长 沙 学 院

软件工程基础实训Ⅱ 实训

|  |  |
| --- | --- |
| **题目** | **基于C/S架构的餐饮管理系统** |
| **学院** | **计算机科学与工程** |
| **专业(班级)** | **软件工程（21软件01）** |
| **姓名** | **高骅林** |
| **学号** | **B20210304116** |
| **指导教师** | **潘怡、杨刚** |
| **起止日期** | **2023.06.05～2022.06.16** |

实训任务书

**课程名称：软件工程基础实训II**

**实训题目：基于C/S架构的餐饮管理系统**

**已知技术参数和设计要求：**

1. 问题描述（功能要求）：

本次案例将完成一个基于C/S架构的餐饮管理系统。

（1）.进行餐饮管理系统的功能设计。

（2）.设计餐饮管理系统的数据库。

（3）.利用JAVA的GUI类设计开发友好的人机接口。

（4）.利用JAVA语言设计监听用户操作和JDBC进行数据库的处理。

餐饮管理系统主要有6大模块：

（1）.员工管理：对员工实现增删改查。

（2）.客户管理：对客户实现增删改查。

（3）.餐台管理：对餐台实现增删改查。

（4）.菜品管理：对菜品分类、菜品实现增删改查。

（5）.点菜管理：服务员对某客户、某一空闲餐台实行开台，同时实现点菜，将餐台号与所点的菜品对应起来，分别显示出来，并记录开台时间。

（6）.结账管理：收银员对某一餐台通过统计显示消费的菜品清单统计出消费金额，通过手动输入实收金额进行找零的计算，并显示，完成结账的操作，并记录成统计数据。

基础实训II有2周时间40学时，学生基本必做任务：独立完成基于C/S架构的餐饮管理系统。

选做扩展：菜品推荐、销售统计。

**2. 运行环境要求：**

（1）客户端：

windows操作系统，JDK1.6，Eclipse。

（2）服务器：

MySQL5.6或 SQL Server 2008 数据库服务器。

3. 技术要求：

1） 掌握软件工程的需求分析和系统设计方法。

2) 掌握数据库的分析与设计，完成餐饮管理系统数据库的设计。

3) 掌握JAVA的GUI设计方法，完成登录、主界面、菜品分类管理界面、菜品管理界面等人机接口。

4） 掌握JAVA面向对象程序设计方法，设计系统所需的各种类。

5) 掌握JDBC方法，对数据库进行操作，完成主程序和菜品分类、菜品管理等功能。

**实训工作量：**

40课时

**工作计划：**

1.班级

20软件1-4

2.课时及教室安排

见实训课程工作计划表。

实训报告

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 学生姓名 | 高骅林 | 学号 | B20210304116 | 班级 | 21软件1班 |
| 专业 | 软件工程 | | 指导教师姓名 | 杨刚、潘怡 | |
| 项目名称 | 基于C/S架构的餐饮管理系统 | | | | |
| **实训过程**：  需求分析：为了解决现代餐饮行业管理中出现的销售信息、管理信息的问题，开发一个基于C/S架构的餐饮管理系统，对餐饮行业进行现代化、便捷化、科学化的管理，实现对员工、客户、菜品以及餐台信息的增删改查；服务员对某客户、某一空闲餐台实行开台，同时实现点菜，将餐台号与所点的菜品对应起来，分别显示出来，并记录开台时间；收银员对某一餐台通过统计显示消费的菜品清单统计出消费金额，通过手动输入实收金额进行找零的计算，并显示，完成结账的操作。  系统设计：系统功能包括员工管理模块、客户管理模块、餐台管理模块、菜品管理模块、客户点菜模块、以及结账管理模块。  编码实现：利用Java的GUI设计人机交互界面；利用Java的JDBC所提供的方法对数据库进行操纵。  系统测试：测试系统的完备性、实用性和可靠性。  交付实施：经测试成功后对系统进行维护和运营。 | | | | | |
| **实训成果**：  实现了餐饮管理系统的基本功能，界面美观、人机交互友好 | | | | | |
| **实训总结**：  心得体会：通过这两周的实训，我对Java的GUI编程和JDBC操作数据库有了更深刻的认识，在编写代码时学会了合理利用数据结构和算法去解决实际遇到的问题，学会了如何设计以及美化界面，实现界面良好的人机交互功能，在遇到疑难问题时，先独立思考，其次查阅网络资料、和同学进行讨论，最终得到自己满意的结果。  存在的问题：数据库安全性有待加强，存在恶意攻击的风险。对实务的并发未作处理。  改进方向：  一、加强数据库的安全防护措施：  1、使用安全的密码策略  2、使用安全的帐号策略  3、加强数据库日志的记录  4、管理扩展存储过程  5、使用协议加密  二、引入多线程对事务进行并发处理 | | | | | |

基于C/S架构的餐饮管理系统

设计说明书

作者：高骅林(B20210304116)

计算机科学与工程学院

2023年 06 月 15日

**摘要**

本课程设计的主要内容为设计一个基于C/S架构的餐饮管理系统。

本课程设计基于Java GUI编程，数据库编程，Java JDBC编程。

本系统包含客户管理、员工管理、菜品管理、餐台管理、点菜管理和结账管理六大基本模块。客户管理实现对客户信息的增删改查；员工管理实现对员工信息的增删改查；菜品管理实现对菜品信息的增删改查；餐台管理实现对餐台信息的增删改查；点菜管理实现服务员对客户的人数进行开台，并进行点菜和退菜，实现购物车的功能；结账管理实现对订单的查询和修改功能。

**关键词：**C/S架构，餐饮管理系统，Java GUI编程，Java JDBC

目录

[1 引言 1](#_Toc23127)

[1.1 编写目的 1](#_Toc12734)

[1.2 参考资料 1](#_Toc28322)

[2 需求规约 2](#_Toc15456)

[2.1 功能需求 2](#_Toc7387)

[2.2 界面需求 7](#_Toc22906)

[2.3 数据需求 7](#_Toc6106)

[3 系统设计 10](#_Toc5298)

[3.1 运行环境 10](#_Toc30770)

[3.2 系统静态结构设计 10](#_Toc31177)

[3.3 人机接口设计 12](#_Toc562)

[3.3.6 用户接口5（客户管理界面）设计 17](#_Toc19737)

[3.4 登录模块设计说明 26](#_Toc13284)

[3.5 员工管理模块设计说明 26](#_Toc20003)

[3.6 客户管理模块设计说明 29](#_Toc7291)

[3.7菜品管理模块设计说明 33](#_Toc22205)

[3.8 点单管理模块设计说明 35](#_Toc23298)

[3.9 结账管理模块设计说明 36](#_Toc3135)

[4 数据库设计 37](#_Toc30933)

[4.1 数据库环境说明 37](#_Toc17119)

[4.2 数据库的命名规则 37](#_Toc17233)

[4.3 逻辑结构设计 38](#_Toc15483)

[4.4 物理结构设计 40](#_Toc24298)

[4.4.1 建库语言 40](#_Toc5507)

[5 测试用例设计 43](#_Toc248)

[附录 46](#_Toc5642)

[附录1 程序运行结果 46](#_Toc240)

[附录2 程序源代码 55](#_Toc28827)

# 1 引言

## 1.1 编写目的

本文档是“餐饮管理系统”的软件设计说明书。编写目的是：明确软件需求、概要设计、详细设计和功能测试用例，为软件后续开发和维护提供指导。本文档的读者对象是需求分析人员、系统设计人员、系统开发人员、系统测试人员。

## 1.2 参考资料

[1] 徐转运, 张扬, 王淼. Java高级程序设计[M]. 北京:清华大学出版社, 2014.

# 2 需求规约

## 2.1 功能需求

本次案例将完成一个基于C/S架构的餐饮管理系统。

（1）.进行餐饮管理系统的功能设计。

（2）.设计餐饮管理系统的数据库。

（3）.利用JAVA的GUI类设计开发友好的人机接口。

（4）.利用JAVA语言设计监听用户操作和JDBC进行数据库的处理。

餐饮管理系统主要有6大模块：

（1）.员工管理：对员工实现增删改查。

（2）.客户管理：对客户实现增删改查。

（3）.餐台管理：对餐台实现增删改查。

（4）.菜品管理：对菜品分类、菜品实现增删改查。

（5）.点菜管理：服务员对某客户、某一空闲餐台实行开台，同时实现点菜，将餐台号与所点的菜品对应起来，分别显示出来，并记录开台时间。

（6）.结账管理：收银员对某一餐台通过统计显示消费的菜品清单统计出消费金额，通过手动输入实收金额进行找零的计算，并显示，完成结账的操作，并记录成统计数据。

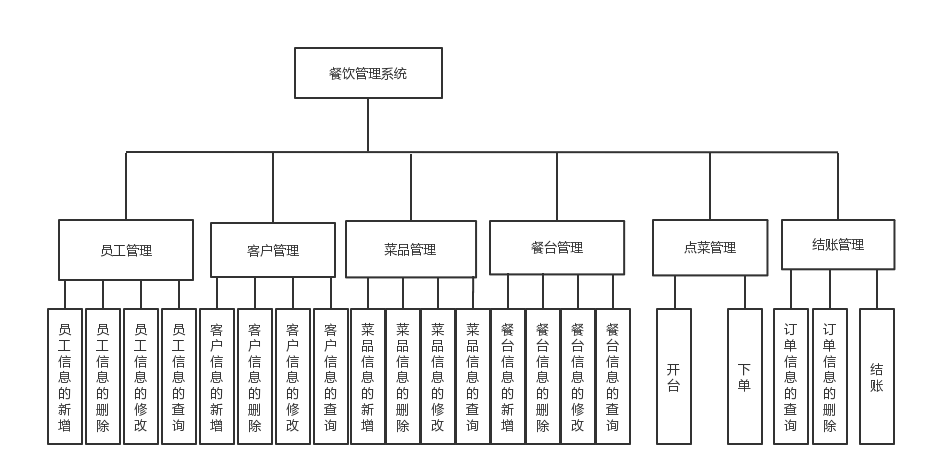


图2.1餐饮管理系统功能模块图

2.1.1 用户登录用例说明

用例名称：用户登录。

用例描述：餐厅管理员登录餐厅管理系统。

前置条件：管理员输入账号和密码。

后置条件：管理员登录成功。

活动步骤：

1. 管理员输入账号和密码
2. 点击登录按钮，管理员登录成功。

异常处理：

1. 用户账号或密码为空，用例终止。
2. 用户输入账号不存在或密码不正确，用例终止。
3. 用户输入账号或密码格式不正确，用例终止。

2.2.3 员工管理用例说明

用例名称：员工管理。

用例描述：餐厅管理员对员工信息进行增删改查。

前置条件：餐厅管理员对员工信息进行筛选。

后置条件：餐厅管理员对员工信息进行一系列操作。

活动步骤：

查询操作：

1. 餐厅管理员在文本框内输入员工信息，包括员工id，员工姓名，若未输入员工信息，系统将查询到所有员工。
2. 餐厅管理员查询的信息将在表格中显示。

新增操作：

1. 餐厅管理员点击新增员工按钮，弹出界面。
2. 餐厅管理员在新的界面里填写员工信息。
3. 员工信息添加成功后立即更新查询后的表格。

修改操作：

1. 餐厅管理员点击表格信息，弹出修改的界面。
2. 餐厅管理员在新的界面内的输入框中填写要修改的信息。
3. 员工信息修改成功后立即更新查询后的表格。

删除操作：

1. 餐厅管理员点击表格中该员工的id。
2. 餐厅管理员点击删除按钮。
3. 员工信息删除后立即更新查询后表格。

异常处理：

新增操作：若输入员工信息有误，系统将提示具体哪一部分出错。

修改操作：若输入要修改的信息有误，系统将提示修改错误，将不会对表格信息进行更新。

2.2.4 客户管理用例

用例名称：客户管理。

用例描述：餐厅管理员对客户信息进行增删改查。

前置条件：餐厅管理员对客户信息进行筛选。

后置条件：餐厅管理员对客户信息进行一系列操作。

活动步骤：

查询操作：

1. 餐厅管理员在文本框内输入客户信息，包括客户id，客户姓名，若未输入客户信息，系统将查询到所有客户。
2. 餐厅管理员查询的信息将在表格中显示

新增操作：

1. 餐厅管理员点击新增客户按钮，弹出界面。
2. 餐厅管理员在新的界面里填写客户信息。
3. 客户信息添加成功后立即更新查询后的表格。

修改操作：

1. 餐厅管理员点击表格信息，弹出修改的界面。
2. 餐厅管理员在新的界面内的输入框中填写要修改的信息。
3. 客户信息修改成功后立即更新查询后的表格。

删除操作：

1. 餐厅管理员点击表格中该客户的id。
2. 餐厅管理员点击删除按钮。
3. 客户信息删除后立即更新查询后表格。

异常处理：

新增操作：若输入客户信息有误，系统将提示具体哪一部分出错。

修改操作：若输入要修改的信息有误，系统将提示修改错误，将不会对表格信息进行更新。

2.2.5 菜品管理用例

用例名称：菜品管理。

用例描述：餐厅管理员对菜品信息进行增删改查。

前置条件：餐厅管理员对菜品信息进行筛选。

后置条件：餐厅管理员对菜品信息进行一系列操作。

活动步骤：

查询操作：

1. 餐厅管理员在文本框内输入菜品信息，包括菜品id，菜名，若未输入菜品信息，系统将查询到所有菜品。
2. 餐厅管理员查询的信息将在表格中显示

新增操作：

1. 餐厅管理员点击新增菜品按钮，弹出界面。
2. 餐厅管理员在新的界面里填写菜品信息。
3. 菜品信息添加成功后立即更新查询后的表格。

修改操作：

1. 餐厅管理员点击表格信息，弹出修改的界面。
2. 餐厅管理员在新的界面内的输入框中填写要修改的信息。
3. 客户信息修改成功后立即更新查询后的表格。

删除操作：

1. 餐厅管理员点击表格中该菜品的id。
2. 餐厅管理员点击删除按钮。
3. 菜品信息删除后立即更新查询后表格。

异常处理：

新增操作：若输入菜品信息有误，系统将提示具体哪一部分出错。

修改操作：若输入要修改的信息有误，系统将提示修改错误，将不会对表格信息进行更新。

2.2.6 餐台管理用例

用例名称：餐台管理。

用例描述：餐厅管理员对餐台信息进行增删改查。

前置条件：餐厅管理员对餐台信息进行筛选。

后置条件：餐厅管理员对餐台信息进行一系列操作。

活动步骤：

查询操作：

1. 餐厅管理员在文本框内输入餐台号，若未输入餐台号，系统将查询到所有餐台。
2. 餐厅管理员查询的信息将在表格中显示

新增操作：

1. 餐厅管理员点击新增餐台按钮，弹出界面。
2. 餐厅管理员在新的界面里填写餐台的座位数。
3. 菜品信息添加成功后立即更新查询后的表格。

修改操作：

1. 餐厅管理员点击表格信息，弹出修改的界面。
2. 餐厅管理员在新的界面内的输入框中填写要修改的信息。
3. 客户信息修改成功后立即更新查询后的表格。

删除操作：

1. 餐厅管理员点击表格中该餐台的餐台号。
2. 餐厅管理员点击删除按钮。
3. 餐台信息删除后立即更新查询后表格。

异常处理：

修改操作：若输入要修改的信息有误，系统将提示修改错误，将不会对表格信息进行更新。

2.2.7 点菜管理用例

用例名称：点菜管理。

用例描述：餐厅管理员对客户进行开台，点菜，下单操作。

前置条件；餐厅管理员输入客户的客户id和人数，对客户进行开台操作。

后置条件：餐厅管理员为客户完成点单，下单操作。

活动步骤：

1. 餐厅管理员输入客户id以及人数，对客户进行开台操作。
2. 系统将根据人数为该用户分配合适的餐台。
3. 餐厅管理员根据菜品信息来查询菜品，提供给客户去选菜。
4. 选菜完成后点击加入购物车后再点击下单，菜品添加到购物车中。
5. 确认无误后点击保存，生成新的订单。

异常处理：

1. 餐厅管理员输入错误的客户id或着非法的人数时，系统会提示具体的输入错误，并且不会对客户进行开台操作。
2. 餐厅管理员输入的人数大于现有空闲餐台的所能容纳的最大人数时，系统将提示找不到合适的餐台。

2.2.8 结账管理用例

用例名称：结账管理。

用例描述：餐厅管理员对订单进行查询，删除，结账的操作。

前置条件：餐厅管理员输入订单号或客户id，对订单进行模糊查询。

后置条件：餐厅管理员对所选订单进行结账处理。

活动步骤：

1. 餐厅管理员输入订单号或客户id，对订单进行查询。
2. 系统将结果显示在表格中。
3. 餐厅管理员选定某一订单，弹出新的界面。
4. 在新的界面中显示订单的应付金额与折后价格以及实付金额。
5. 餐厅管理员在实付金额处填写客户支付的金额，点击支付，系统将对该客户进行找零操作，同时更新查询后的表格。

异常处理：餐厅管理员输入金额小于折后价格，将提醒支付失败。

## 2.2 界面需求

1. 界面友好，操作简单。

2. 界面美观，用户体验感良好。

## 2.3 数据需求

表2.1 用户实体

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 属性名称 | 类型 | 长度 | 是否为空 | 唯一标识 | 说明 |
| Username | char | 255 | 否 | 是 | 用户名 |
| Password | char | 255 | 是 | 否 | 密码 |

表2.2客户实体

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 属性名称 | 类型 | 长度 | 是否为空 | 唯一标识 | 说明 |
| id | int | 11 | 否 | 是 | 客户号 |
| name | char | 40 | 否 | 否 | 姓名 |
| sex | char | 40 | 否 | 否 | 性别 |
| company | char | 40 | 是 | 否 | 单位 |
| tel | char | 11 | 是 | 否 | 电话号码 |
| cardId | char | 40 | 否 | 否 | 会员卡号 |

表2.3餐台实体

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 属性名称 | 类型 | 长度 | 是否为空 | 唯一标识 | 说明 |
| id | int | 11 | 否 | 否 | 餐台号 |
| seating | int | 11 | 否 | 否 | 座位号 |
| no | char | 40 | 否 | 是 | 序号 |
| status | char | 40 | 否 | 否 | 状态 |

表2.4菜品实体

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 属性名称 | 类型 | 长度 | 是否为空 | 唯一标识 | 说明 |
| id | int | 11 | 否 | 否 | 菜品号 |
| name | char | 40 | 否 | 是 | 菜名 |
| dishTypeId | int | 11 | 否 | 否 | 菜品号 |
| code | char | 40 | 否 | 否 | 品种分类 |
| unit | char | 40 | 是 | 否 | 单位 |
| price | double |  | 否 | 否 | 价格 |
| status | char | 40 | 否 | 否 | 状态 |

表2.5员工实体

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 属性名称 | 类型 | 长度 | 是否为空 | 唯一标识 | 说明 |
| id | int | 11 | 否 | 是 | 员工号 |
| name | char | 40 | 否 | 否 | 姓名 |
| sex | char | 40 | 否 | 否 | 性别 |
| tel | char | 40 | 是 | 否 | 电话号码 |
| birthday | date | 0 | 是 | 否 | 出生日期 |
| identityId | char | 0 | 是 | 否 | 身份证号 |
| address | char | 40 | 是 | 否 | 住址 |
| position | char | 40 | 否 | 否 | 职位 |
| freeze | char | 40 | 否 | 否 | 是否在职 |

表2.6订单实体

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 属性名称 | 类型 | 长度 | 是否为空 | 唯一标识 | 说明 |
| id | int | 11 | 否 | 是 | 订单编号 |
| orderNo | char | 40 | 否 | 否 | 订单流水号 |
| tableId | int | 11 | 是 | 否 | 餐桌号 |
| money | double |  | 否 | 否 | 消费金额 |
| createtime | datetime |  | 否 | 否 | 订单时间 |
| state | char | 40 | 否 | 否 | 状态 |
| customerId | int | 11 | 是 | 否 | 会员号 |
| number | int | 11 | 否 | 否 | 就餐人数 |

