ActionListener() {  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 if (selectProduct != null && ProductService.*buyProduct*(userId, selectProduct.getId()) == 1) {  
 JOptionPane.*showMessageDialog*(null, "购买成功，谢谢惠顾！");  
 } else {  
 JOptionPane.*showMessageDialog*(null, "请选择商品后再购买");  
 }  
 }  
});

**搜索按钮**

search.addActionListener(new ActionListener() {  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 int selectedIndex = selectModel.getSelectedIndex();  
 switch (selectedIndex) {  
 case 0:  
 JOptionPane.*showMessageDialog*(null, "请选择查询的内容");  
 break;  
 case 1:  
 List<Product> withCategory = ProductService.*searchByCategory*(CategoryDB.*fromCategoryGetId*((String) categoryList.getSelectedItem()));  
 initializeTableList(withCategory);  
 break;  
 case 2:  
 if (!searchText.getText().isEmpty()) {  
 List<Product> withPrice = ProductService.*searchByPrice*(Double.*parseDouble*(searchText.getText()));  
 initializeTableList(withPrice);  
  
 } else {  
 JOptionPane.*showMessageDialog*(null, "请输入内容！");  
 }  
 break;  
 case 3:  
 if (!searchText.getText().isEmpty()) {  
 List<Product> withName = ProductService.*searchByName*(searchText.getText());  
 initializeTableList(withName);  
 } else {  
 JOptionPane.*showMessageDialog*(null, "请输入内容！");  
 }  
 break;  
 default:  
 JOptionPane.*showMessageDialog*(null, "查询出错！");  
 }  
 }  
});

**我的已购界面**

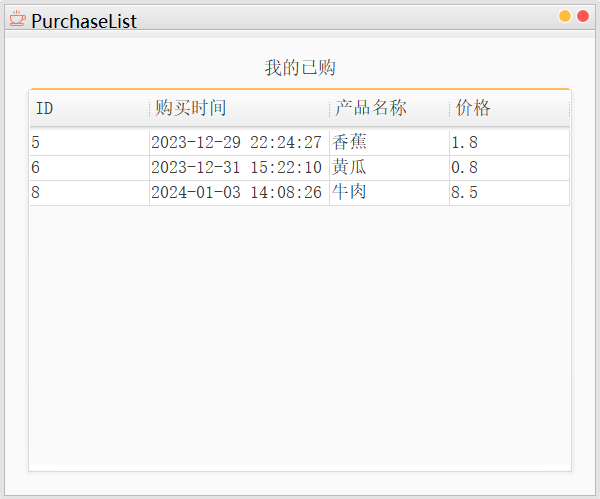
1. 在用户点击我的已购后，通过存储在Login界面的静态User变量获取用户身份来进行已购列表的查询，并将查询到的ProductList显示在界面中。如图4.18所示。

图 4.18 我的已购界面

1. 进行列表查询的核心代码如下：
2. private void initializeTableList() {  
    *// 初始化表格组件* DefaultTableModel model = new DefaultTableModel();  
    *// 设置表头* model.setColumnIdentifiers(new Object[]{"ID", "购买时间", "产品名称", "价格"});  
    *// 获取产品数据* try (ResultSet resultSet = ProductDB.*getPurchaseListByUserId*(userId)) {  
    while (resultSet.next()) {  
    Object[] rowData = new Object[4];  
    rowData[0] = resultSet.getInt(1);  
    rowData[1] = resultSet.getString(2);  
    rowData[2] = resultSet.getString(3);  
    rowData[3] = resultSet.getDouble(4);  
    model.addRow(rowData);  
    }  
    } catch (Exception e) {  
    e.printStackTrace();  
    }  
     
    *// 添加表格列头* tableList.setModel(model); *// 设置表格的数据模型* tableList.getColumnModel().getColumn(0).setPreferredWidth(10);  
    tableList.getColumnModel().getColumn(2).setPreferredWidth(10);  
    tableList.getColumnModel().getColumn(3).setPreferredWidth(10);  
   }