# 3 系统设计

## 3.1 运行环境

操作系统：Windows 11

数据库：MySQL 8.0.33

## 3.2 系统静态结构设计

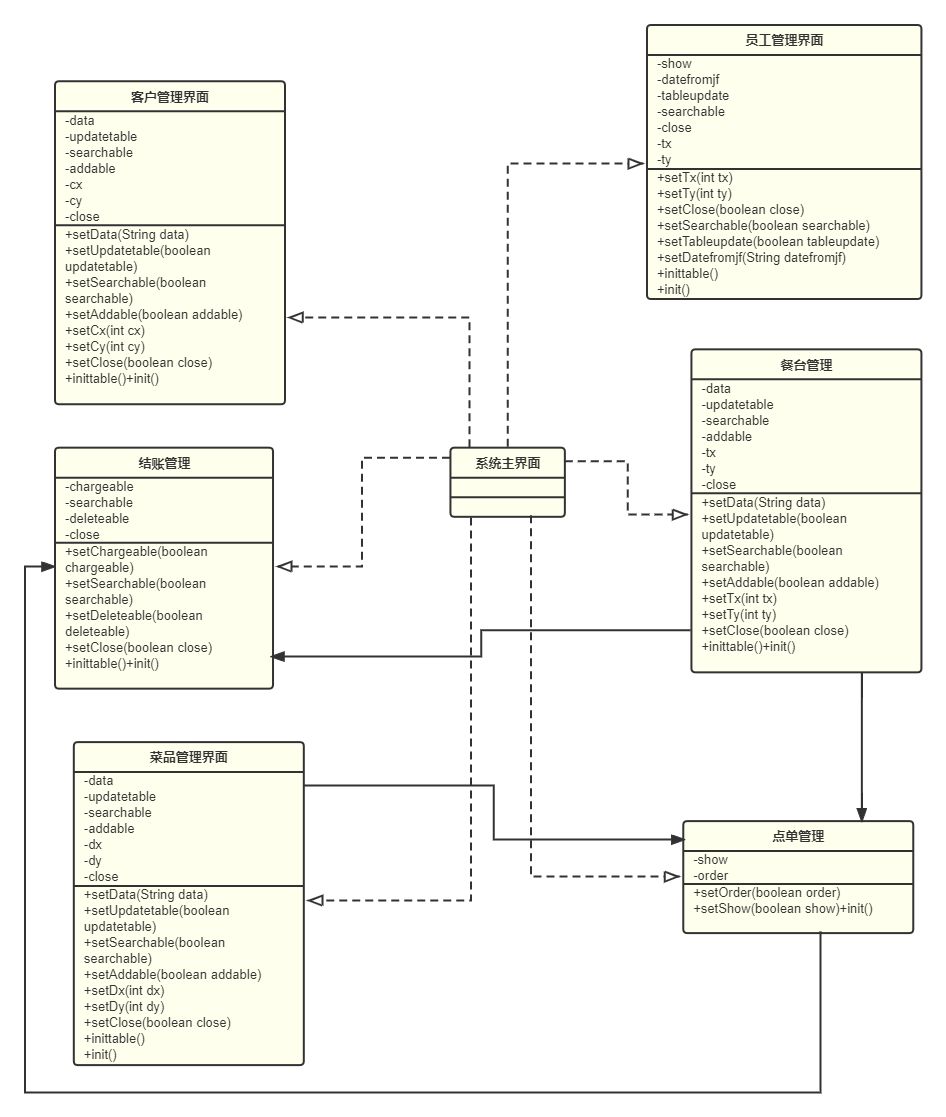


图3.1餐饮管理系统总体类图

系统静态结构说明

餐厅管理员通过主界面可以跳转至不同的界面，在员工管理界面，管理员可以实现对员工信息的增删改查，在客户管理界面，管理员可以实现对客户信息的增删改查，在菜品管理界面，管理员可以实现对菜品信息的增删改查，在餐台界面，管理员可以实现对餐台信息的增删改查，在点菜管理界面，管理员可以为顾客提供点菜服务，点菜完成后生成新的订单，在结账管理界面，管理员可以对订单进行查询和结账操作。

## 3.3 人机接口设计

3.3.1 设计应用类

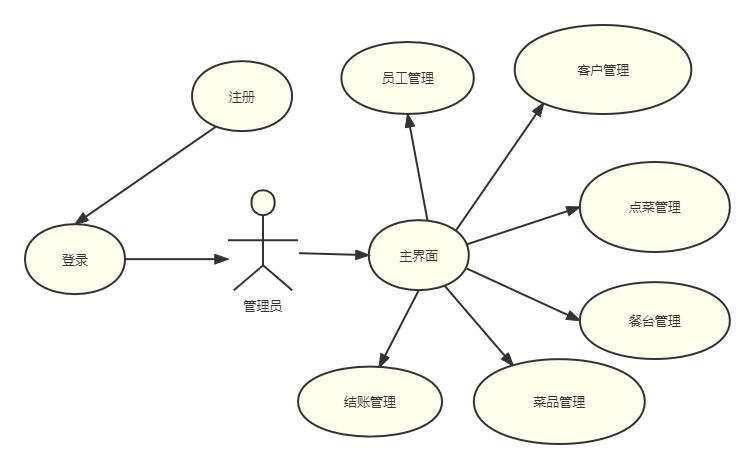


图3.2餐饮管理系统设计类图

3.3.2 用户接口1（登录界面）设计

**界面主要控件：**

JTextField，输入框，接收账号和密码。

JButton，按钮，触发登录注册事件。

**实现功能：**

1. 餐厅管理员输入账号密码，点击登录按钮跳转餐厅管理界面；账号密码为数据库账号密码。
2. 餐厅管理员点击注册按钮跳转注册界面。

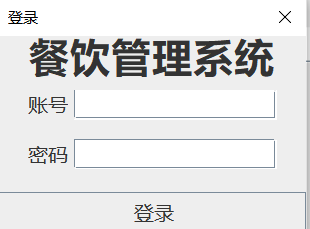


图3.3用户登录界面布局设计

3.3.4 用户接口3（餐饮管理系统主界面）设计

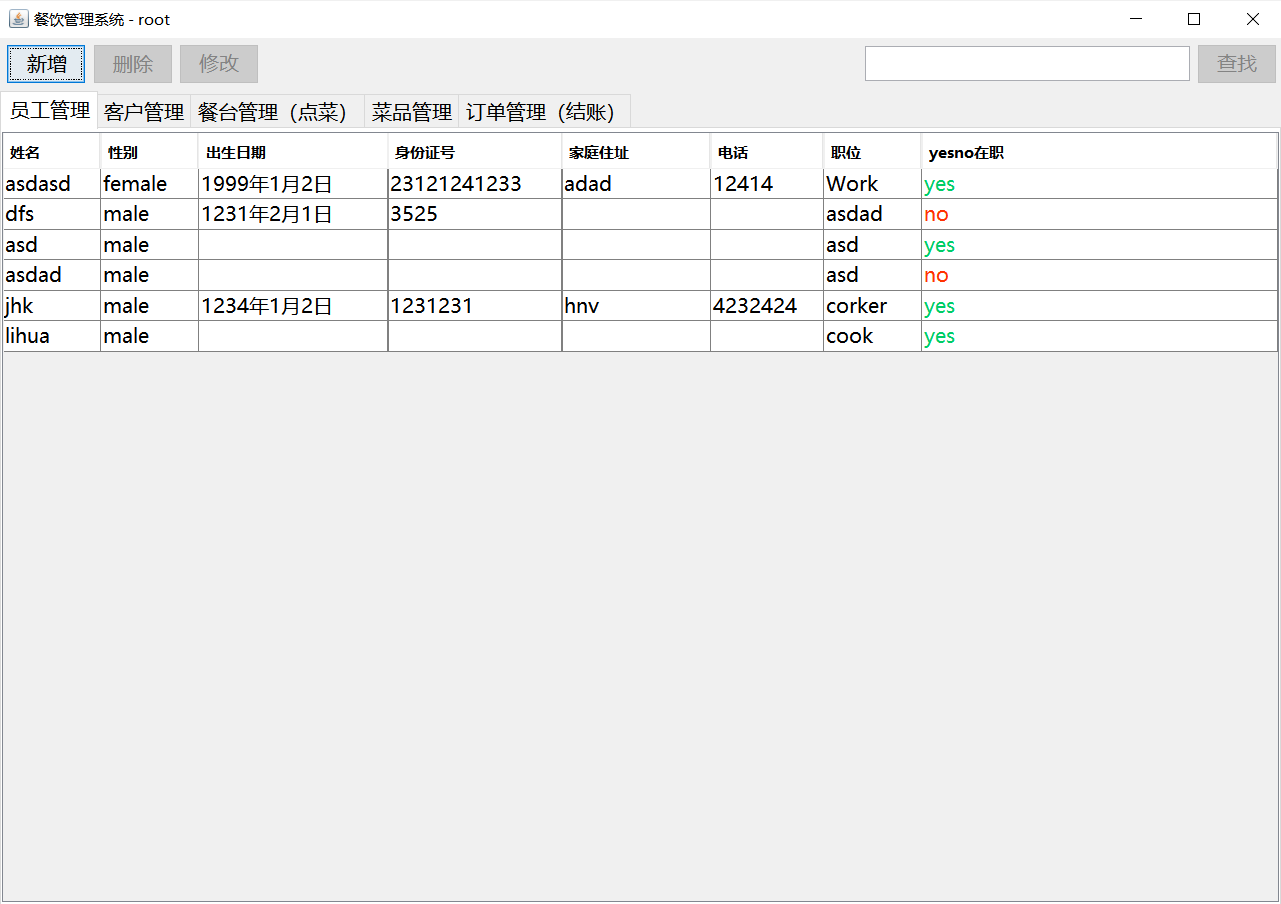


图3.5餐饮管理系统主界面

**界面主要控件：**

JButton，按钮，触发点击事件。

**实现功能：**

1. 点击员工管理，跳转至员工管理界面。
2. 点击客户管理，跳转至客户管理界面。
3. 点击餐台管理，跳转至餐台管理界面。
4. 点击菜品管理，跳转至菜品管理界面。
5. 点击结账管理，跳转至结账管理界面。
6. 点击登出按钮，退出餐饮管理系统。

3.3.5 用户接口4（员工管理界面）设计

**界面主要控件：**

JButton，按钮，触发点击事件。

JTextField，输入框，接收员工id和员工姓名。

**实现功能：**

1. 点击查询按钮，将对员工进行查询。
2. 点击新增按钮，将对进行员工的增添。
3. 点击表格中的某个员工后再点击删除员工，将对该员工进行删除。
4. 点击表格，将对员工信息进行修改。

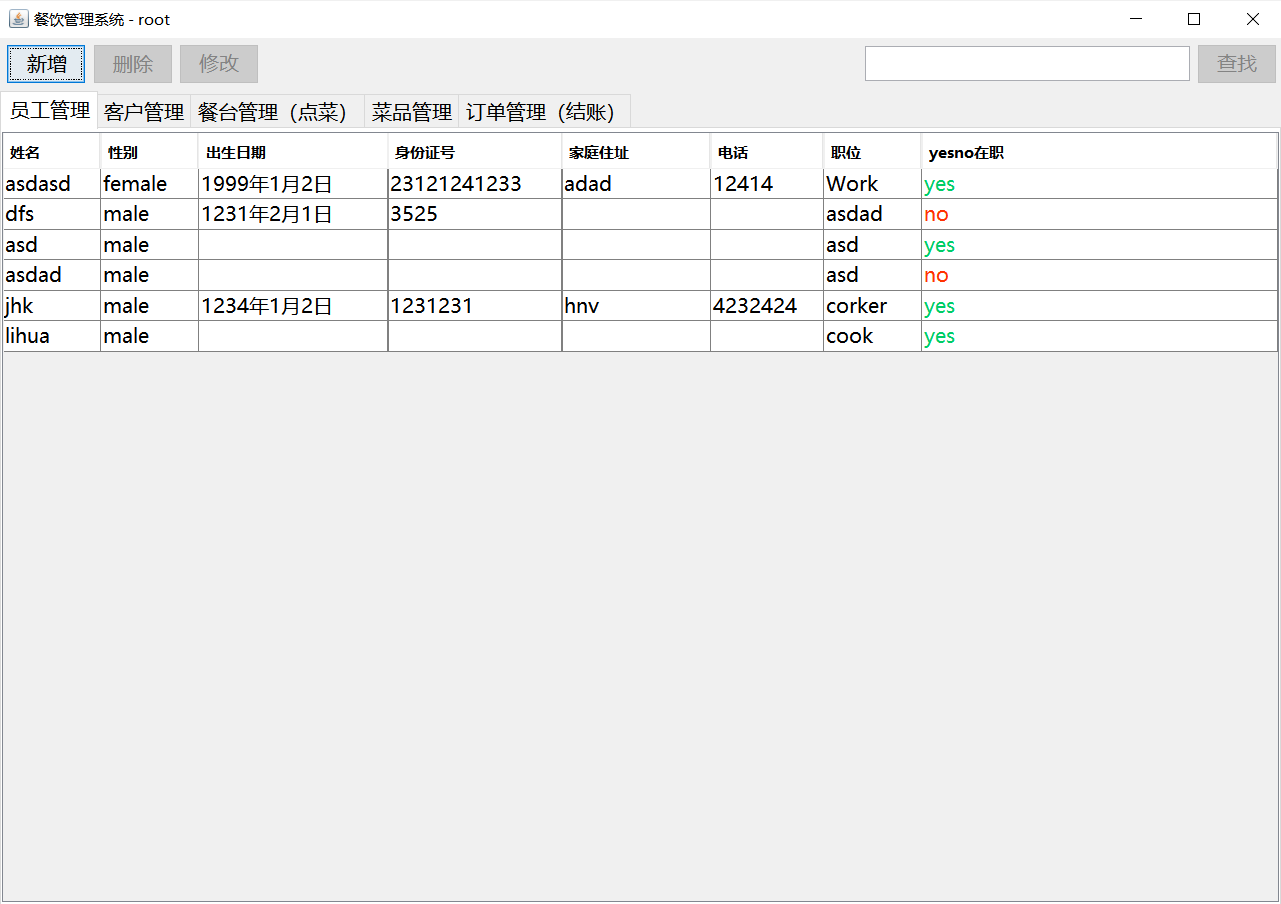


图3.6员工管理界面

## 3.3.6 用户接口5（客户管理界面）设计

**界面主要控件：**

JButton，按钮，触发点击事件。

JTextField，输入框，接收客户id和客户姓名。

**实现功能：**

1. 点击查询按钮，将对客户进行查询。
2. 点击新增按钮，将对进行客户的增添。
3. 点击表格中的某个客户后再点击删除客户，将对该客户进行删除。
4. 点击表格，将对客户信息进行修改。