用户点击返回登录后则跳转到用户登录界面。

4.4.4 管理员界面

界面包含四个窗体类：UserIframe、ChangeUserInfo、UserManagementIframe、ProductManagementIframe和PurchaseList，运行结果如图4.19，4.23，4.20，4.18。

图 4.19 管理员登录界面

**用户管理界面**

1. ****当用户身份为管理员时，菜单显示变为“用户管理”、“商品管理”、“销量统计”，在管理员点击“用户管理”时，跳转到UserManagementIframe界面，通过数据交互层查询用户列表，将封装为包含User的List显示到界面中的表格。如图4.20所示。

图 4.20 用户管理界面

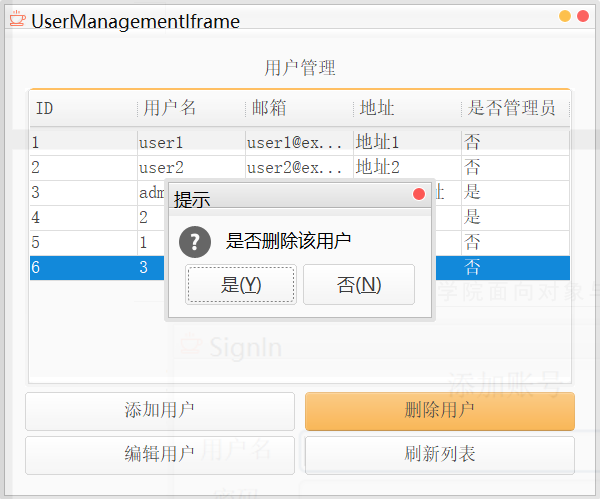
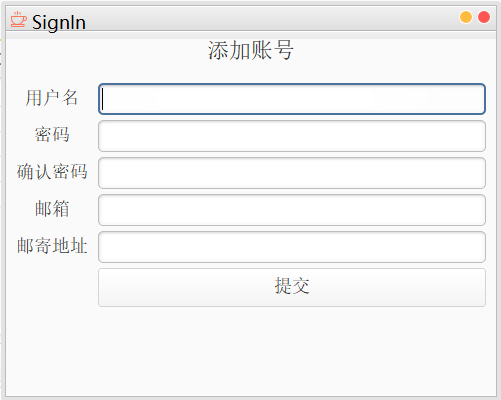
1. 点击添加用户跳转到SignIn界面，执行和注册相同的逻辑，如图4.21所示。
2. 当管理员点击删除用户时，执行删除用户逻辑，通过数据交互层操作数据库执行指定sql语句，完成删除用户的操作，在点击删除时会提示“是否删除用户”，点击是即删除并刷新用户列表，点击否则取消删除，不做操作。如图4.22所示。

图 4.22 删除用户

图 4.21添加用户

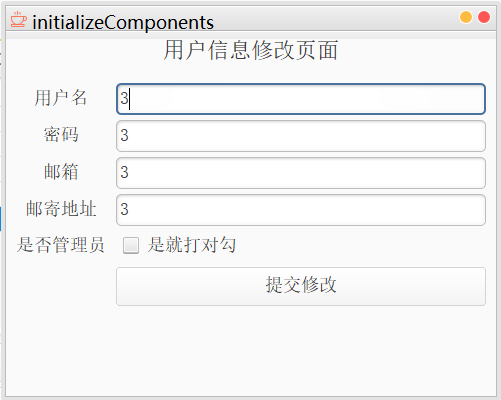
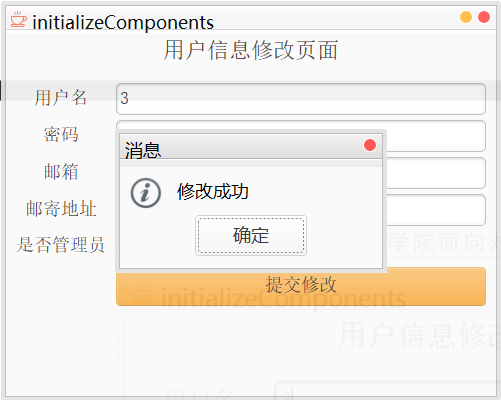
1. 当管理员点击编辑用户时，跳转到ChangeUserInfo界面，通过在Login界面中存储的静态变量确认登录用户身份，如果为管理员，则设置用户名界面可编辑，并通过传入管理员选择列表中的Use对象进行修改，并将传入的用户对象属性显示在对应界面中以供修改，点击提交修改后用户修改成功并提示。如图4.23和4.24所示。