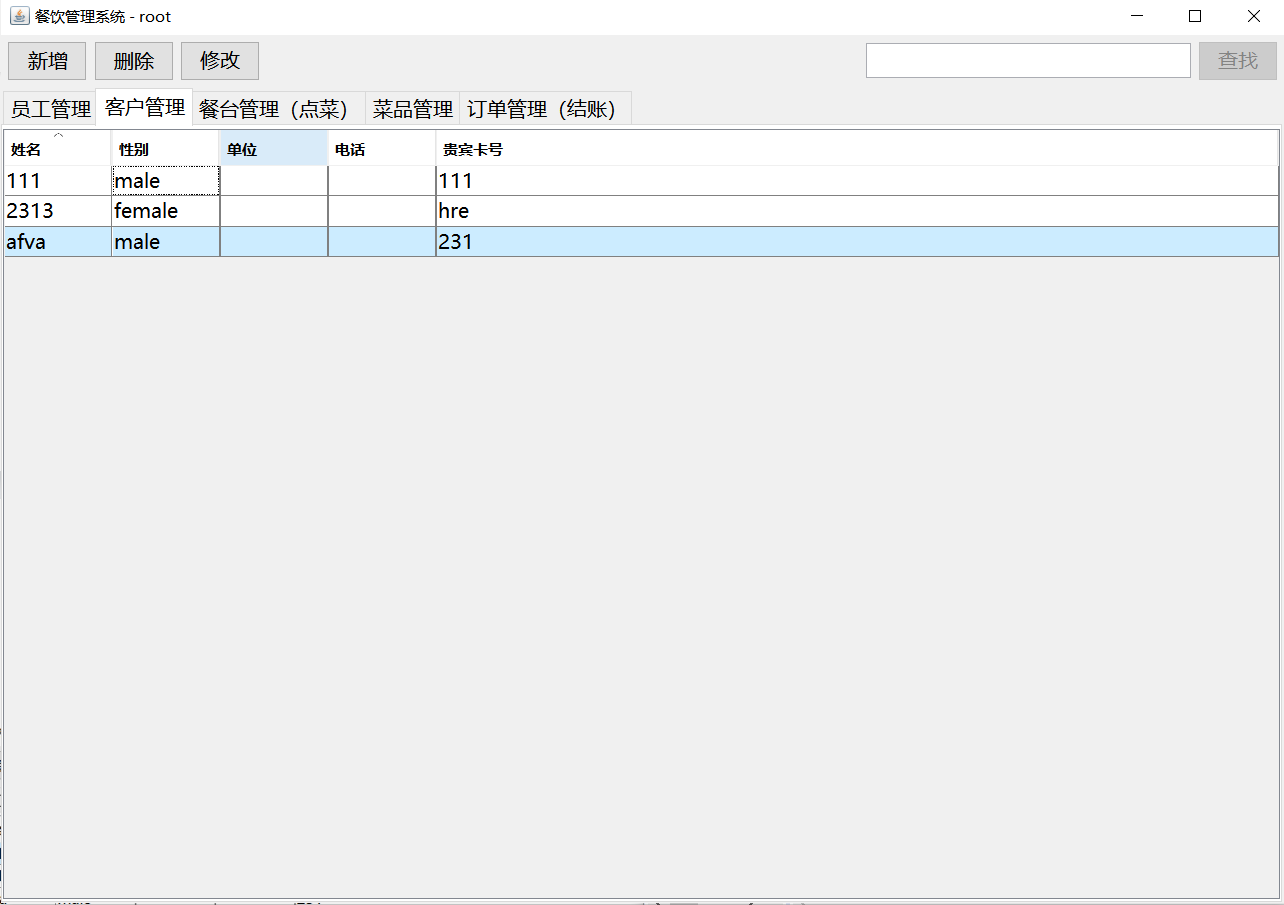
****

图3.7客户管理界面

3.3.7 用户接口6（餐台管理界面）设计

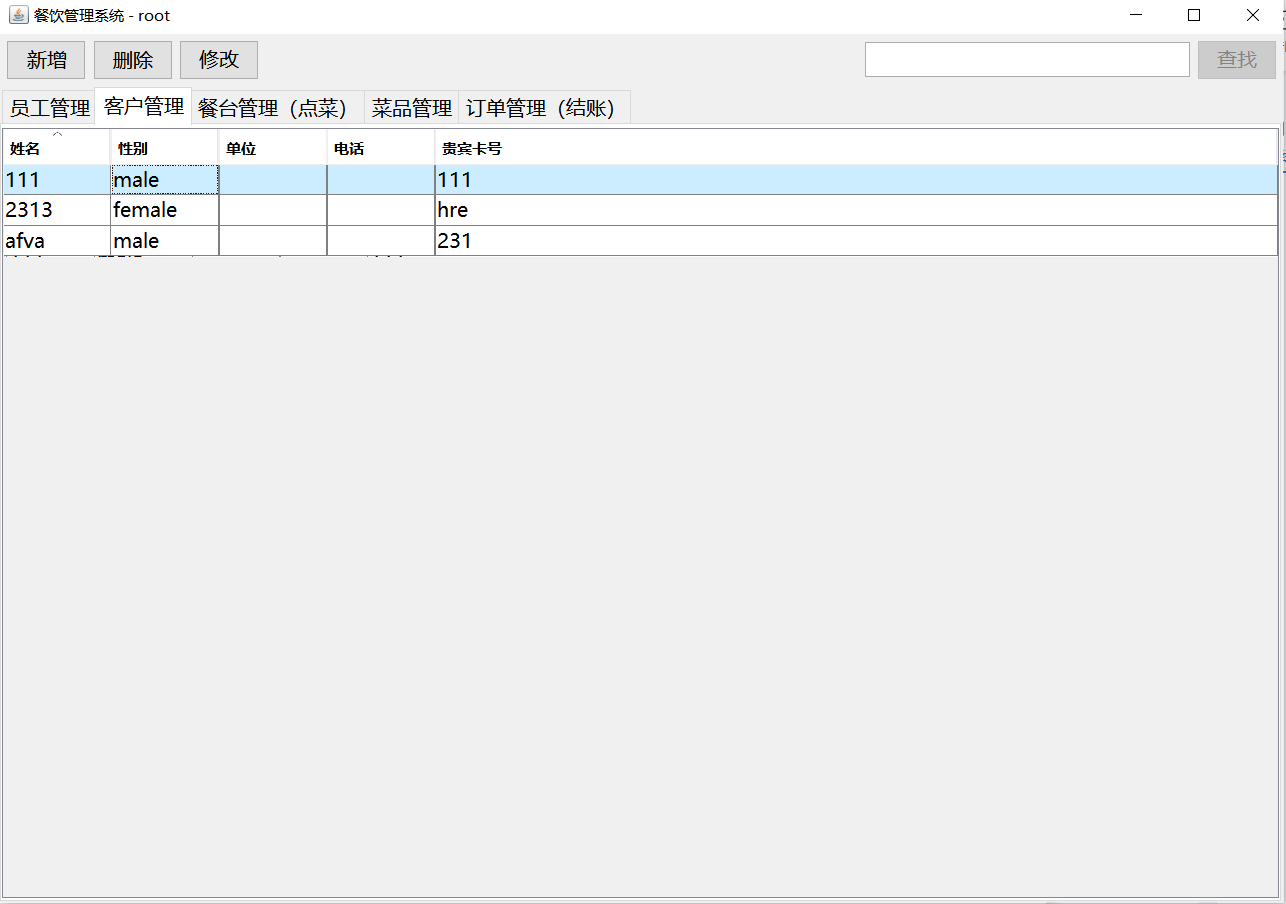


图3.8餐台管理界面

**界面主要控件：**

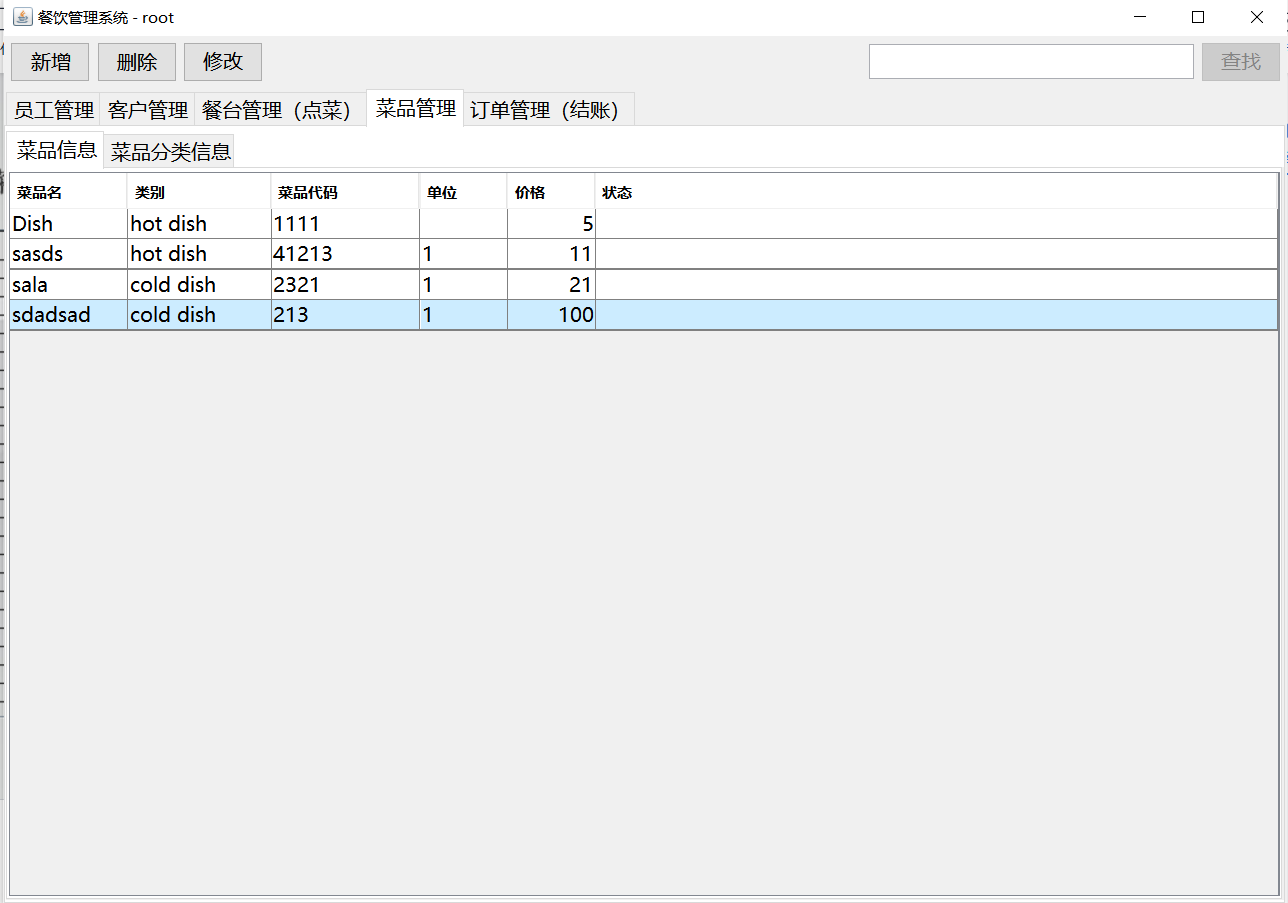
JButton，按钮，触发点击事件。

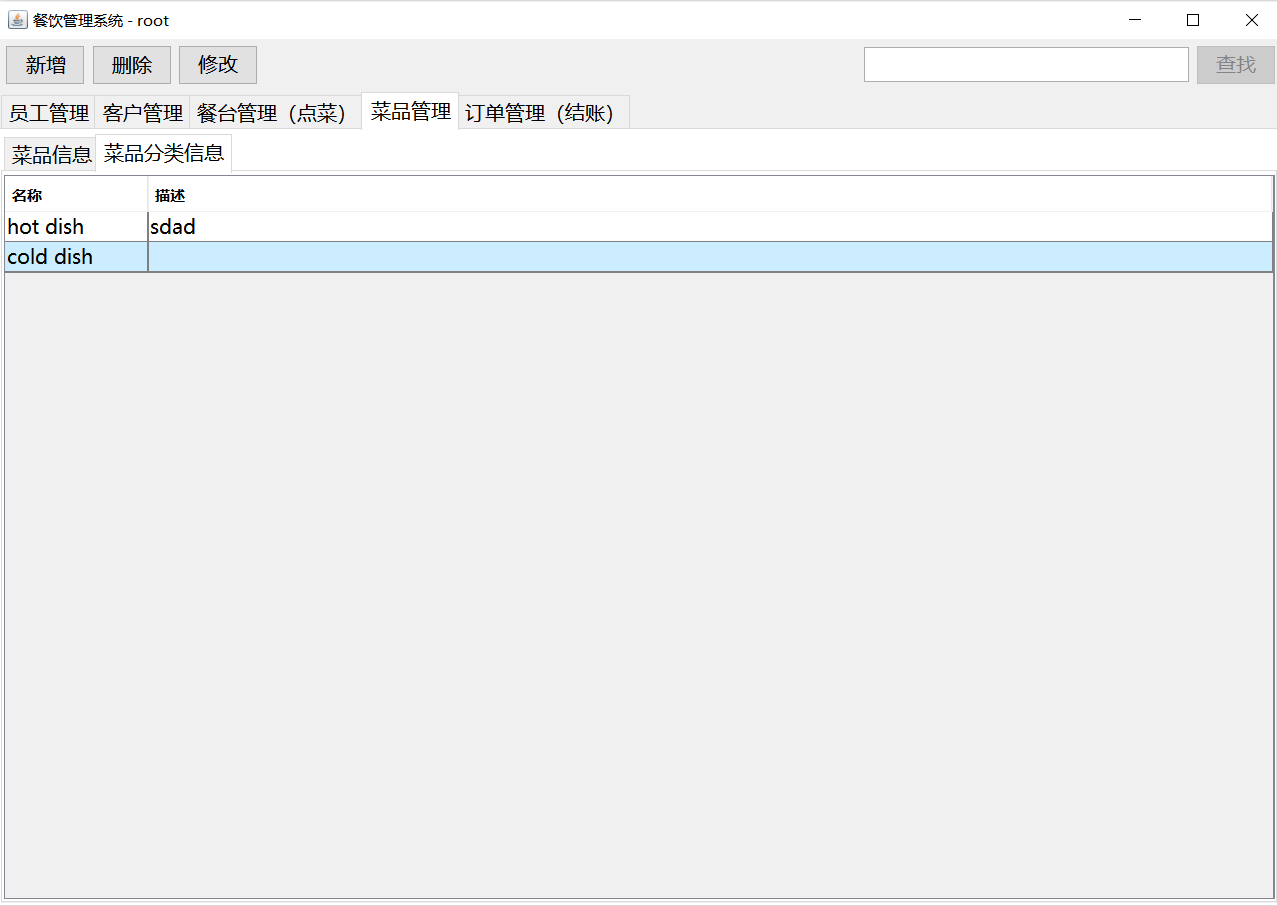
JTextField，输入框，接收餐台号。

**实现功能：**

1. 点击查询按钮，将对餐台进行查询。
2. 点击新增按钮，将对进行餐台的增添。
3. 点击表格中的某个餐台后再点击删除餐台，将对该餐台进行删除。
4. 点击表格，将对餐台信息进行修改。

3.3.8 用户接口7（菜品管理界面）设计

 图3.9.1菜品管理界面



## 图3.9.2菜品管理界面

**界面主要控件：**

JButton，按钮，触发点击事件。

JTextField，输入框，接收菜品id和菜品名。

**实现功能：**

1. 点击查询按钮，将对菜品进行查询。
2. 点击新增按钮，将对进行菜品的增添。
3. 点击表格中的某个菜品后再点击删除菜品，将对该菜品进行删除。
4. 点击表格，将对菜品信息进行修改。
5. 点击查询按钮，将对菜品分类进行查询。
6. 点击新增按钮，将对进行菜品类别的增添。
7. 点击表格中的某个菜品类别后再点击删除，将对该菜品类别进行删除。
8. 点击表格，将对菜品类别信息进行修改。

3.3.9 用户接口8（点菜管理界面）设计

**界面主要控件：**

界面1：

JButton，按钮，触发点击事件。

JTextField，输入框，接收客户id和就餐人数。

界面2：

JButton，按钮，触发点击事件。

JTextField，输入框，接收菜名。

JComboBox，下拉列表，对菜品信息进行筛选。

**实现功能：**

界面1：

1. 输入客户编号和就餐人数，点击开台，系统根据人数进行分配合适的餐台。
2. 点击点菜跳转至界面2

界面2：

1. 输入菜名和选择菜品类型，点击搜索，系统筛选出符合要求的菜品。
2. 点击点菜，选择菜品，点击下单，菜品将添加到订单中

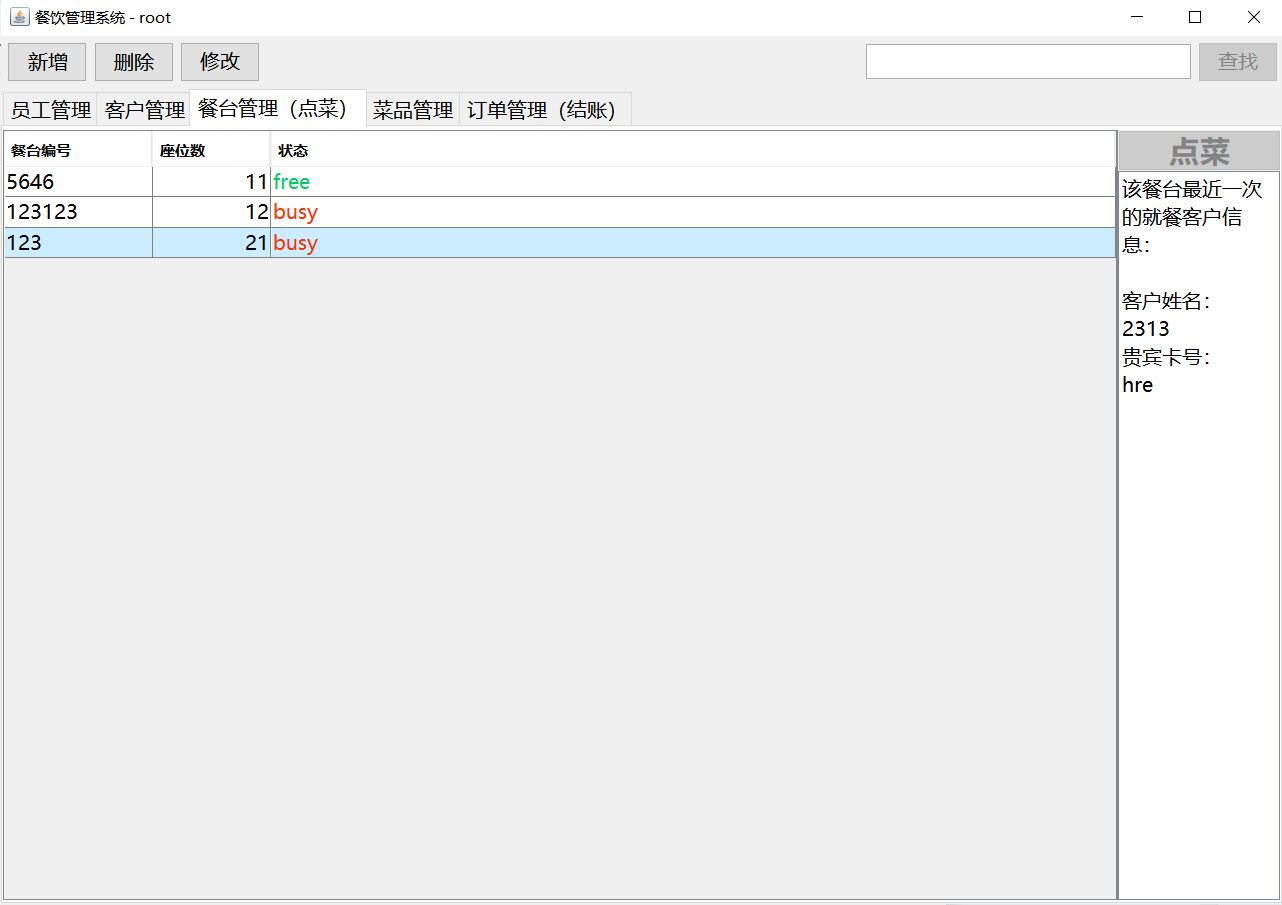


图3.10点菜管理界面（1）

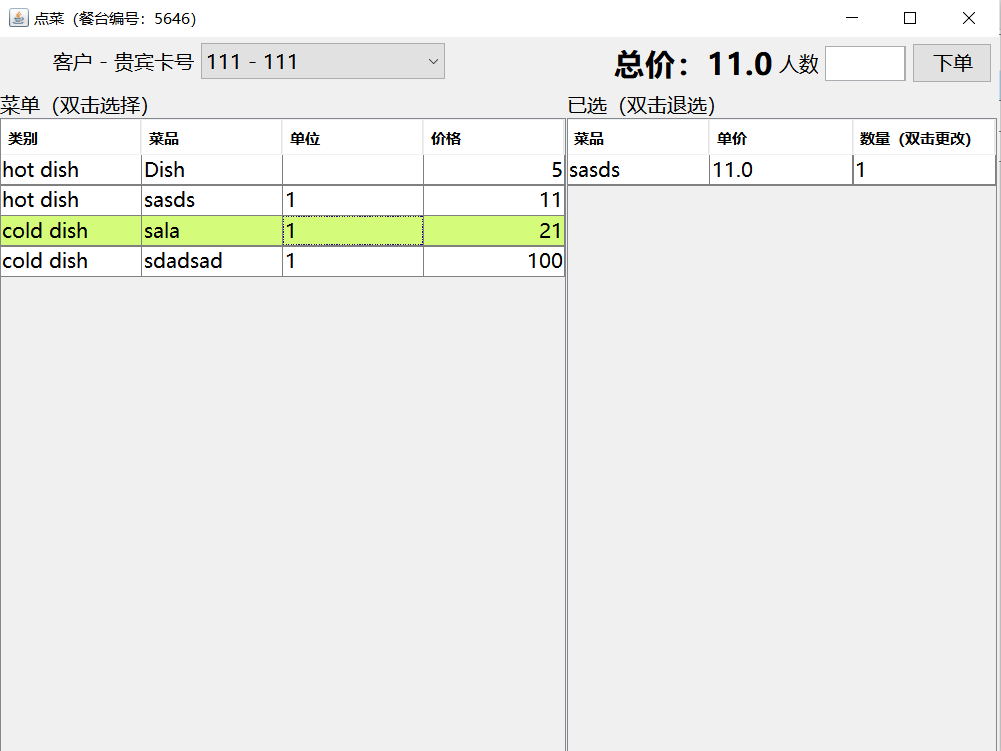


图3.10点菜管理界面（2）

3.3.10 用户接口9（结账管理界面）设计

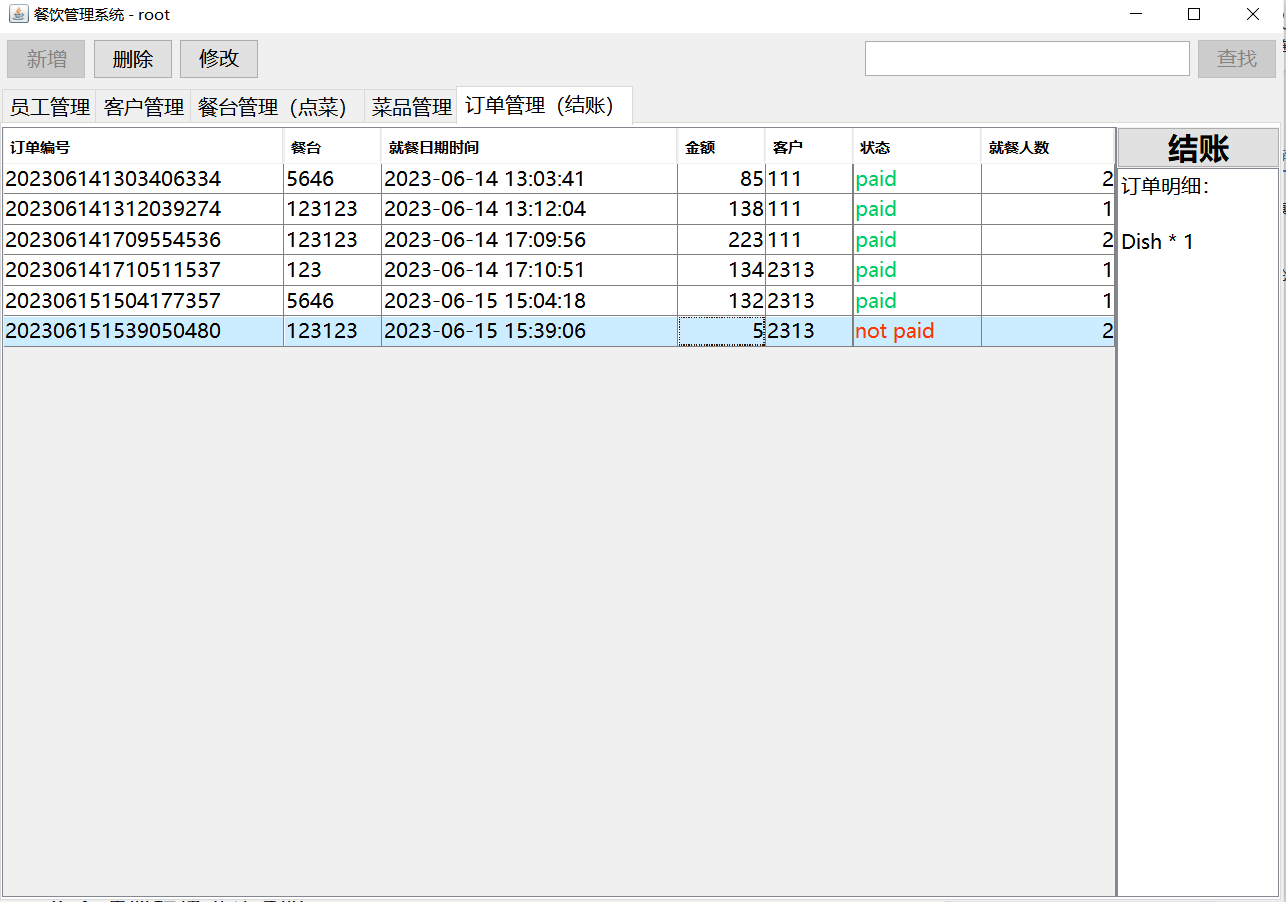
**界面主要控件：**

JButton，按钮，触发点击事件。

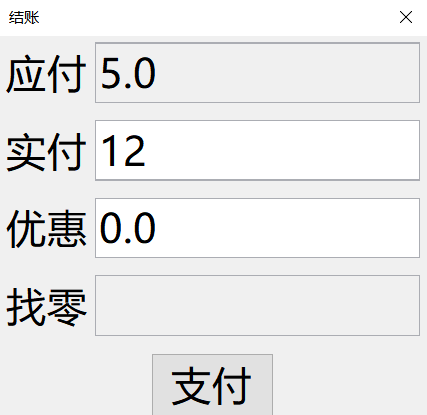
JTextField，输入框，接收订单号和客户id。

**实现功能：**

1. 点击查询按钮，将对订单进行查询。
2. 点击表格中的某个餐台后再点击删除订单，将对该订单进行删除。
3. 点击订单，将对订单进行结账操作。



## 图3.11结账管理界面



## 图3.11结账界面

## 3.4 登录模块设计说明

（1）功能：该模块主要是管理员对餐饮管理系统进行登录操作。

（2）测试要点：

1. 输入的账号或密码为空将提示账号或密码为空。

2. 输入的账号密码错误则提示账号密码有误。

（3）程序流程图如图3.12所示：

## wps

图3.12登录模块程序流程图

## 3.5 员工管理模块设计说明

（1）功能：实现管理员对员工的增删改查。

1. 新增员工，程序流程图如图3.13所示：

测试要点：

1. 点击新增员工按钮，弹出新增员工窗口。
2. 输入员工的信息，若员工信息无误且完整，则提示新增成功。
3. 若信息不完整，且信息有误，系统会给出提醒。

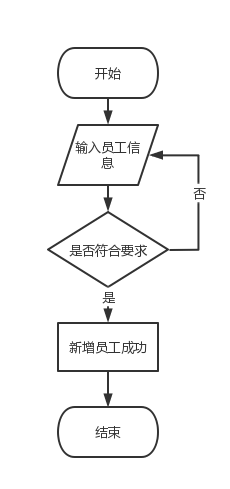


图3.13新增员工信息流程图

1. 删除员工，程序流程图如图3.14所示：



图3.14删除员工流程图

测试要点：

1. 点击表格中员工的信息，点击删除，系统提示删除成功。
2. 修改员工，程序流程图如图3.15所示：

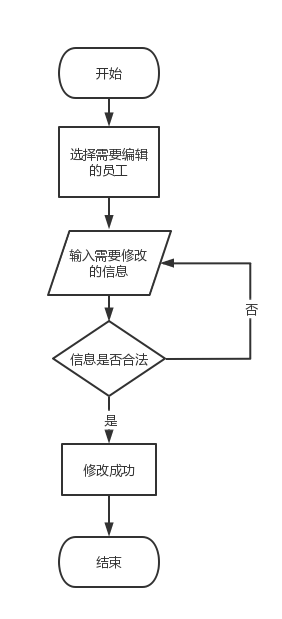


图3.15修改员工信息流程图

测试要点：

1. 点击表格中员工的信息，弹出新的界面。
2. 在新的界面中填写要修改的信息，若输入的信息合法则提示修改成功
3. 若输入的信息不合法，则提示信息不合法。
4. 查询员工，程序流程图如图3.16所示：

测试要点：

1. 在文本框中输入员工的姓名和id，点击查询按钮。
2. 查询信息将在表格中显示。

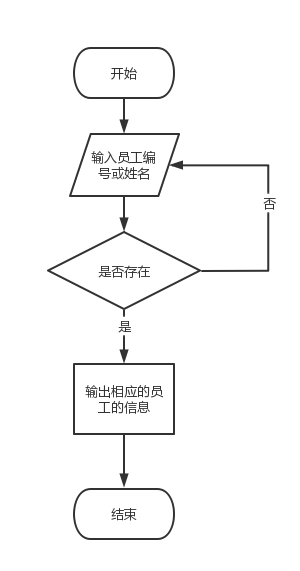


图3.16查询员工信息流程图

## 3.6 客户管理模块设计说明

（1）功能：实现管理员对客户的增删改查。

1. 新增客户，程序流程图如图3.17所示：

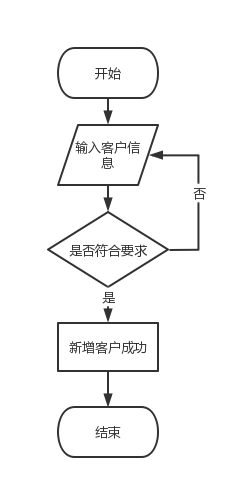


图3.17新增客户流程图

测试要点：

1. 点击新增客户按钮，弹出新增客户窗口。
2. 输入客户的信息，若客户信息无误且完整，则提示新增成功。
3. 若信息不完整，且信息有误，系统会给出提醒。
4. 删除客户，程序流程图如图3.18所示：



图3.18删除客户流程图

测试要点：

1. 点击表格中客户的信息，点击删除，系统提示删除成功。
2. 修改客户，程序流程图如图3.19所示：

测试要点：

1. 点击表格中客户的信息，弹出新的界面。
2. 在新的界面中填写要修改的信息，若输入的信息合法则提示修改成功。
3. 若输入的信息不合法，则提示信息不合法。
4. 查询客户，程序流程图如图3.20所示：

测试要点：

1. 在文本框中输入客户的姓名和id，点击查询按钮。
2. 查询信息将在表格中显示。

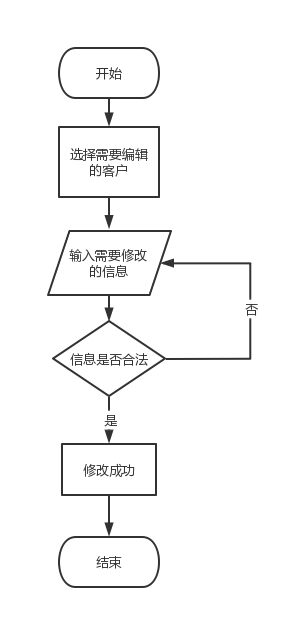
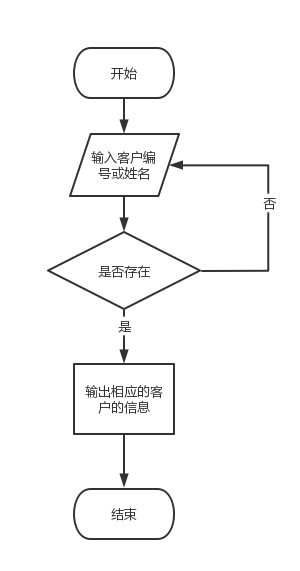
 

图3.19修改客户流程图 示：

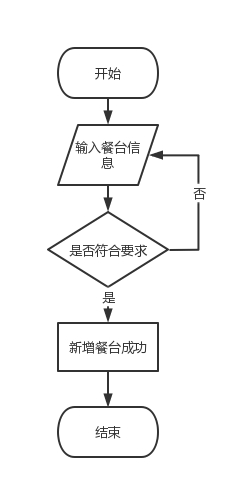


图3.21新增餐台流程图

测试要点：

1. 点击新增餐台按钮，弹出新增餐台界面。
2. 输入餐台的座位数，若输入为数字且合法，则新增餐台成功，否则提示新增失败。
3. 删除餐台，程序流程图如图3.22所示：



图3.22删除餐台流程图

测试要点：

1. 点击餐台信息，点击删除按钮，系统提示删除成功
2. 修改餐台，程序流程图如图3.23所示：

测试要点：

1. 点击表格中餐台的信息，弹出新的界面。
2. 在新的界面中填写要修改的信息，若输入的信息合法则提示修改成功。
3. 若输入的信息不合法，则提示信息不合法。
4. 查询餐台，程序流程图如图3.24所示：

测试要点：

1. 在文本框中输入餐台号，点击查询按钮。
2. 查询信息将在表格中显示。

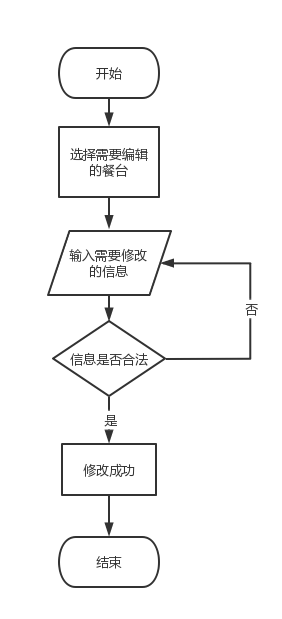
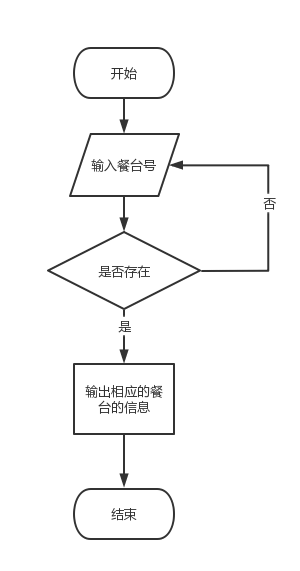
 

图3.23修改餐台流程图 图3.24查询餐台流程图

## 3.7菜品管理模块设计说明

（1）功能：实现管理员对菜品信息的增删改查

1. 新增菜品，程序流程图如图3.25所示：

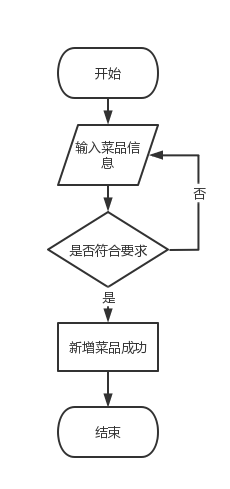


图3.25新增菜品程序流程图

测试要点：

1. 点击新增菜品按钮，弹出新增菜品窗口。
2. 输入菜品的信息，若菜品信息无误且完整，则提示新增成功。
3. 若信息不完整，且信息有误，系统会给出提醒。
4. 删除菜品，程序流程图如图3.26所示：



图3.26删除菜品流程图

测试要点：

1. 点击菜品信息，点击删除按钮，系统提示删除成功。
2. 修改菜品，程序流程图如图3.27所示：

测试要点：

1. 点击表格中餐台的信息，弹出新的界面。
2. 在新的界面中填写要修改的信息，若输入的信息合法则提示修改成功。
3. 若输入的信息不合法，则提示信息不合法。
4. 查询菜品，程序流程图如图3.28所示：

测试要点：

1. 在文本框中输入菜品名或菜品id，点击查询按钮。
2. 查询信息将在表格中显示。

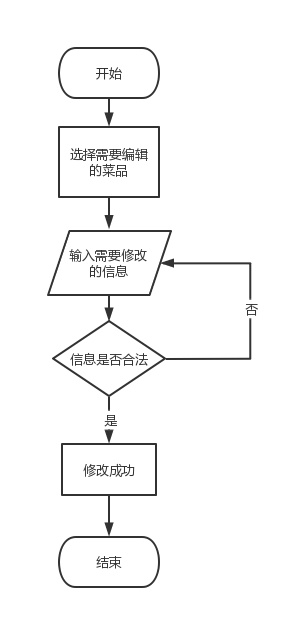
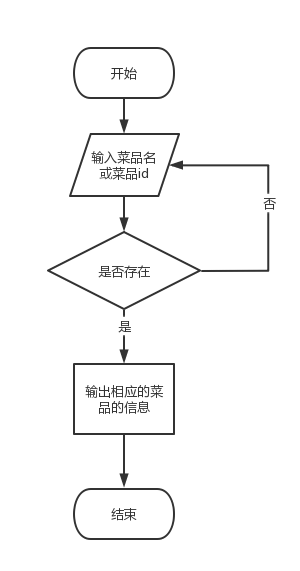
 

图3.27删除菜品程序流程图 图3.28查询菜品程序流程图

## 3.8 点单管理模块设计说明

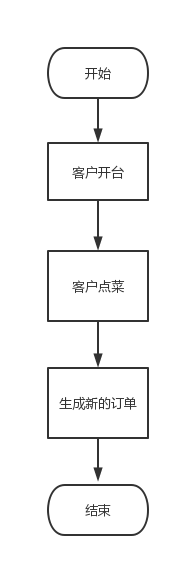
（1）功能：实现管理员对客户进行开台，点单操作。

（2）测试要点：

1. 输入客户的客户号以及人数，为客户开台。

2. 客户选菜，生成新的订单。

（3）程序流程图如图3.29所示：



3.29点菜管理程序流程图

## 3.9 结账管理模块设计说明

（1）功能：实现管理员对订单进行结账管理。

（2）测试要点：

1. 在输入框中输入订单编号或客户编号，查询的订单将在表格中显示。

2. 点击订单，跳转至结账界面，将会根据客户的身份对客户进行优惠。

3. 在新的界面输入金额，找零

（3）程序流程图如图3.30所示：

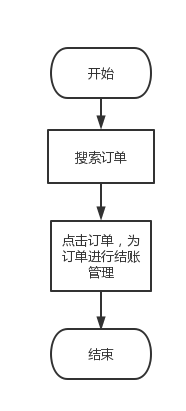


图3.30订单管理流程图

# 4 数据库设计

## 4.1 数据库环境说明

数据库版本：MySQL 5.7

运行环境：Windows 10

## 4.2 数据库的命名规则

数据库名：restaurant\_management

数据表命名规范：

（1）采用26个英文字母（区分大小写）和0-9的自然数（经常不需要）加上下划线’\_’组成，命名简洁明确，多个单词用下划线’\_’分隔。

（2）全部小写命名，禁止出现大写。

（3）禁止使用数据库关键字，如：name,time, datetime,password等。

（4）表名称不应该取得太长（一般不超过三个英文单词）。

（5）表的名称一般使用名词或者动宾短语。

（6）用单数形式表示名称。

（7）表必须填写描述信息（使用SQL语句建表时）。

字段命名规范：

（1）采用26个英文字母（区分大小写）和0-9个自然数（经常不需要）加下划线’\_’组成，命名简洁明确，多个单词用下划线’\_’分隔。

（2）全部小写命名，禁止出现大写。

（3）字段必须填写描述信息。

（4）禁止使用数据库关键字。

（5）字段名称一般采用名词或动宾短语。

（6）采用字段的名称必须是易于理解，一般不超过三个英文单词。

（7）在命名表的列时，不要重复表的名称。

（8）不要在列的名称中包含数据类型。

（9）字段命名使用完整命名，禁止缩写。

## 4.3 逻辑结构设计

关系模型（主键用下划线标注，双下划线表示既是主键又是外键）

管理员（用户名，密码）

客户（客户编号，客户姓名，性别，出生日期，联系方式，身份）

菜品（菜品编号，菜品名，菜品价格，菜品类别，菜品口味，烹饪方式）

员工（员工编号，员工姓名，员工性别，出生日期，联系方式，就职日期）

餐台（餐台编号，餐台状态，餐台人数）

餐台消费记录（订单编号，餐台编号）

客户消费记录（订单编号，消费时间，客户编号）

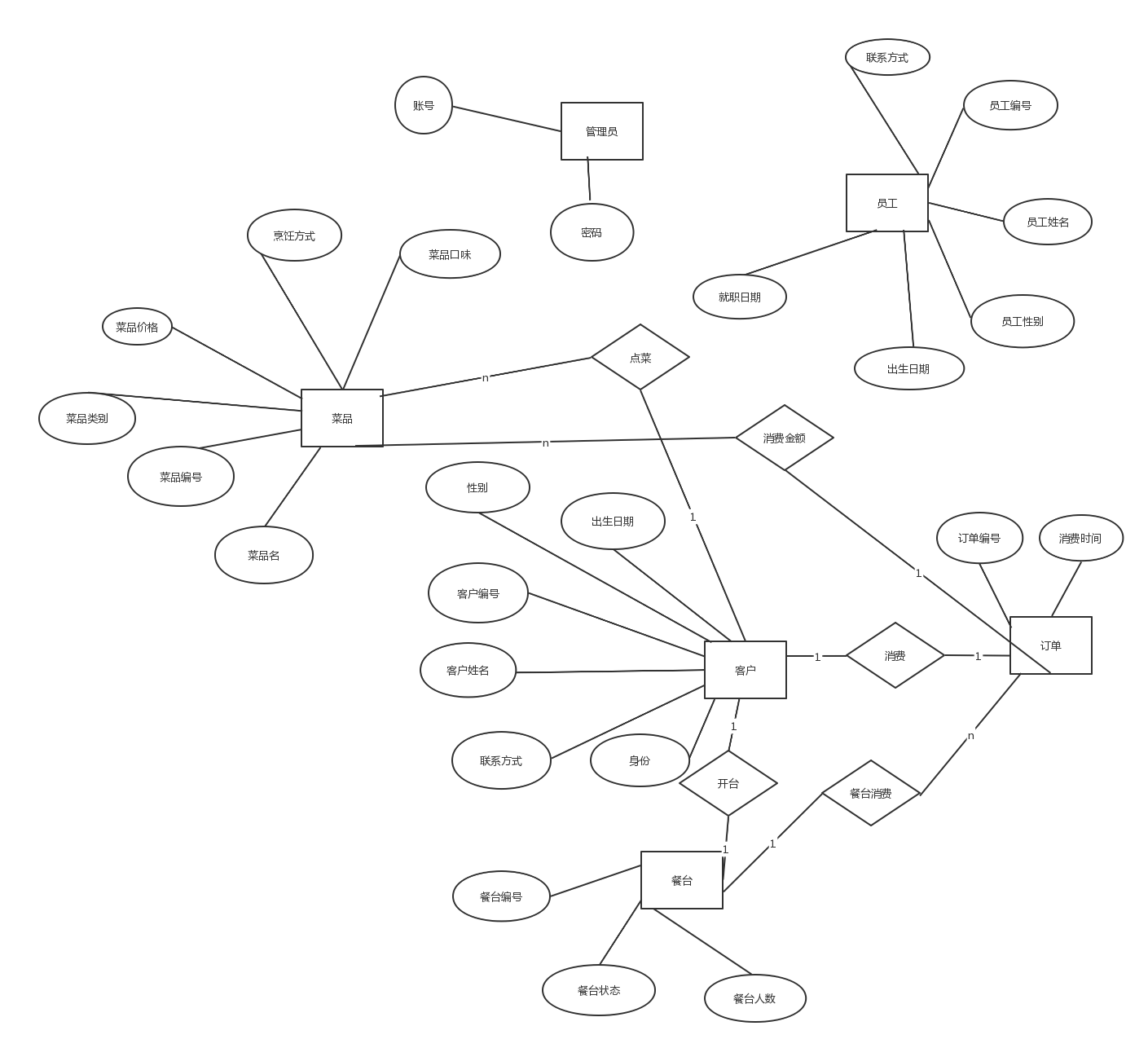
点单（订单编号，菜品编号，数量）

订单（订单编号，餐台编号，客户编号，开台时间，消费金额，订单状态，人数）

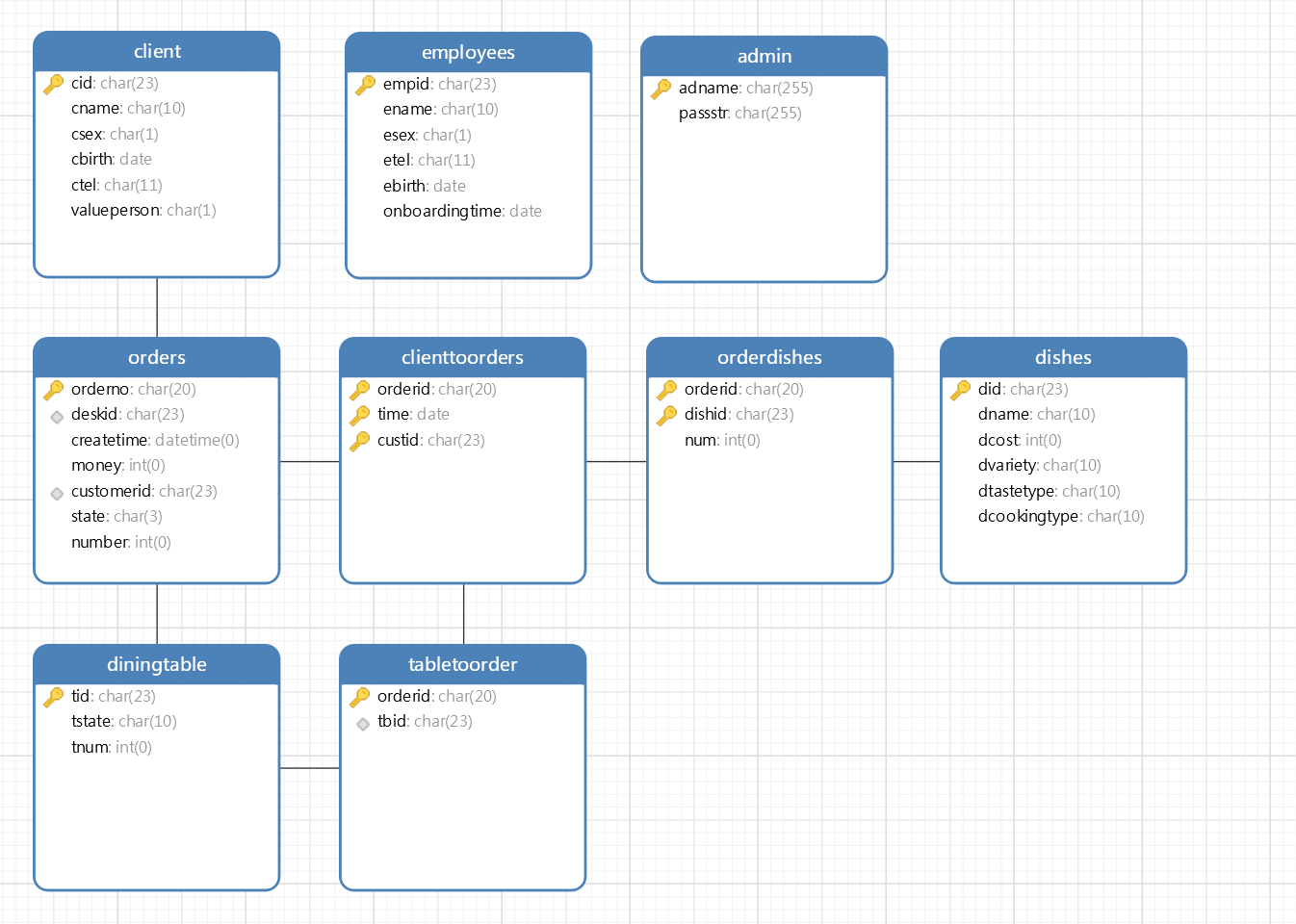
范式分析

所列关系均为第三范式，因为它消除了非主属性对主属性的传递依赖

E-R图



逻辑模型图



## 4.4 物理结构设计

### 4.4.1 建库语言

**drop** database if **exists** restaurant\_management;

**create** database restaurant\_management;

use restaurant\_management;

**create** **table** if **not** **exists** employee(

id **int** **primary** **key** auto\_increment,

`name` **varchar**(40) **not** **null**,

sex **varchar**(1) **not** **null** **check**(sex **in** ('男', '女')),

birthday **date**,

identityId **varchar**(18),

address **varchar**(40),

tel **varchar**(11),

position **varchar**(40) **not** **null**,

freeze **varchar**(1) **not** **null**);

**create** **table** if **not** **exists** customer(

id **int** **primary** **key** auto\_increment,

`name` **varchar**(40) **not** **null**,

sex **varchar**(1) **not** **null**,

company **varchar**(40),

tel **varchar**(11),

cardId **varchar**(40) **not** **null**,

**unique**(cardId));

**create** **table** if **not** **exists** dishType(

id **int** **primary** **key** auto\_increment,

`name` **varchar**(40) **not** **null**,

`describe` **varchar**(40),

**unique**(`name`));

**create** **table** if **not** **exists** dish(

id **int** **primary** **key** auto\_increment,

`name` **varchar**(40) **not** **null**,

dishTypeId **int** **not** **null** **references** dishType(id),

`code` **varchar**(40) **not** **null**,

unit **varchar**(40),

price **double** **not** **null**,

`status` **varchar**(2) **not** **null** **check**(`status` **in** ('', '售罄')),

**unique**(`name`));

**create** **table** if **not** **exists** `**table**`(

id **int** **primary** **key** auto\_increment,

`**no**` **varchar**(40) **not** **null**,

seating **int** **not** **null**,

`status` **varchar**(2) **not** **null** **check**(`status` **in** ('空闲', '忙碌')),

**unique**(`**no**`));

**create** **table** if **not** **exists** `**order**`(

id **int** **primary** **key** auto\_increment,

orderNo **varchar**(40) **not** **null**,

tableId **int** **references** `**table**`(id),

createtime datetime **not** **null**,

money **double** **not** **null**,

customerId **int** **references** customer(id),

`status` **varchar**(3) **not** **null** **check**(`status` **in** ('已支付', '未支付')),