图 4.24 提示“修改成功”

图 4.23 用户信息修改界面

1. 用户管理页面核心代码如下：

**判断是否管理员**

private void checkAdmin(User changeUser) {  
 user = Login.*user*;  
 if (user.getIs\_admin().equals("1") && changeUser != null) {  
 user = changeUser;  
 inUserNameTextField.setVisible(true);  
 inUsername.setVisible(false);  
 isAdmin.setVisible(true);  
 inIsAdmin.setVisible(true);  
 } else {  
 inUserNameTextField.setVisible(false);  
 inUsername.setVisible(true);  
 isAdmin.setVisible(false);  
 inIsAdmin.setVisible(false);  
 }  
}

**删除用户按钮逻辑**

delUser.addActionListener(new ActionListener() {  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 int selectedRow = tableList.getSelectedRow();  
 if (selectedRow == -1) {  
 JOptionPane.*showMessageDialog*(null, "请选择用户");  
 return;  
 }  
 *// 询问是否删除* int confirm = JOptionPane.*showConfirmDialog*(null, "是否删除该用户", "提示", JOptionPane.*YES\_NO\_OPTION*);  
 int userId = (int) tableList.getValueAt(selectedRow, 0);  
 if (confirm == JOptionPane.*YES\_OPTION*) {  
 int res = UserDB.*deleteUser*(userId);  
 if (res == 1) {  
 JOptionPane.*showMessageDialog*(null, "删除成功");  
 initializeTableList();  
 } else {  
 JOptionPane.*showMessageDialog*(null, "删除失败");  
 }  
 }  
 }  
});

**商品管理页面**

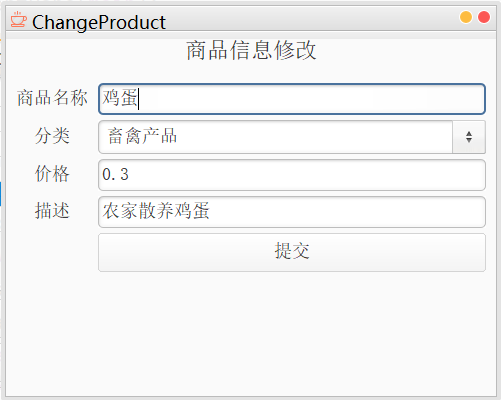
1. 管理员点击商品管理，跳转到ProductManagementIframe，从数据交互层获取数据并显示。如图4.25所示。
2. 管理员在选中商品且点击商品修改后，跳转到ChangeProduct并显示修改前商品的信息，点击提交后获取输入框内容并封装为Product对象进行数据库操作，添加商品。如图4.26所示。