`number` **int** **not** **null**);

**create** **table** if **not** **exists** orderitem(

id **int** **primary** **key** auto\_increment,

orderId **int** **references** `**order**`(id),

dishId **int** **references** dish(id),

amount **int** **not** **null**);

# 5 测试用例设计

表5.1 登录-注册测试用例

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 功能 | 用例编号 | 操作步骤  （输入条件） | 测试数据 | 预期结果 | 实际结果/判定 |
| 登录 | yl-001 | 输入‘用户名’和‘密码’ | Root，123456 | 登录成功 | 通过 |
| yl-002 | 2942990081，1234567 | 登录失败 | 通过 |
| yl-003 | 29429900811，1237236182 | 登录失败 | 通过 |
| yl-004 | 2942990081,空 | 登录失败 | 通过 |
| yl-005 | 空,1237236182 | 登录失败 | 通过 |
| yl-006 | 2942990081,一个空格 | 登录失败 | 通过 |
| yl-007 | 一个空格，1237236182 | 登录失败 | 通过 |
| yl-008 | 2942990081，密码长度超过32 | 登录失败 | 通过 |
| yl-009 | 用户名超过36，1237236182 | 登录失败 | 通过 |

表5.2 员工管理测试用例

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 功能 | 用例编号 | 操作步骤  （输入条件） | 测试数据 | 预期结果 | 实际结果/判定 |
| 增加员工 | y2-001 | 点击添加按  钮，输入‘姓  名’，‘性别’，‘出生日期’，‘电话号码’，‘就职日期’ | Anish，female，15273751972,  2002-08-03,2022-06-18 | 新增成功 | 通过 |
| 删除员工 | y2-002 | 点击删除按钮 | Anish | 删除成功 | 通过 |
| 修改员工 | y2-003 | 点击表格 | Anish->A | 修改成功 | 通过 |
| 查询员工 | y2-004 | 输入员工编  号或姓名，点击查询  按钮 | 220618545 | 查询成功 | 通过 |

表5.3 客户管理测试用例

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 功能 | 用例编号 | 操作步骤  （输入条件） | 测试数据 | 预期结果 | 实际结果/判定 |
| 增加客户 | y3-001 | 点击添加按  钮，输入‘姓  名’，‘性别’，‘出生日期’，‘电话号码’，‘会员’ | asd，female，2022-06-22，12345678901，是 | 新增成功 | 通过 |
| 删除客户 | y3-002 | 点击删除按钮 | asd | 删除成功 | 通过 |
| 修改客户 | y3-003 | 点击表格 | female->male | 修改成功 | 通过 |
| 查询客户 | y3-004 | 输入员工编  号或姓名，点击查询  按钮 | 220622787 | 查询成功 | 通过 |

表5.4 菜品管理测试用例

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 功能 | 用例编号 | 操作步骤  （输入条件） | 测试数据 | 预期结果 | 实际结果/判定 |
| 增加菜品 | y4-001 | 点击添加按  钮，输入‘菜品名’，‘类别’，‘菜品代码’，‘单位’，‘价格’，‘状态’ | Dish，hot dish，1111，2，5 | 新增成功 | 通过 |
| 删除菜品 | y4-002 | 点击删除按钮 | Dish | 删除成功 | 通过 |
| 修改菜品 | y4-003 | 点击表格 | 5->4 | 修改成功 | 通过 |
| 查询菜品 | y4-004 | 输入菜品编  号或菜名，点击查询  按钮 | 1111 | 查询成功 | 通过 |

表5.5 餐台管理测试用例

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 功能 | 用例编号 | 操作步骤  （输入条件） | 测试数据 | 预期结果 | 实际结果/判定 |
| 增加餐台 | y5-001 | 点击添加按  钮，输入’餐台编号’，‘座位数’ | 5646，10 | 新增成功 | 通过 |
| 删除餐台 | y5-002 | 点击删除按钮 | 5646 | 删除成功 | 通过 |
| 修改餐台 | y5-003 | 点击表格 | 10->20 | 修改成功 | 通过 |
| 查询餐台 | y5-004 | 输入菜品编  号或菜名，点击查询  按钮 | 5646 | 查询成功 | 通过 |

表5.6 点菜管理测试用例

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 功能 | 用例编号 | 操作步骤  （输入条件） | 测试数据 | 预期结果 | 实际结果/判定 |
| 选菜 | y6-001 | 双击选中菜品与数量 | Dish，1 | 选菜成功 | 通过 |
| 生成订单 | y6-002 | 点菜后点击保存 | 无 | 操作成功 | 通过 |

表5.7 结账管理测试用例

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 功能 | 用例编号 | 操作步骤  （输入条件） | 测试数据 | 预期结果 | 实际结果/判定 |
| 结账操作 | Y7-001 | 点击订单结账后输入金额 | 500 | 支付成功 | 通过 |

# 附录

## 附录1 程序运行结果

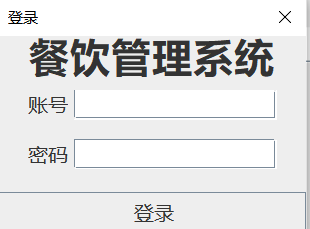


图1 登录界面

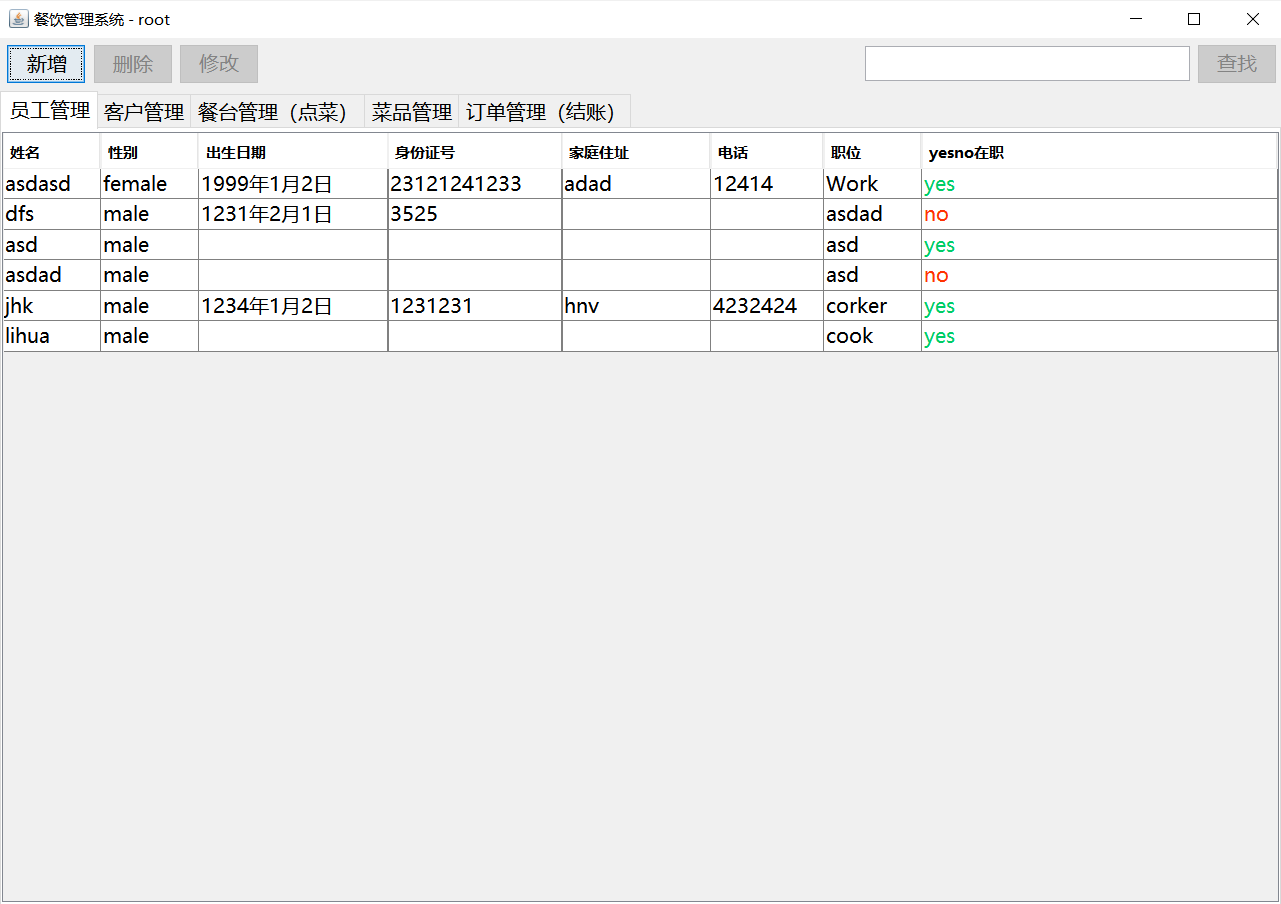


图2 系统主界面

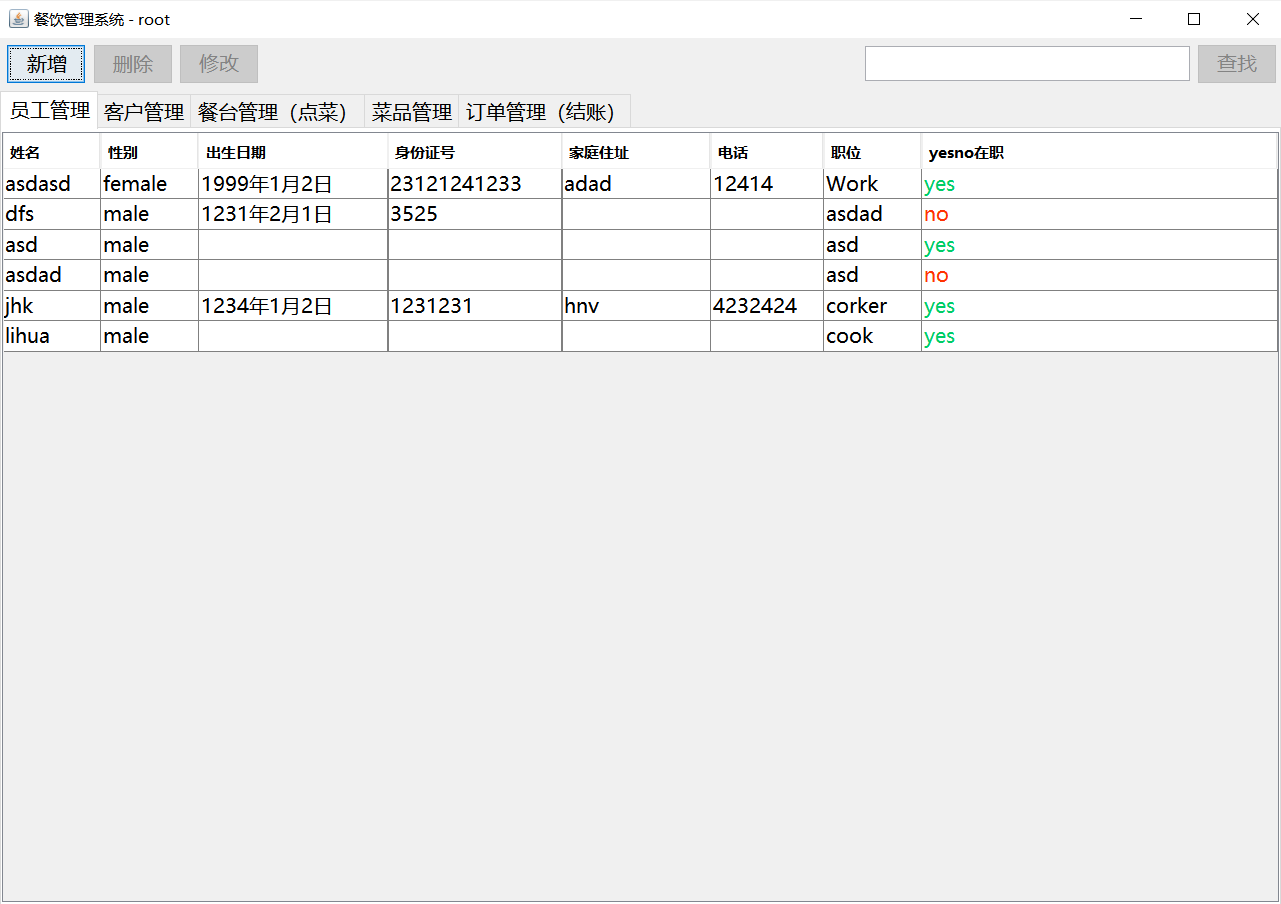


图3 员工管理界面

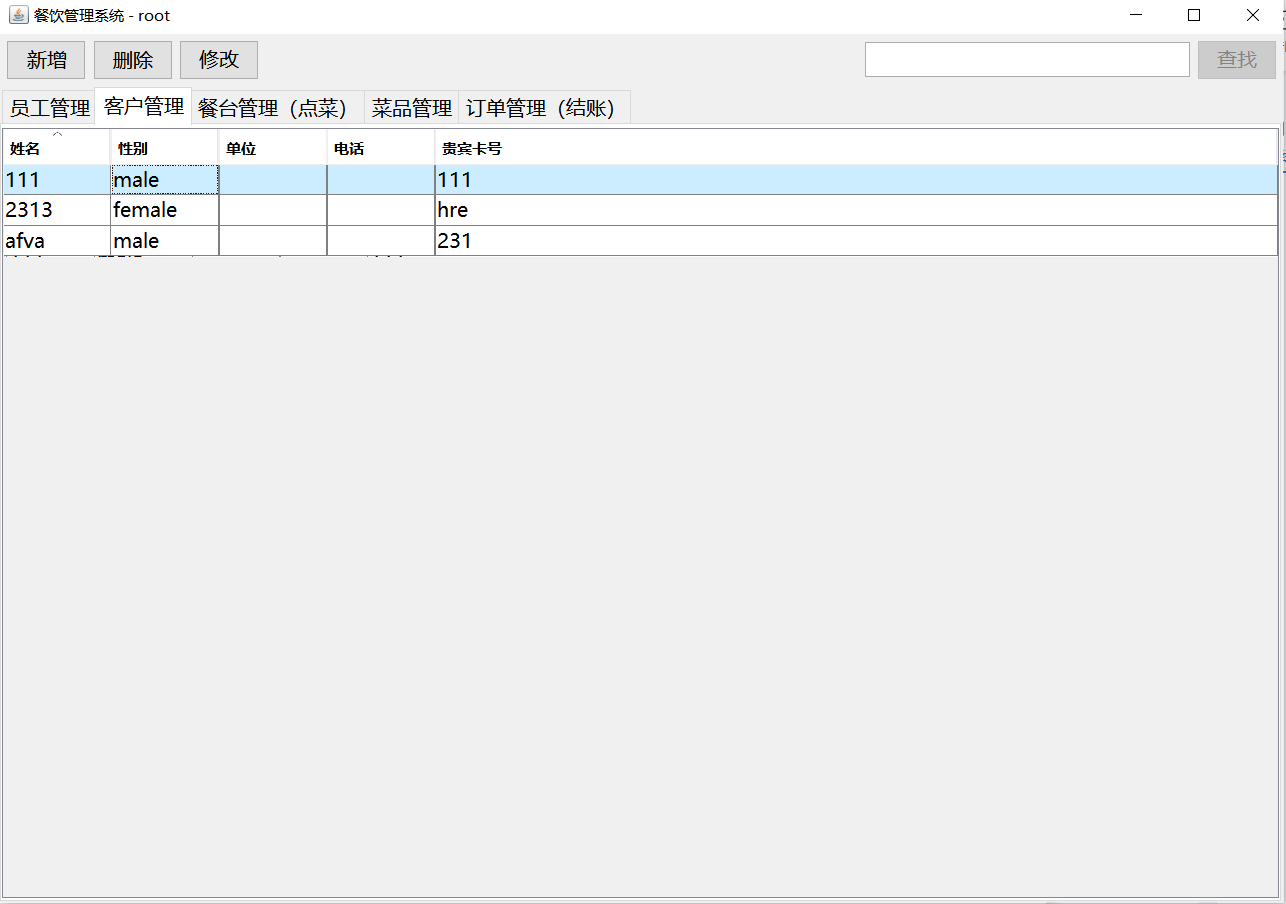


图4 客户管理界面

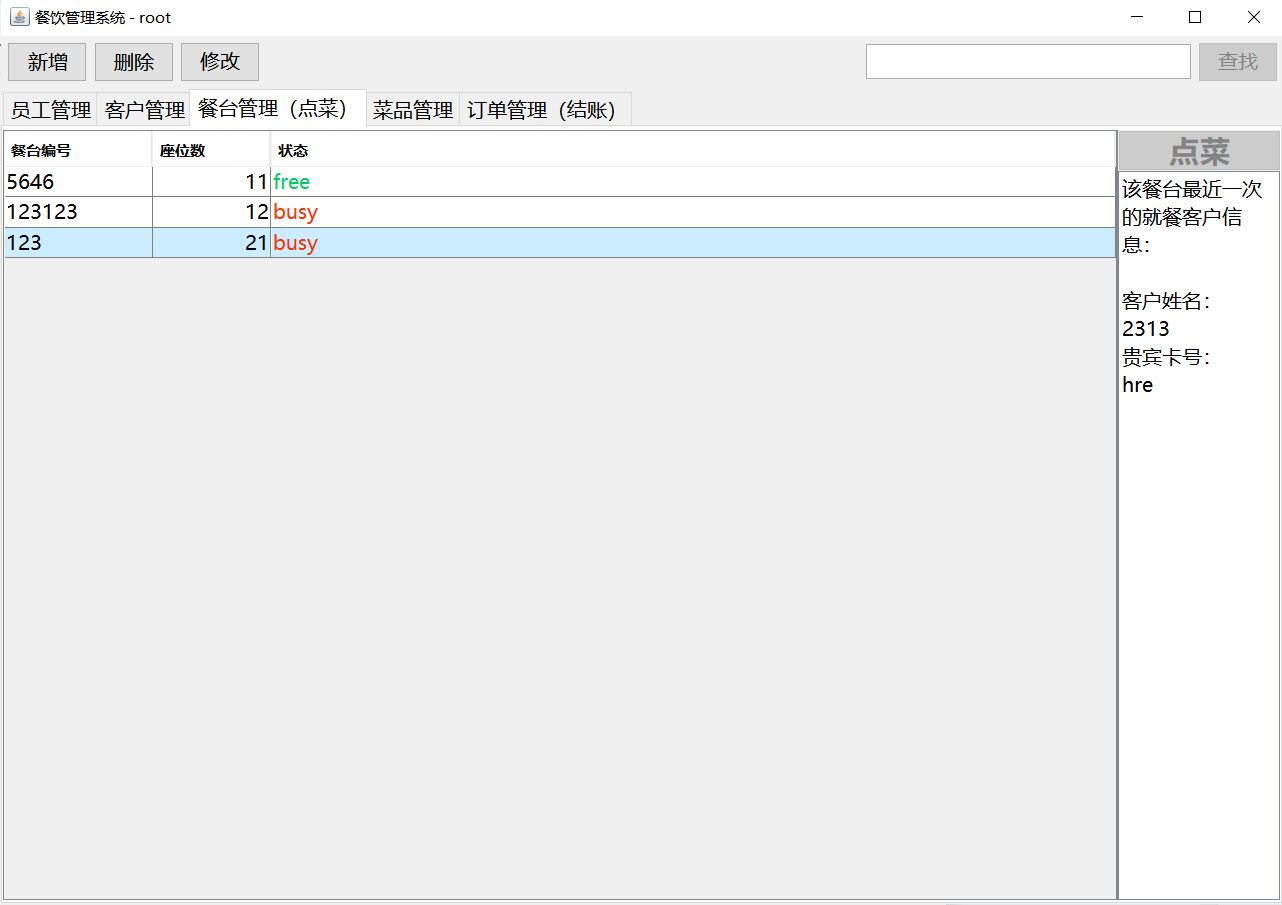


图5 餐台管理界面

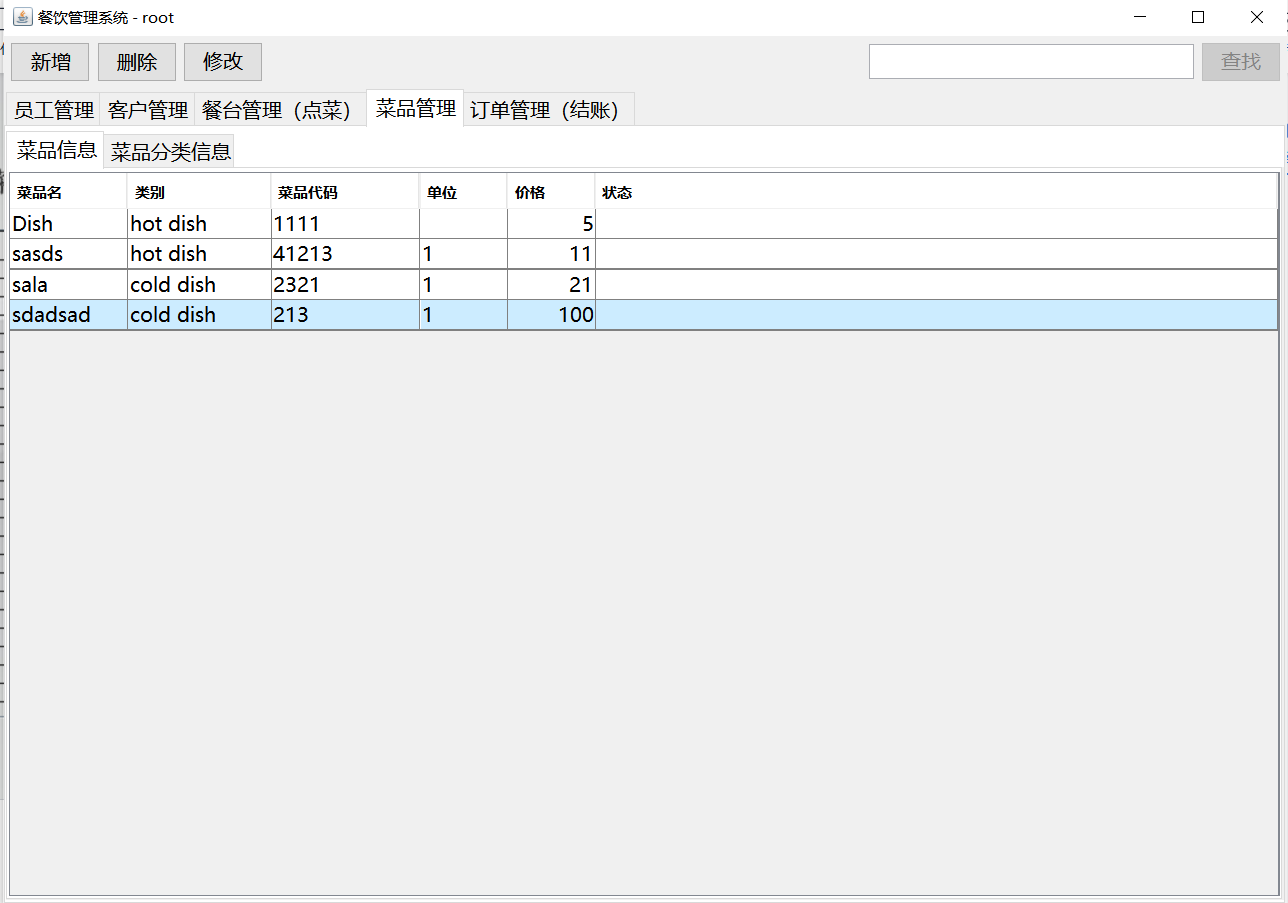


图6 菜品管理界面

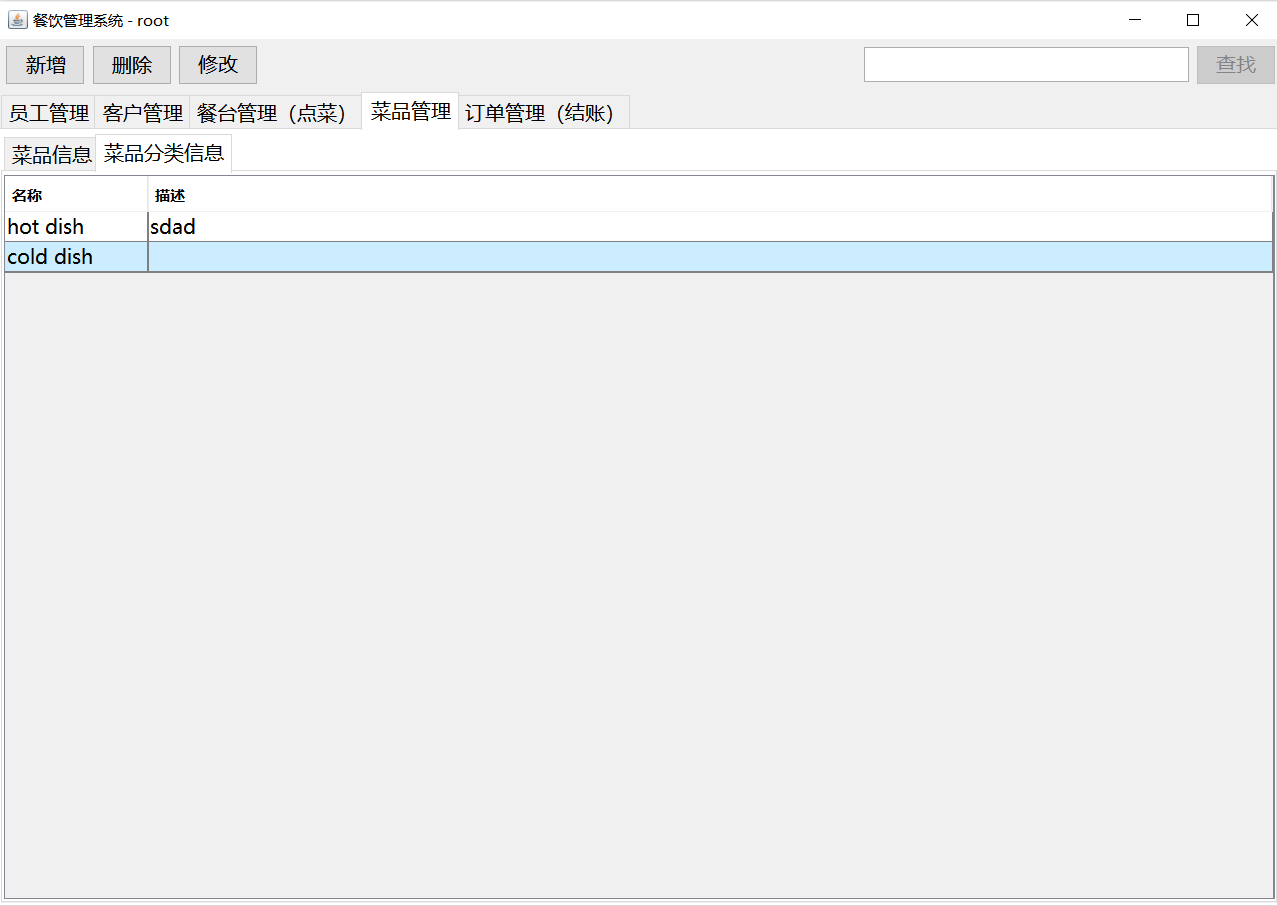


图7 菜品分类管理界面

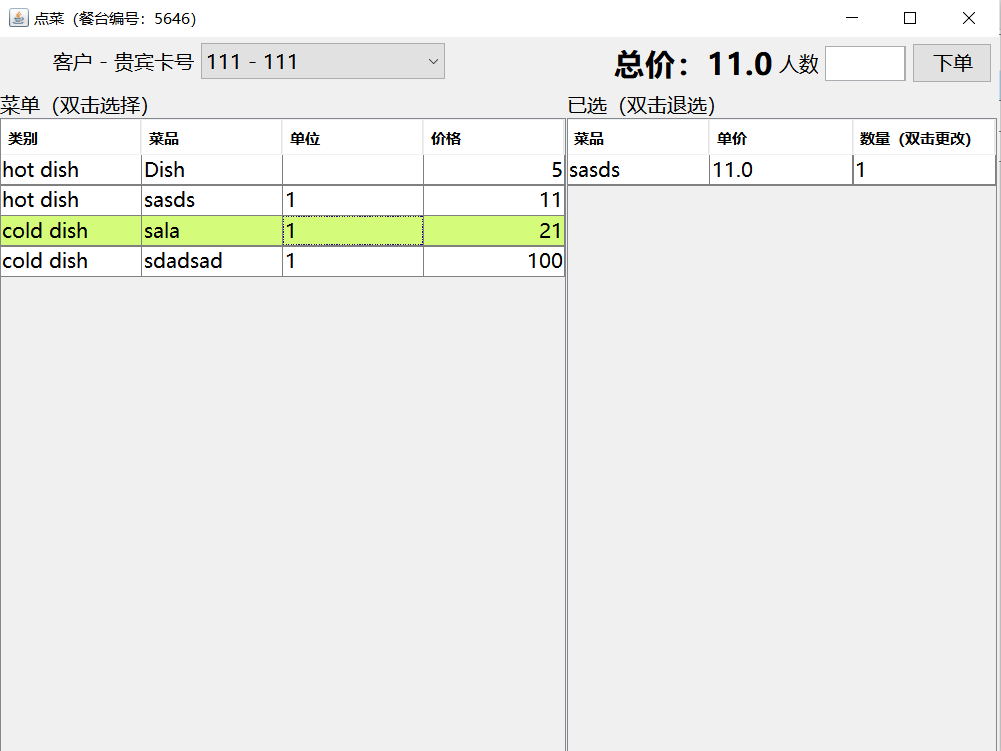


图8 点菜界面

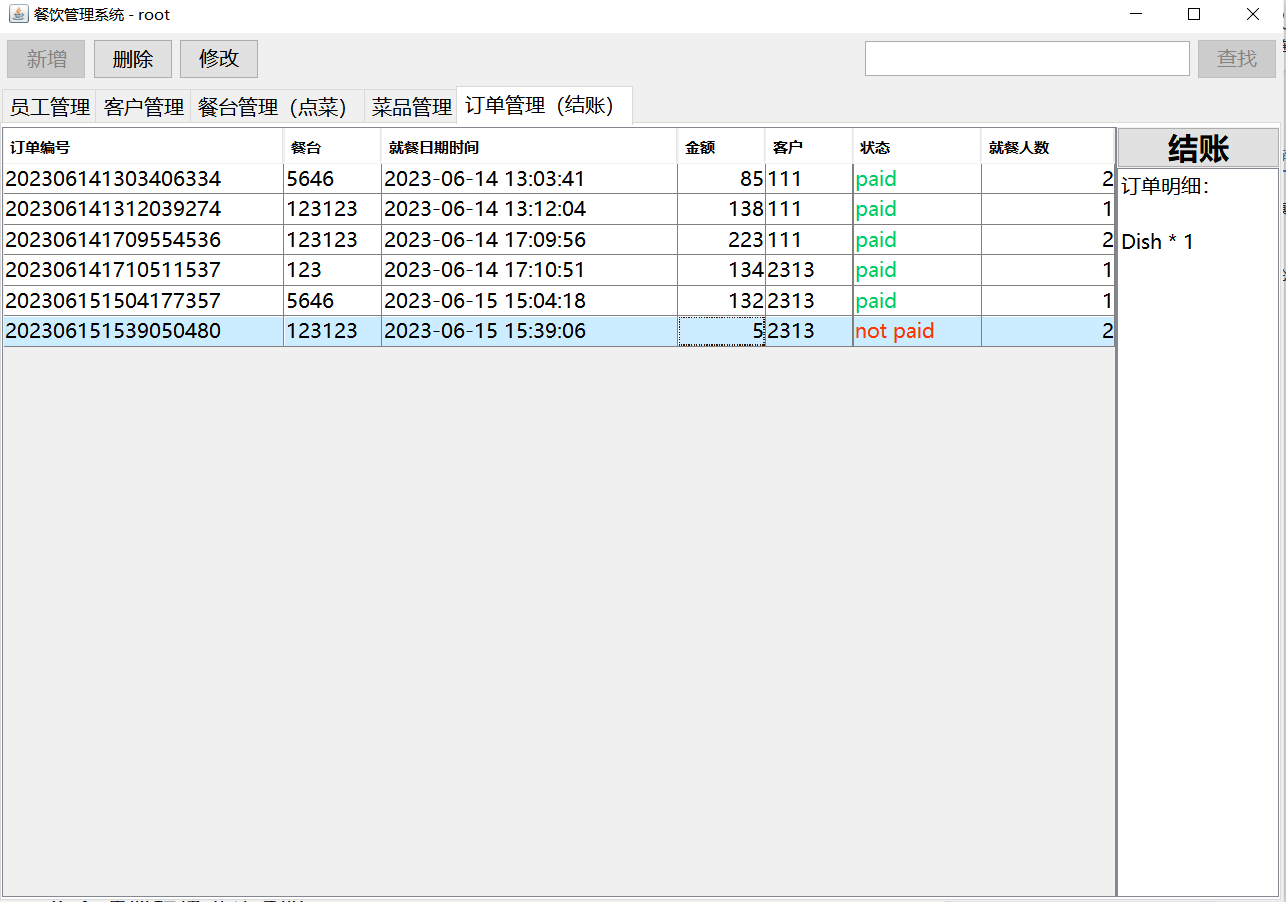


图9 结账管理界面

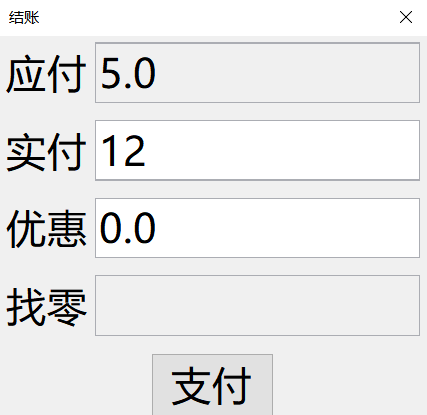


图10 结账界面

## 附录2 程序源代码

**dao包**

package dao;

import java.sql.PreparedStatement;

import java.sql.ResultSet;

import java.sql.Statement;

import java.util.ArrayList;

import java.util.List;

import entity.Customer;

import tool.DataOperation;

//定义抽象类 CustomerOperation，用于定义与客户相关的数据库操作方法

public abstract class CustomerOperation {

// 从数据库中读取所有客户的信息，并返回一个列表

public static List<Customer> load() {

List<Customer> list = new ArrayList<Customer>();

try {

// 创建 PreparedStatement 对象，执行查询语句，获取结果集

PreparedStatement preparedStatement = DataOperation.connection.prepareStatement("select \* from customer");

ResultSet resultSet = preparedStatement.executeQuery();

// 遍历结果集，将每个客户信息封装为 Customer 对象，添加到列表中

while (resultSet.next()) {

Customer customer = new Customer(

resultSet.getInt(1),

resultSet.getString(2),

resultSet.getString(3),

resultSet.getString(4),

resultSet.getString(5),

resultSet.getString(6));

list.add(customer);

}

} catch (Exception e) {

DataOperation.showException(e);

}

// 返回客户信息列表

return list;

}

// 向数据库中添加一个新客户

public static int add(Customer customer) {

try {

// 创建 PreparedStatement 对象，执行插入语句，获取自动生成的主键

PreparedStatement preparedStatement = DataOperation.connection.prepareStatement(

"insert into customer(name,sex,company,tel,cardId) values(?,?,?,?,?)",

Statement.RETURN\_GENERATED\_KEYS);

preparedStatement.setString(1, customer.getName());

preparedStatement.setString(2, customer.getSex());

preparedStatement.setString(3, customer.getCompany());

preparedStatement.setString(4, customer.getTel());

preparedStatement.setString(5, customer.getCardId());

if (preparedStatement.executeUpdate() != 0) {

ResultSet resultSet = preparedStatement.getGeneratedKeys(); // 获取插入的数据的主键

if (resultSet.next())

return resultSet.getInt(1);

}

} catch (Exception e) {

DataOperation.showException(e);

}

return -1;

}

// 根据给定的客户ID，从数据库中删除对应的客户信息

public static boolean delete(int id) {

try {

// 创建 PreparedStatement 对象，执行删除语句

PreparedStatement preparedStatement = DataOperation.connection.prepareStatement(

"delete from customer where id=?");

preparedStatement.setInt(1, id);

return preparedStatement.executeUpdate() != 0;

} catch (Exception e) {

DataOperation.showException(e);

}

return false;

}

// 根据给定的客户信息，更新数据库中对应客户的信息

public static boolean change(Customer customer) {

try {

// 创建 PreparedStatement 对象，执行更新语句

PreparedStatement preparedStatement = DataOperation.connection.prepareStatement(

"update customer set name=?,sex=?,company=?,tel=?,cardId=? where id=?");

preparedStatement.setString(1, customer.getName());

preparedStatement.setString(2, customer.getSex());

preparedStatement.setString(3, customer.getCompany());

preparedStatement.setString(4, customer.getTel());

preparedStatement.setString(5, customer.getCardId());

preparedStatement.setInt(6, (int) customer.getId());

return preparedStatement.executeUpdate() != 0;

} catch (Exception e) {

DataOperation.showException(e);

}

return false;

}

// 根据给定的卡号，返回对应客户的ID

public static int toCustomerId(String cardId) {

try {

// 创建 PreparedStatement 对象，执行查询语句，获取结果集

PreparedStatement preparedStatement = DataOperation.connection

.prepareStatement("select id from customer where cardId=?");

preparedStatement.setString(1, cardId);

ResultSet resultSet = preparedStatement.executeQuery();

while (resultSet.next()) {

return resultSet.getInt(1);

}

} catch (Exception e) {

DataOperation.showException(e);

}

return -1;

}

// 根据给定的客户ID，返回对应客户的姓名

public static String toCustomerName(int id) {

try {

// 创建 PreparedStatement 对象，执行查询语句，获取结果集

PreparedStatement preparedStatement = DataOperation.connection

.prepareStatement("select name from customer where id=?");

preparedStatement.setInt(1, id);

ResultSet resultSet = preparedStatement.executeQuery();

while (resultSet.next()) {

return resultSet.getString(1);

}

} catch (Exception e) {

DataOperation.showException(e);

}

return "???";

}

// 判断给定的卡号是否唯一，如果唯一则返回 true，否则返回 false

public static boolean isUnique(String cardId, int id) {

try {

// 创建 PreparedStatement 对象，执行查询语句，获取结果集

PreparedStatement preparedStatement = DataOperation.connection

.prepareStatement("select id from customer where cardId=?");

preparedStatement.setString(1, cardId);

ResultSet resultSet = preparedStatement.executeQuery();

while (resultSet.next()) {

if (id != resultSet.getInt(1))

return false;

}

} catch (Exception e) {

DataOperation.showException(e);

}

return true;

}

// 重载的 isUnique 方法，不带客户ID，仅判断给定的卡号在数据库中是否唯一

public static boolean isUnique(String cardId) {

return isUnique(cardId, -1);

}

}

package dao;

import java.sql.PreparedStatement;

import java.sql.ResultSet;

import java.sql.Statement;

import java.util.ArrayList;

import java.util.List;

import entity.Dish;

import tool.DataOperation;

public abstract class DishOperation {

// 从数据库中读取所有餐品的信息，并以 Dish 对象的形式返回一个餐品信息列表。

public static List<Dish> load() {

List<Dish> list = new ArrayList<Dish>();

try {

// 创建 PreparedStatement 对象，使用 select 语句从数据库中查询数据

PreparedStatement preparedStatement = DataOperation.connection.prepareStatement("select \* from dish");

// 执行查询操作

ResultSet resultSet = preparedStatement.executeQuery();

// 遍历结果集，将每条记录转化为 Dish 对象，并添加到列表中

while (resultSet.next()) {

Dish dish = new Dish(

resultSet.getInt(1),

resultSet.getString(2),

resultSet.getInt(3),

resultSet.getString(4),

resultSet.getString(5),

resultSet.getDouble(6),

resultSet.getString(7));

list.add(dish);

}

} catch (Exception e) {

// 如果出现异常，则打印异常信息

DataOperation.showException(e);

}

// 返回餐品列表

return list;

}

// 向数据库中添加一个新餐品，参数是一个 Dish 对象，返回值为插入的数据的主键。

public static int add(Dish dish) {

try {

// 创建 PreparedStatement 对象，使用 insert 语句向数据库中插入数据，并返回插入的主键

PreparedStatement preparedStatement = DataOperation.connection.prepareStatement(

"insert into dish(name,dishTypeId,code,unit,price,status) values(?,?,?,?,?,?)",

Statement.RETURN\_GENERATED\_KEYS);

preparedStatement.setString(1, dish.getName());

preparedStatement.setInt(2, dish.getDishTypeId());

preparedStatement.setString(3, dish.getCode());

preparedStatement.setString(4, dish.getUnit());

preparedStatement.setDouble(5, dish.getPrice());

preparedStatement.setString(6, dish.getStatus());

// 执行插入操作，并获取插入的主键

if (preparedStatement.executeUpdate() != 0) {

ResultSet resultSet = preparedStatement.getGeneratedKeys(); // 获取插入的数据的主键

if (resultSet.next())

return resultSet.getInt(1);

}

} catch (Exception e) {

// 如果出现异常，则打印异常信息

DataOperation.showException(e);

}

// 如果插入失败，则返回 -1

return -1;

}

// 根据给定的餐品ID，从数据库中删除对应的餐品信息，返回删除操作的结果。

public static boolean delete(int id) {

try {

// 创建 PreparedStatement 对象，使用 delete 语句从数据库中删除数据

PreparedStatement preparedStatement = DataOperation.connection.prepareStatement(

"delete from dish where id=?");

preparedStatement.setInt(1, id);

// 执行删除操作，返回删除结果

return preparedStatement.executeUpdate() != 0;

} catch (Exception e) {

// 如果出现异常，则打印异常信息

DataOperation.showException(e);

}

// 如果删除失败，则返回 false

return false;

}

// 根据给定的餐品信息，更新数据库中对应餐品的信息，返回更新操作的结果。

public static boolean change(Dish dish) {

try {

// 创建 PreparedStatement 对象，使用 update 语句更新数据库中的数据

PreparedStatement preparedStatement = DataOperation.connection.prepareStatement(

"update dish set name=?,dishTypeId=?,code=?,unit=?,price=?,status=? where id=?");

preparedStatement.setString(1, dish.getName());

preparedStatement.setInt(2, dish.getDishTypeId());

preparedStatement.setString(3, dish.getCode());

preparedStatement.setString(4, dish.getUnit());

preparedStatement.setDouble(5, dish.getPrice());

preparedStatement.setString(6, dish.getStatus());

preparedStatement.setInt(7, dish.getId());