图 4.26 商品信息修改界面

图 4.25 商品管理界面

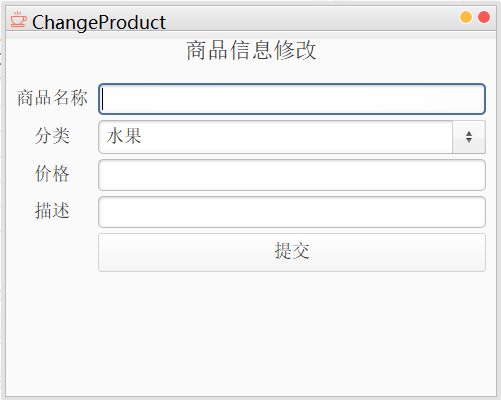
1. 管理员在点击添加商品后，跳转到ChangeProduct界面，界面通过传入值来判断管理员选择修改还是添加商品，管理员在输入商品信息后点击提交按钮可将各个输入框的内容封装为Product对象并提交到数据库进行新增商品操作。如图4.27所示。
2. 如果检查到商品属性输入框中无内容，系统提示“请填写完整信息”，如图4.28所示。

图 4.27 新增商品

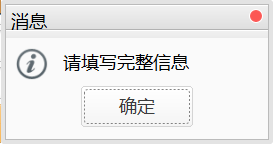
1. 点击“提交”按钮事件核心代码如下：

图 4.28 提示“请填写完整信息”

edit.addActionListener(new ActionListener() {  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 int selectedRow = tableList.getSelectedRow();  
 if (selectedRow == -1) {  
 JOptionPane.*showMessageDialog*(null, "请选择产品");  
 return;  
 }  
 int productId = (int) tableList.getValueAt(selectedRow, 0);  
 Product product = ProductService.*getProduct*(productId);  
 ChangeProduct changeProduct = new ChangeProduct(product);  
 }  
});

**分类管理界面**

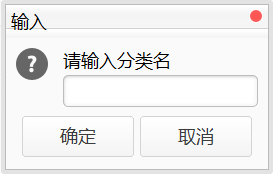
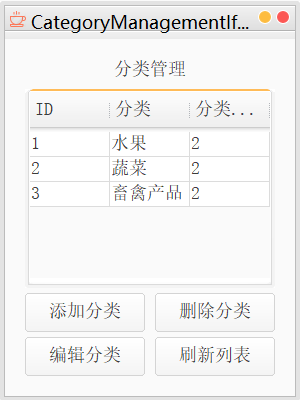
1. 管理员点击分类管理，跳转到CategoryManagementIframe，从数据交互层获取数据并显示。如图4.29所示。
2. 当管理员点击添加分类时，弹出输入框，通过输入在输入框中的内容新增商品分类并刷新列表。如图4.30所示。

图 4.30 添加分类

图4.29 分类管理界面

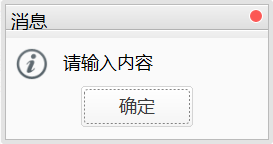
1. 如果输入为空则提示“请输入内容”，如图4.31所示。
2. 管理员在选择分类后点击编辑分类时，跳出包含选择分类名称的输入框，在修改分类名称之后刷新分类列表。如图4.32所示。

图 4.31 提示“请输入内容”

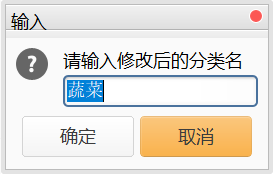
1. 分类管理的核心代码如下：

图 4.32 编辑分类

**添加分类**

addCategory.addActionListener(new ActionListener() {  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 String name = JOptionPane.*showInputDialog*(null, "请输入分类名");  
 if (name == null) {  
 return;  
 }  
 *// 输入判空* if (name.isEmpty()) {  
 JOptionPane.*showMessageDialog*(null, "请输入内容");  
 return;  
 }  
 if (CategoryDB.*checkSameName*(name)) {  
 JOptionPane.*showMessageDialog*(null, "该分类已存在");  
 } else {  
 CategoryDB.*addCategory*(name);  
 }  
 initializeTableList();  
 }  
});