// 执行更新操作，返回更新结果

return preparedStatement.executeUpdate() != 0;

} catch (Exception e) {

// 如果出现异常，则打印异常信息

DataOperation.showException(e);

}

// 如果更新失败，则返回 false

return false;

}

// 根据给定的餐品名称，返回对应餐品的ID。

public static int toDishId(String dishName) {

try {

// 创建 PreparedStatement 对象，使用 select 语句查询数据库中的数据

PreparedStatement preparedStatement = DataOperation.connection

.prepareStatement("select id from dish where name=?");

preparedStatement.setString(1, dishName);

// 执行查询操作，并返回查询结果

ResultSet resultSet = preparedStatement.executeQuery();

while (resultSet.next()) {

return resultSet.getInt(1);

}

} catch (Exception e) {

// 如果出现异常，则打印异常信息

DataOperation.showException(e);

}

// 如果查询失败，则返回 -1

return -1;

}

// 判断给定的餐品名称是否唯一，如果唯一则返回 true，否则返回 false。

public static boolean isUnique(String name, int id) {

try {

// 创建 PreparedStatement 对象，使用 select 语句查询数据库中的数据

PreparedStatement preparedStatement = DataOperation.connection

.prepareStatement("select id from dish where name=?");

preparedStatement.setString(1, name);

// 执行查询操作，并返回查询结果

ResultSet resultSet = preparedStatement.executeQuery();

while (resultSet.next()) {

if (id != resultSet.getInt(1))

return false;

}

} catch (Exception e) {

// 如果出现异常，则打印异常信息

DataOperation.showException(e);

}

// 如果查询成功，则说明给定名称不是唯一的，返回 false；否则返回 true

return true;

}

// 重载的 isUnique 方法，不带餐品ID，仅判断给定的餐品名称在数据库中是否唯一。

public static boolean isUnique(String name) {

// 调用带 id 参数的 isUnique 方法，id 传入 -1，表示不需要指定 id，只判断名称唯一性即可。

return isUnique(name, -1);

}

}

package dao;

import java.sql.PreparedStatement;

import java.sql.ResultSet;

import java.sql.Statement;

import java.util.ArrayList;

import java.util.List;

import entity.DishType;

import tool.DataOperation;

//DishTypeOperation 类，包含了一些对菜品类型进行操作的静态方法。

public abstract class DishTypeOperation {

//从数据库中读取所有菜品类型的信息，并以 DishType 对象的形式返回一个菜品类型信息列表。

public static List<DishType> load() {

List<DishType> list = new ArrayList<DishType>();

try {

PreparedStatement preparedStatement = DataOperation.connection.prepareStatement("select \* from dishType");

ResultSet resultSet = preparedStatement.executeQuery();

while (resultSet.next()) {

DishType dishType = new DishType(

resultSet.getInt(1),

resultSet.getString(2),

resultSet.getString(3));

list.add(dishType);

}

} catch (Exception e) {

DataOperation.showException(e);

}

return list;

}

//向数据库中添加一个新的菜品类型。

public static int add(DishType dishType) {

try {

PreparedStatement preparedStatement = DataOperation.connection.prepareStatement(

"insert into dishType(name,`describe`) values(?,?)",

Statement.RETURN\_GENERATED\_KEYS);

preparedStatement.setString(1, dishType.getName());

preparedStatement.setString(2, dishType.getDescribe());

if (preparedStatement.executeUpdate() != 0) {

ResultSet resultSet = preparedStatement.getGeneratedKeys(); // 获取插入的数据的主键

if (resultSet.next())

return resultSet.getInt(1);

}

} catch (Exception e) {

DataOperation.showException(e);

}

return -1;

}

package dao;

import java.sql.PreparedStatement;

import java.sql.ResultSet;

import java.sql.Statement;

import java.util.ArrayList;

import java.util.List;

import entity.Employee;

import tool.DataOperation;

public abstract class EmployeeOperation {

// 从数据库中读取所有员工信息，并以 Employee 对象的形式返回一个员工信息列表。

public static List<Employee> load() {

List<Employee> list = new ArrayList<Employee>();

try {

PreparedStatement preparedStatement = DataOperation.connection.prepareStatement("select \* from employee");

ResultSet resultSet = preparedStatement.executeQuery();

while (resultSet.next()) {

Employee employee = new Employee(

resultSet.getInt(1), // 员工ID

resultSet.getString(2), // 员工姓名

resultSet.getString(3), // 性别

resultSet.getDate(4), // 生日

resultSet.getString(5), // 身份证号码

resultSet.getString(6), // 住址

resultSet.getString(7), // 电话

resultSet.getString(8), // 职位

resultSet.getString(9)); // 冻结状态

list.add(employee);

}

} catch (Exception e) {

DataOperation.showException(e); // 打印异常信息到控制台

e.printStackTrace();

}

return list;

}

// 向数据库中添加一个新的员工信息。

public static int add(Employee employee) {

try {

PreparedStatement preparedStatement = DataOperation.connection.prepareStatement(

"insert into employee(name,sex,birthday,identityId,address,tel,position,freeze) values(?,?,?,?,?,?,?,?)",

Statement.RETURN\_GENERATED\_KEYS);

preparedStatement.setString(1, employee.getName());

preparedStatement.setString(2, employee.getSex());

preparedStatement.setDate(3, employee.getBirthday());

preparedStatement.setString(4, employee.getIdentityId());

preparedStatement.setString(5, employee.getAddress());

preparedStatement.setString(6, employee.getTel());

preparedStatement.setString(7, employee.getPosition());

preparedStatement.setString(8, employee.getFreeze());

if (preparedStatement.executeUpdate() != 0) {

ResultSet resultSet = preparedStatement.getGeneratedKeys(); // 获取插入的数据的主键

if (resultSet.next())

return resultSet.getInt(1); // 返回新插入的员工信息的ID

}

} catch (Exception e) {

DataOperation.showException(e); // 打印异常信息到控制台

e.printStackTrace();

}

return -1;

}// 根据给定的员工ID，从数据库中删除对应的员工信息。

public static boolean delete(int id) {

try {

PreparedStatement preparedStatement = DataOperation.connection.prepareStatement(

"delete from employee where id=?");

preparedStatement.setInt(1, id);

return preparedStatement.executeUpdate() != 0;

} catch (Exception e) {

DataOperation.showException(e); // 打印异常信息到控制台

e.printStackTrace();

}

return false;

}

// 根据给定的员工信息，更新数据库中对应员工的信息。

public static boolean change(Employee employee) {

try {

PreparedStatement preparedStatement = DataOperation.connection.prepareStatement(

"update employee set name=?,sex=?,birthday=?,identityId=?,address=?,tel=?,position=?,freeze=? where id=?");

preparedStatement.setString(1, employee.getName());

preparedStatement.setString(2, employee.getSex());

preparedStatement.setDate(3, employee.getBirthday());

preparedStatement.setString(4, employee.getIdentityId());

preparedStatement.setString(5, employee.getAddress());

preparedStatement.setString(6, employee.getTel());

preparedStatement.setString(7, employee.getPosition());

preparedStatement.setString(8, employee.getFreeze());

preparedStatement.setInt(9, employee.getId());

return preparedStatement.executeUpdate() != 0;

} catch (Exception e) {

DataOperation.showException(e); // 打印异常信息到控制台

e.printStackTrace();

}

return false;

}

}

package dao;

import java.sql.PreparedStatement;

import java.sql.ResultSet;

import java.sql.Statement;

import java.util.ArrayList;

import java.util.LinkedHashMap;

import java.util.List;

import java.util.Map;

import entity.Order;

import entity.OrderItem;

import tool.DataOperation;

public abstract class OrderOperation {

// 读取所有顾客的身份证号码和姓名，返回一个 Map 对象，其中 key 为身份证号码，value 为顾客姓名。

public static Map<String, String> loadCustomerNameAndCardId() {

Map<String, String> map = new LinkedHashMap<String, String>();

try {

PreparedStatement preparedStatement = DataOperation.connection

.prepareStatement("select cardId,name from customer");

ResultSet resultSet = preparedStatement.executeQuery();

while (resultSet.next()) {

map.put(resultSet.getString(1), resultSet.getString(2));

}

} catch (Exception e) {

DataOperation.showException(e);

}

return map; // 返回结果

}

// 读取所有订单的信息，返回一个 List 对象，其中每个元素是一个 Order 对象，表示一个订单。

public static List<Order> loadOrder() {

List<Order> list = new ArrayList<Order>();

try {

PreparedStatement preparedStatement = DataOperation.connection.prepareStatement("select \* from `order`");

ResultSet resultSet = preparedStatement.executeQuery();

while (resultSet.next()) {

Order order = new Order(

resultSet.getInt(1),

resultSet.getString(2),

resultSet.getInt(3),

resultSet.getTimestamp(4),

resultSet.getDouble(5),

resultSet.getInt(6),

resultSet.getString(7),

resultSet.getInt(8));

list.add(order); // 将每条记录转换为 Order 对象，并添加到 list 中

}

} catch (Exception e) {

DataOperation.showException(e);

}

return list; // 返回结果

}

// 向数据库中添加一条新的订单记录，返回新插入的订单记录的 ID。

public static int addOrder(Order order) {

try {

PreparedStatement preparedStatement = DataOperation.connection.prepareStatement(

"insert into `order`(orderNo,tableId,createtime,money,customerId,status,number) values(?,?,?,?,?,?,?)",

Statement.RETURN\_GENERATED\_KEYS);

preparedStatement.setString(1, order.getOrderNo());

preparedStatement.setInt(2, order.getTableId());

preparedStatement.setTimestamp(3, order.getCreatetime());

preparedStatement.setDouble(4, order.getMoney());

preparedStatement.setInt(5, order.getCustomerId());

preparedStatement.setString(6, order.getStatus());

preparedStatement.setInt(7, order.getNumber());

if (preparedStatement.executeUpdate() != 0) {

ResultSet resultSet = preparedStatement.getGeneratedKeys(); // 获取插入的数据的主键

if (resultSet.next())

return resultSet.getInt(1); // 返回新插入的订单记录的 ID

}

} catch (Exception e) {

DataOperation.showException(e);

}

return -1;

}

// 修改指定 ID 的订单状态为“已支付”，返回操作是否成功。

public static boolean changeOrder(int id) {

try {

PreparedStatement preparedStatement = DataOperation.connection.prepareStatement(

"update `order` set status='paid' where id=?");

preparedStatement.setInt(1, id);

return preparedStatement.executeUpdate() != 0; // 返回操作结果

} catch (Exception e) {

DataOperation.showException(e);

}

return false;

}

// 向数据库中添加一条新的订单项记录，返回新插入的订单记录的 ID。

public static int addOrderItem(OrderItem orderItem) {

try {

PreparedStatement preparedStatement = DataOperation.connection.prepareStatement(

"insert into orderitem(orderId,dishId,amount) values(?,?,?)",

Statement.RETURN\_GENERATED\_KEYS);

preparedStatement.setInt(1, orderItem.getOrderId());

preparedStatement.setInt(2, orderItem.getDishId());

preparedStatement.setInt(3, orderItem.getAmount());

if (preparedStatement.executeUpdate() != 0) {

ResultSet resultSet = preparedStatement.getGeneratedKeys(); // 获取插入的数据的主键

if (resultSet.next())

return resultSet.getInt(1); // 返回新插入的订单记录的 ID

}

} catch (Exception e) {

DataOperation.showException(e);

}

return -1;

}

// 根据给定的订单 ID，从数据库中读取该订单的详细信息，并将其以字符串的形式返回。

// 返回的字符串包含订单的菜品详情，格式为“菜品名称 \* 数量”（每个菜品占一行）。

public static String getDetails(int id) {

String dishName = new String(), amount = new String(), result = new String();

try {

PreparedStatement preparedStatement = DataOperation.connection.prepareStatement(

"select dish.name,orderitem.amount from dish,orderitem where orderitem.dishId = dish.id and orderitem.orderId = ?");

preparedStatement.setInt(1, id);

ResultSet resultSet = preparedStatement.executeQuery();

while (resultSet.next()) {

dishName = resultSet.getString(1);

amount = resultSet.getInt(2) + "\n";

result += dishName + " \* " + amount; // 将菜品详情拼接为一个字符串

}

} catch (Exception e) {

DataOperation.showException(e);

}

return "订单明细：\n\n" + result; // 返回订单的菜品详情

}

}

**package** dao;

**import** java.sql.PreparedStatement;

**import** java.sql.ResultSet;

**import** java.sql.Statement;

**import** java.util.ArrayList;

**import** java.util.List;

**import** entity.Table;

**import** tool.DataOperation;

**public** **abstract** **class** TableOperation {

// 从 `table` 表中读取所有数据并封装成一个 `List<Table>` 返回。

**public** **static** List<Table> load() {

List<Table> list = **new** ArrayList<Table>();

**try** {

// 按照 SQL 语句创建 PreparedStatement 对象

PreparedStatement preparedStatement = DataOperation.*connection*.prepareStatement("select \* from `table`");

// 执行查询语句，并获取结果集

ResultSet resultSet = preparedStatement.executeQuery();

**while** (resultSet.next()) {

// 将每个结果集中的行数据创建为一个 Table 对象并添加到列表中

Table table = **new** Table(

resultSet.getInt(1),

resultSet.getString(2),

resultSet.getInt(3),

resultSet.getString(4));

list.add(table);

}

} **catch** (Exception e) {

DataOperation.*showException*(e);

}

**return** list;

}

// 向表中插入一个新行，并返回插入行的主键。

**public** **static** **int** add(Table table) {

**try** {

// 按照 SQL 语句创建 PreparedStatement 对象，同时要求返回插入行的主键

PreparedStatement preparedStatement = DataOperation.*connection*.prepareStatement(

"insert into `table`(`no`,seating,status) values(?,?,?)",

Statement.***RETURN\_GENERATED\_KEYS***);

// 设置 SQL 语句中的参数值

preparedStatement.setString(1, table.getNo());

preparedStatement.setInt(2, table.getSeating());

preparedStatement.setString(3, table.getStatus());

// 执行插入操作，并返回影响的行数

**if** (preparedStatement.executeUpdate() != 0) {

// 获取插入的数据的主键

ResultSet resultSet = preparedStatement.getGeneratedKeys();

**if** (resultSet.next())

**return** resultSet.getInt(1);

}

} **catch** (Exception e) {

DataOperation.*showException*(e);

}

**return** -1;

}

// 从表中删除指定的行。

**public** **static** **boolean** delete(**int** id) {

**try** {

// 按照 SQL 语句创建 PreparedStatement 对象

PreparedStatement preparedStatement = DataOperation.*connection*.prepareStatement(

"delete from `table` where id=?");

// 设置 SQL 语句中的参数值

preparedStatement.setInt(1, id);

// 执行删除操作，并返回影响的行数

**return** preparedStatement.executeUpdate() != 0;

} **catch** (Exception e) {

DataOperation.*showException*(e);

}

**return** **false**;

}

// 修改表中指定的行。

**public** **static** **boolean** change(Table table) {

**try** {

// 按照 SQL 语句创建 PreparedStatement 对象

PreparedStatement preparedStatement = DataOperation.*connection*.prepareStatement(

"update `table` set `no`=?,seating=?,status=? where id=?");

// 设置 SQL 语句中的参数值

preparedStatement.setString(1, table.getNo());

preparedStatement.setInt(2, table.getSeating());

preparedStatement.setString(3, table.getStatus());

preparedStatement.setInt(4, (**int**) table.getId());

// 执行更新操作，并返回影响的行数

**return** preparedStatement.executeUpdate() != 0;

} **catch** (Exception e) {

DataOperation.*showException*(e);

}

**return** **false**;

}

// 根据一个餐桌编号获取该餐桌在表中的主键。

**public** **static** **int** toTableId(String tableNo) {

**try** {

// 按照 SQL 语句创建 PreparedStatement 对象

PreparedStatement preparedStatement = DataOperation.*connection*

.prepareStatement("select id from `table` where `no`=?");

// 设置 SQL 语句中的参数值

preparedStatement.setString(1, tableNo);

// 执行查询操作，并获取结果集

ResultSet resultSet = preparedStatement.executeQuery();

**while** (resultSet.next()) {

// 返回结果集中 id 列的数值

**return** resultSet.getInt(1);

}

} **catch** (Exception e) {

DataOperation.*showException*(e);

}

**return** -1;

}

// 根据一个餐桌主键获取该餐桌的编号。

**public** **static** String toTableNo(**int** tableId) {

**try** {

// 按照 SQL 语句创建 PreparedStatement 对象

PreparedStatement preparedStatement = DataOperation.*connection*

.prepareStatement("select `no` from `table` where id=?");

// 设置 SQL 语句中的参数值

preparedStatement.setInt(1, tableId);

// 执行查询操作，并获取结果集

ResultSet resultSet = preparedStatement.executeQuery();

**while** (resultSet.next()) {

// 返回结果集中 no 列的字符串值

**return** resultSet.getString(1);

}

} **catch** (Exception e) {

DataOperation.*showException*(e);

}

**return** "???";

}

// 判断一个餐桌编号是否唯一，可传入一个主键排除自身

**public** **static** **boolean** isUnique(String no, **int** id) {

**try** {

// 按照 SQL 语句创建 PreparedStatement 对象

PreparedStatement preparedStatement = DataOperation.*connection*

.prepareStatement("select id from `table` where `no`=?");

// 设置 SQL 语句中的参数值

preparedStatement.setString(1, no);

// 执行查询操作，并获取结果集

ResultSet resultSet = preparedStatement.executeQuery();

**while** (resultSet.next()) {

// 如果存在其他记录的编号和该编号相同，则返回 false

**if** (id != resultSet.getInt(1))

**return** **false**;

}

} **catch** (Exception e) {

DataOperation.*showException*(e);

}

// 否则返回 true

**return** **true**;

}

// 判断一个餐桌编号是否唯一，不传入主键则认为自身是新记录。

**public** **static** **boolean** isUnique(String no) {

**return** *isUnique*(no, -1);

}

// 获取该餐桌最近一次客户的姓名和贵宾卡号，并返回字符串形式的结果。

**public** **static** String getDetails(**int** id) {

String customerName = **null**, customerCardId = **null**;

**try** {

// 按照 SQL 语句创建 PreparedStatement 对象

PreparedStatement preparedStatement = DataOperation.*connection*.prepareStatement(

"select name,cardId from customer where id in (select customerId from `order` where createtime in (select max(createtime) from `order` where tableId = ?) and tableId = ?)");

**for** (**int** i = 1; i <= 2; i++)

preparedStatement.setInt(i, id);

// 执行查询操作，并获取结果集

ResultSet resultSet = preparedStatement.executeQuery();

**while** (resultSet.next()) {

// 获取结果集中 name 列和 cardId 列的值

customerName = resultSet.getString(1);

customerCardId = resultSet.getString(2);

}

} **catch** (Exception e) {

DataOperation.*showException*(e);

}

// 如果没有获取到客户信息，返回默认字符串。

**if** (customerName == **null**)

**return** "无";

// 否则返回格式化后的字符串。

**return** "该餐台最近一次的就餐客户信息：\n\n客户姓名：\n" + customerName + "\n贵宾卡号：\n" + customerCardId;

}

}

**entity包**

**package** entity;

**public** **class** Customer {

**private** **int** id;

**private** String name, sex, company, tel, cardId;

**public** Customer(**int** id, String name, String sex, String company, String tel, String cardId) {

**this**.id = id;

**this**.name = name;

**this**.sex = sex;

**this**.company = company;

**this**.tel = tel;

**this**.cardId = cardId;

**if** (company != **null** && company.equals(""))

**this**.company = **null**;

**if** (tel != **null** && tel.equals(""))

**this**.tel = **null**;

}

**public** **int** getId() {

**return** id;

}

**public** String getName() {

**return** name;

}

**public** String getSex() {

**return** sex;

}

**public** String getCompany() {

**return** company;

}

**public** String getTel() {

**return** tel;

}

**public** String getCardId() {

**return** cardId;

}

}

**package** entity;

**public** **class** Dish {

**private** **int** id, dishTypeId;

**private** String name, code, unit, status;

**private** **double** price;

**public** Dish(**int** id, String name, **int** dishTypeId, String code, String unit, **double** price, String status) {

**this**.id = id;

**this**.dishTypeId = dishTypeId;

**this**.name = name;

**this**.code = code;

**this**.unit = unit;

**this**.status = status;

**this**.price = price;

**if** (unit != **null** && unit.equals(""))

**this**.unit = **null**;

}

**public** **int** getId() {

**return** id;

}

**public** **int** getDishTypeId() {

**return** dishTypeId;

}

**public** String getName() {

**return** name;

}

**public** String getCode() {

**return** code;

}

**public** String getUnit() {

**return** unit;

}

**public** String getStatus() {

**return** status;

}

**public** **double** getPrice() {

**return** price;

}

}

**package** entity;

**public** **class** DishType {

**int** id;

String name, describe;