**编辑分类**

edit.addActionListener(new ActionListener() {  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 int selectedRow = tableList.getSelectedRow();  
 if (selectedRow == -1) {  
 JOptionPane.*showMessageDialog*(null, "请选择分类");  
 return;  
 }  
 String name = JOptionPane.*showInputDialog*(null, "请输入修改后的分类名", tableList.getValueAt(selectedRow, 1));  
 *// 点击弹窗的否则无事件* if (name == null) {  
 return;  
 }  
 if (CategoryDB.*checkSameName*(name)) {  
 JOptionPane.*showMessageDialog*(null, "该分类已存在");  
 } else {  
 int categoryId = (int) tableList.getValueAt(selectedRow, 0);  
 int res = CategoryDB.*updateCategory*(categoryId, name);  
 if (res == 1) {  
 JOptionPane.*showMessageDialog*(null, "修改成功");  
 }  
 }  
 initializeTableList();  
 }  
});

**删除分类**

delCategory.addActionListener(new ActionListener() {  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 int selectedRow = tableList.getSelectedRow();  
 if (selectedRow == -1) {  
 JOptionPane.*showMessageDialog*(null, "请选择分类");  
 return;  
 }  
 if (CategoryDB.*checkCategoryHaveProduct*((int) tableList.getValueAt(selectedRow, 0)) != 0) {  
 JOptionPane.*showMessageDialog*(null, "该分类已有商品，无法删除");  
 return;  
 }  
 *// 询问是否删除* int confirm = JOptionPane.*showConfirmDialog*(null, "是否删除该分类", "提示", JOptionPane.*YES\_NO\_OPTION*);  
 if (confirm == JOptionPane.*YES\_OPTION*) {  
 int categoryId = (int) tableList.getValueAt(selectedRow, 0);  
 int res = CategoryDB.*deleteCategory*(categoryId);  
 if (res == 1) {  
 JOptionPane.*showMessageDialog*(null, "删除成功");  
 }  
 initializeTableList();  
 }  
 }  
});

**销售统计界面**

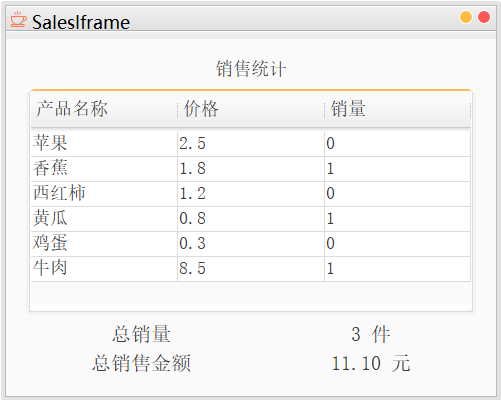
1. 管理员点击销售统计按钮后，跳转到SalesLframe界面，并查询数据库中的各个用户购买订单的内容集成到表格中，在获取已销售商品的列表后，将数据进行统计，分为总销量和总销售金额。如图4.31所示。

图 4.31 销售统计界面

1. 销售统计界面核心代码如下：

private void initializeTableList() {  
 *// 初始化表格组件* DefaultTableModel model = new DefaultTableModel();  
 *// 设置表头* model.setColumnIdentifiers(new Object[]{"产品名称", "价格", "销量"});  
 *// 获取销量数据* List<Sales> salesList = SalesDB.*getSalesList*();  
 int totalSum = 0;  
 double moneySum = 0.0;  
 for (Sales sales : salesList) {  
 Product product = ProductService.*getProduct*(sales.getProductId());  
 Object[] rowData = new Object[4];  
 rowData[0] = product.getName();  
 rowData[1] = product.getPrice();  
 rowData[2] = sales.getTotalSales();  
 totalSum += sales.getTotalSales();  
 moneySum += product.getPrice() \* sales.getTotalSales();  
 model.addRow(rowData);  
 }  
 outTotal.setText(totalSum + " 件");  
 outMoney.setText(String.*format*("%.2f", moneySum) + " 元");  
 tableList.setModel(model);  
}