**public** DishType(**int** id, String name, String describe) {

**this**.id = id;

**this**.name = name;

**this**.describe = describe;

**if** (describe != **null** && describe.equals(""))

**this**.describe = **null**;

}

**public** **int** getId() {

**return** id;

}

**public** String getName() {

**return** name;

}

**public** String getDescribe() {

**return** describe;

}

}

**package** entity;

**import** java.sql.Date;

**public** **class** Employee {

**private** **int** id;

**private** String name, sex, identityId, address, tel, position, freeze;

**private** Date birthday;

**public** Employee(**int** id, String name, String sex, Date birthday, String identityId, String address, String tel,

String position, String freeze) {

**this**.id = id;

**this**.name = name;

**this**.sex = sex;

**this**.identityId = identityId;

**this**.address = address;

**this**.tel = tel;

**this**.position = position;

**this**.freeze = freeze;

**this**.birthday = birthday;

**if** (identityId != **null** && identityId.equals(""))

**this**.identityId = **null**;

**if** (address != **null** && address.equals(""))

**this**.address = **null**;

**if** (tel != **null** && tel.equals(""))

**this**.tel = **null**;

}

**public** **int** getId() {

**return** id;

}

**public** String getName() {

**return** name;

}

**public** String getSex() {

**return** sex;

}

**public** String getIdentityId() {

**return** identityId;

}

**public** String getAddress() {

**return** address;

}

**public** String getTel() {

**return** tel;

}

**public** String getPosition() {

**return** position;

}

**public** String getFreeze() {

**return** freeze;

}

**public** Date getBirthday() {

**return** birthday;

}

}

**package** entity;

**import** java.sql.Timestamp;

**public** **class** Order {

**private** **int** id, tableId, customerId, number;

**private** String orderNo, status;

**private** Timestamp createtime;

**private** **double** money;

**public** Order(**int** id, String orderNo, **int** tableId, Timestamp createtime, **double** money, **int** customerId, String status,

**int** number) {

**this**.id = id;

**this**.tableId = tableId;

**this**.customerId = customerId;

**this**.number = number;

**this**.orderNo = orderNo;

**this**.status = status;

**this**.createtime = createtime;

**this**.money = money;

}

**public** **int** getId() {

**return** id;

}

**public** **int** getTableId() {

**return** tableId;

}

**public** **int** getCustomerId() {

**return** customerId;

}

**public** **int** getNumber() {

**return** number;

}

**public** String getOrderNo() {

**return** orderNo;

}

**public** String getStatus() {

**return** status;

}

**public** Timestamp getCreatetime() {

**return** createtime;

}

**public** **double** getMoney() {

**return** money;

}

}

**package** entity;

**public** **class** OrderItem {

**private** **int** id, orderId, dishId, amount;

**public** OrderItem(**int** id, **int** orderId, **int** dishId, **int** amount) {

**this**.id = id;

**this**.orderId = orderId;

**this**.dishId = dishId;

**this**.amount = amount;

}

**public** **int** getId() {

**return** id;

}

**public** **int** getOrderId() {

**return** orderId;

}

**public** **int** getDishId() {

**return** dishId;

}

**public** **int** getAmount() {

**return** amount;

}

}

**package** entity;

**public** **class** Table {

**private** **int** id, seating;

**private** String no, status;

**public** Table(**int** id, String no, **int** seating, String status) {

**this**.id = id;

**this**.seating = seating;

**this**.no = no;

**this**.status = status;

}

**public** **int** getId() {

**return** id;

}

**public** **int** getSeating() {

**return** seating;

}

**public** String getNo() {

**return** no;

}

**public** String getStatus() {

**return** status;

}

}

**gui包**

package gui;

import java.awt.BorderLayout;

import java.awt.Dimension;

import java.awt.Font;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import java.sql.SQLException;

import javax.swing.JButton;

import javax.swing.JDialog;

import javax.swing.JLabel;

import javax.swing.JOptionPane;

import javax.swing.JPanel;

import javax.swing.JPasswordField;

import javax.swing.JTextField;

import javax.swing.SwingConstants;

import tool.Appearance;

import tool.DataOperation;

public class LoginWindow extends JDialog {

// 定义窗口中的组件，包括标题、账号和密码输入框、登录按钮等。

private JLabel titleLabel = new JLabel("餐饮管理系统"), usernameLabel = new JLabel("账号"),

passwordLabel = new JLabel("密码");

private JTextField usernameText = new JTextField();

private JPasswordField passwordText = new JPasswordField();

private JButton loginButton = new JButton("登录");

private JPanel usernamePanel = new JPanel(), passwordPanel = new JPanel(), infoPanel = new JPanel();

// 构造方法，用于初始化窗口的各种属性和组件。

public LoginWindow() {

setTitle("登录");

setResizable(false);

setSize(256, 192);

setDefaultCloseOperation(DISPOSE\_ON\_CLOSE);

// 将窗口设置在屏幕中央

Appearance.setToCenter(this);

// 设置输入框和标签的默认尺寸、位置以及字体等样式属性

usernameText.setPreferredSize(new Dimension(160, 24));

passwordText.setPreferredSize(new Dimension(160, 24));

titleLabel.setPreferredSize(new Dimension(256, 32));

titleLabel.setHorizontalAlignment(SwingConstants.CENTER);

// 对账号和密码输入框进行布局

usernamePanel.add(usernameLabel);

usernamePanel.add(usernameText);

passwordPanel.add(passwordLabel);

passwordPanel.add(passwordText);

infoPanel.add(usernamePanel);

infoPanel.add(passwordPanel);

// 将组件添加到窗口中，并设置每个组件的布局位置

add(titleLabel, BorderLayout.NORTH);

add(infoPanel, BorderLayout.CENTER);

add(loginButton, BorderLayout.SOUTH);

// 添加登录按钮的点击事件监听器

loginButton.addActionListener(loginAction);

usernameText.addActionListener(loginAction);

passwordText.addActionListener(loginAction);

// 设置窗口的样式和字体

Appearance.setToSystemTheme();

Appearance.setAllFont(this, Appearance.DEFAULT\_FONT);

titleLabel.setFont(new Font("Microsoft YaHei UI", Font.BOLD, 32));

setVisible(true);

}

// 登录按钮点击事件监听器，用于尝试连接到数据库并显示主窗口

ActionListener loginAction = new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

if (loginButton.isEnabled()) {

// 获取账号和密码输入框中的信息

String usr = usernameText.getText(), psw = new String(passwordText.getPassword());

if (usr.equals("")) { // 当账号为空时提示错误信息

JOptionPane.showMessageDialog(null, "请输入账号！", "错误", JOptionPane.ERROR\_MESSAGE);

} else if (psw.equals("")) { // 当密码为空时提示错误信息

JOptionPane.showMessageDialog(null, "请输入密码！", "错误", JOptionPane.ERROR\_MESSAGE);

} else {

setTitle("正在登录……"); // 将窗口标题改为“正在登录……”

loginButton.setEnabled(false); // 禁用登录按钮

try {

DataOperation.getConnection(usr, psw); // 尝试连接到数据库

new MainFrame(usr); // 显示主窗口并传入用户名参数

dispose(); // 关闭登录窗口

} catch (SQLException ex) { // 连接失败时处理 SQL 异常

JOptionPane.showMessageDialog(null, "无法连接到数据库！\n请检查账号和密码是否正确，数据库是否可用。\n\n" + ex.getMessage(),

"错误", JOptionPane.ERROR\_MESSAGE);

setTitle("登录"); // 将窗口标题改回“登录”

loginButton.setEnabled(true); // 重新启用登录按钮

}

}

}

}

};

}

package gui;

import java.awt.BorderLayout;

import java.awt.Dimension;

import java.awt.FlowLayout;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import javax.swing.JButton;

import javax.swing.JFrame;

import javax.swing.JOptionPane;

import javax.swing.JPanel;

import javax.swing.JTabbedPane;

import javax.swing.JTable;

import javax.swing.JTextField;

import javax.swing.event.ChangeEvent;

import javax.swing.event.ChangeListener;

import javax.swing.event.DocumentEvent;

import javax.swing.event.DocumentListener;

import javax.swing.table.DefaultTableModel;

import dao.CustomerOperation;

import dao.DishOperation;

import dao.DishTypeOperation;

import dao.EmployeeOperation;

import dao.TableOperation;

import gui.dialog.CustomerDialog;

import gui.dialog.DishDialog;

import gui.dialog.DishTypeDialog;

import gui.dialog.EmployeeDialog;

import gui.dialog.TableDialog;

import gui.panel.CustomerPanel;

import gui.panel.DishPanel;

import gui.panel.DishTypePanel;

import gui.panel.EmployeePanel;

import gui.panel.OrderPanel;

import gui.panel.TablePanel;

import tool.Appearance;

public class MainFrame extends JFrame {

public int count = 0; // 打开的点菜和结账对话框的数量

private JTabbedPane tabbedPane = new JTabbedPane(), dishTab = new JTabbedPane();

public JButton addButton = new JButton("新增"), delButton = new JButton("删除"), changeButton = new JButton("修改"),

searchButton = new JButton("查找");

private JTextField searchText = new JTextField();

public JPanel topPanel = new JPanel(new BorderLayout()),

operationPanel = new JPanel(new FlowLayout(FlowLayout.LEFT)),

searchPanel = new JPanel(new FlowLayout(FlowLayout.RIGHT));

public EmployeePanel employeePanel = new EmployeePanel();

public CustomerPanel customerPanel = new CustomerPanel();

public TablePanel tablePanel = new TablePanel();

public DishPanel dishPanel = new DishPanel();

public DishTypePanel dishTypePanel = new DishTypePanel();

public OrderPanel orderPanel = new OrderPanel();

public MainFrame(String usr) {

setTitle("餐饮管理系统 - " + usr); // 标题

setSize(1024, 720); // 大小

setDefaultCloseOperation(EXIT\_ON\_CLOSE); // 关闭操作

Appearance.setToSystemTheme(); // 设置外观主题为系统主题

Appearance.setToCenter(this); // 设置窗口居中显示

searchText.setPreferredSize(new Dimension(256, 28)); // 设置搜索框的大小

searchText.setHorizontalAlignment(JTextField.RIGHT); // 设置搜索框中文字靠右对齐

Appearance.add(operationPanel, addButton, delButton, changeButton); // 将按钮添加到操作面板中

Appearance.add(searchPanel, searchText, searchButton); // 将搜索框和搜索按钮添加到搜索面板中

topPanel.add(operationPanel, BorderLayout.WEST); // 将操作面板添加到顶部面板左侧

topPanel.add(searchPanel, BorderLayout.EAST); // 将搜索面板添加到顶部面板右侧

tabbedPane.addTab("员工管理", employeePanel); // 向选项卡中添加员工管理面板

tabbedPane.addTab("客户管理", customerPanel); // 向选项卡中添加客户管理面板

tabbedPane.addTab("餐台管理（点菜）", tablePanel); // 向选项卡中添加餐桌管理面板

dishTab.addTab("菜品信息", dishPanel); // 向菜品管理选项卡中添加菜品信息管理面板

dishTab.addTab("菜品分类信息", dishTypePanel); // 向菜品管理选项卡中添加菜品分类管理面板

tabbedPane.addTab("菜品管理", dishTab); // 向选项卡中添加菜品管理选项卡

tabbedPane.addTab("订单管理（结账）", orderPanel); // 向选项卡中添加订单管理面板

addButton.addActionListener(updateAction); // 新增按钮添加点击事件处理

changeButton.addActionListener(updateAction); // 修改按钮添加点击事件处理

delButton.addActionListener(delAction); // 删除按钮添加点击事件处理

tabbedPane.addChangeListener(changeAction); // 选项卡添加切换事件处理

searchText.getDocument().addDocumentListener(typeAction); // 搜索框添加文本变更事件处理

searchText.addActionListener(findAction); // 搜索框添加回车事件处理

searchButton.addActionListener(findAction); // 搜索按钮添加点击事件处理

add(tabbedPane, BorderLayout.CENTER); // 将选项卡添加到主界面中央

add(topPanel, BorderLayout.NORTH); // 将顶部面板添加到主界面上方

Appearance.setAllFont(this, Appearance.DEFAULT\_FONT); // 设置组件字体

tablePanel.orderButton.setFont(Appearance.DEFAULT\_BIG\_FONT); // 设置点餐按钮字体

orderPanel.payButton.setFont(Appearance.DEFAULT\_BIG\_FONT); // 设置结账按钮字体

changeAction.stateChanged(null); // 初始状态下默认显示第一个选项卡面板

typeAction.changedUpdate(null); // 初始状态下搜索框为空，不触发搜索事件

setVisible(true); // 显示主界面

}

// 新增和修改按钮的点击事件

public ActionListener updateAction = new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

switch (tabbedPane.getSelectedIndex()) {

case 0: // 增改员工

new EmployeeDialog(e.getSource() == addButton, employeePanel.table); // 弹出员工信息对话框，可以选择是新增还是修改

break;

case 1: // 增改客户

new CustomerDialog(e.getSource() == addButton, customerPanel.table); // 弹出客户信息对话框，可以选择是新增还是修改

orderPanel.refreshOrderTable(); // 刷新订单管理界面中的订单信息

break;

case 2: // 增改餐台

new TableDialog(e.getSource() == addButton, tablePanel.table); // 弹出餐桌信息对话框，可以选择是新增还是修改

tablePanel.selectAction.valueChanged(null); // 触发餐桌选择事件处理，更新点餐操作面板

orderPanel.refreshOrderTable(); // 刷新订单管理界面中的订单信息

break;

case 3:

if (dishTab.getSelectedIndex() == 0) // 增改菜品

new DishDialog(e.getSource() == addButton, dishPanel.table); // 弹出菜品信息对话框，可以选择是新增还是修改

else // 增改菜类

new DishTypeDialog(e.getSource() == addButton, dishTypePanel.table); // 弹出菜品分类信息对话框，可以选择是新增还是修改

dishPanel.refreshDishTable(); // 刷新菜品信息管理面板中的菜品信息

break;

}

}

};

// 删除按钮的点击事件

public ActionListener delAction = new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

if (JOptionPane.showConfirmDialog(null, "确定要删除所选内容吗？", "删除",

JOptionPane.YES\_NO\_OPTION) == JOptionPane.YES\_OPTION) { // 弹出确认删除对话框，只有确认删除才会执行删除操作

boolean success = true;

switch (tabbedPane.getSelectedIndex()) {

case 0: // 删除员工

int employeeSelected = employeePanel.table

.convertRowIndexToModel(employeePanel.table.getSelectedRow()); // 获取选中的员工行号

DefaultTableModel employeeModel = (DefaultTableModel) employeePanel.table.getModel(); // 获取员工表格模型

success = EmployeeOperation.delete((int) employeeModel.getValueAt(employeeSelected, 0)); // 根据员工 ID 进行删除操作

if (success)

employeeModel.removeRow(employeeSelected); // 如果删除成功，则从表格模型中移除该行

break;

case 1: // 删除客户

int customerSelected = customerPanel.table

.convertRowIndexToModel(customerPanel.table.getSelectedRow()); // 获取选中的客户行号

DefaultTableModel customerModel = (DefaultTableModel) customerPanel.table.getModel(); // 获取客户表格模型

success = CustomerOperation.delete((int) customerModel.getValueAt(customerSelected, 0)); // 根据客户 ID 进行删除操作

if (success)

customerModel.removeRow(customerSelected); // 如果删除成功，则从表格模型中移除该行

orderPanel.refreshOrderTable(); // 刷新订单管理界面中的订单信息

break;

case 2: // 删除餐台

int tableSelected = tablePanel.table

.convertRowIndexToModel(tablePanel.table.getSelectedRow()); // 获取选中的餐桌行号

DefaultTableModel tableModel = (DefaultTableModel) tablePanel.table.getModel(); // 获取餐桌表格模型

success = TableOperation.delete((int) tableModel.getValueAt(tableSelected, 0)); // 根据餐桌 ID 进行删除操作

if (success)

tableModel.removeRow(tableSelected); // 如果删除成功，则从表格模型中移除该行

orderPanel.refreshOrderTable(); // 刷新订单管理界面中的订单信息

break;

case 3:

if (dishTab.getSelectedIndex() == 0) { // 删除菜品

int dishSelected = dishPanel.table

.convertRowIndexToModel(dishPanel.table.getSelectedRow()); // 获取选中的菜品行号

DefaultTableModel dishModel = (DefaultTableModel) dishPanel.table.getModel(); // 获取菜品表格模型

success = DishOperation.delete((int) dishModel.getValueAt(dishSelected, 0)); // 根据菜品 ID 进行删除操作

if (success)

dishModel.removeRow(dishSelected); // 如果删除成功，则从表格模型中移除该行

} else { // 删除菜类

int dishTypeSelected = dishTypePanel.table

.convertRowIndexToModel(dishTypePanel.table.getSelectedRow()); // 获取选中的菜品分类行号

DefaultTableModel dishTypeModel = (DefaultTableModel) dishTypePanel.table.getModel(); // 获取菜品分类表格模型

success = DishTypeOperation.delete((int) dishTypeModel.getValueAt(dishTypeSelected, 0)); // 根据菜品分类 ID 进行删除操作

if (success) {

dishTypeModel.removeRow(dishTypeSelected); // 如果删除成功，则从表格模型中移除该行

dishPanel.refreshDishTable(); // 刷新菜品信息管理面板中的菜品信息

}

}

break;

}

if (!success)

JOptionPane.showMessageDialog(null, "删除失败！", "错误", JOptionPane.ERROR\_MESSAGE); // 如果删除失败，则弹出提示对话框

}

}

};

public ChangeListener changeAction = new ChangeListener() { // 标签页切换事件

@Override

public void stateChanged(ChangeEvent e) {

boolean isSelected;

if (tabbedPane.getSelectedIndex() == 4) {

orderPanel.selectAction.valueChanged(null);

addButton.setEnabled(false);

delButton.setEnabled(false);

changeButton.setEnabled(false);

} else {

addButton.setEnabled(true);

delButton.setEnabled(true);

changeButton.setEnabled(true);

}

switch (tabbedPane.getSelectedIndex()) {

case 0:

isSelected = employeePanel.table.getSelectedRow() != -1;

customerPanel.setVisible(false);

tablePanel.setVisible(false);

dishPanel.setVisible(false);

dishTypePanel.setVisible(false);

orderPanel.setVisible(false);

break;

case 1:

isSelected = customerPanel.table.getSelectedRow() != -1;

break;

case 2:

isSelected = tablePanel.table.getSelectedRow() != -1;

break;

case 3:

if (dishTab.getSelectedIndex() == 0)

isSelected = dishPanel.table.getSelectedRow() != -1;

else

isSelected = dishTypePanel.table.getSelectedRow() != -1;

break;

default:

isSelected = false;

}

delButton.setEnabled(isSelected);

changeButton.setEnabled(isSelected);

}

};

private DocumentListener typeAction = new DocumentListener() { // 搜索框键入内容事件

@Override

public void changedUpdate(DocumentEvent e) {

searchButton.setEnabled(!searchText.getText().equals(""));

}

@Override

public void insertUpdate(DocumentEvent e) {

changedUpdate(e);

}

@Override

public void removeUpdate(DocumentEvent e) {

changedUpdate(e);

}

};

private ActionListener findAction = new ActionListener() { // 搜索框搜索事件

@Override

public void actionPerformed(ActionEvent e) {

if (searchText.getText().equals(""))

return;

JTable table = null;

switch (tabbedPane.getSelectedIndex()) {

case 0:

table = employeePanel.table;

break;

case 1:

table = customerPanel.table;

break;

case 2:

table = tablePanel.table;

break;

case 3:

if (dishTab.getSelectedIndex() == 0)

table = dishPanel.table;

else

table = dishTypePanel.table;

break;

case 4:

table = orderPanel.table;

break;

}

boolean found = false;

int x = table.getSelectedRow() == table.getRowCount() ? 0 : table.getSelectedRow() + 1;

for (int r = x; r < table.getRowCount() && !found; r++) {

for (int c = 0; c < table.getColumnCount() && !found; c++) {

Object obj = table.getValueAt(r, c);

if (obj != null && obj.toString().contains(searchText.getText())) {

table.setRowSelectionInterval(r, r);

found = true;

}

}

}

if (!found) {

for (int r = 0; r < table.getSelectedRow() && !found; r++) {

for (int c = 0; c < table.getColumnCount() && !found; c++) {

Object obj = table.getValueAt(r, c);

if (obj != null && obj.toString().contains(searchText.getText())) {

table.setRowSelectionInterval(r, r);

found = true;

}

}

}

}

if (!found && table.getSelectedRow() != -1) {

for (int c = 0; c < table.getColumnCount() && !found; c++) {

Object obj = table.getValueAt(table.getSelectedRow(), c);

if (obj != null && obj.toString().contains(searchText.getText()))

found = true;

}

}

if (!found)

JOptionPane.showMessageDialog(null, "该表未能找到含有“" + searchText.getText() + "”的单元格。", "查找",

JOptionPane.PLAIN\_MESSAGE);

}

};

}

**gui.dialog包**

package gui.dialog;

import java.awt.Dimension;

import java.awt.FlowLayout;

import java.awt.GridLayout;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import javax.swing.ButtonGroup;

import javax.swing.JButton;

import javax.swing.JDialog;

import javax.swing.JLabel;

import javax.swing.JOptionPane;

import javax.swing.JPanel;

import javax.swing.JRadioButton;

import javax.swing.JTable;

import javax.swing.JTextField;

import javax.swing.table.DefaultTableModel;

import dao.CustomerOperation;

import entity.Customer;

import tool.Appearance;

import tool.DataOperation;

public class CustomerDialog extends JDialog {

private boolean isNew;

private JTable table;

private JLabel nameLabel = new JLabel("姓名\*"),

sexLabel = new JLabel("性别\*"),

companyLabel = new JLabel("单位"),

telLabel = new JLabel("电话"),

cardIdLabel = new JLabel("贵宾卡号\*");

private JTextField nameText = new JTextField(),

companyText = new JTextField(),

telText = new JTextField(),

cardIdText = new JTextField();

private FlowLayout layout = new FlowLayout(FlowLayout.RIGHT);

private JPanel namePanel = new JPanel(layout),

sexPanel = new JPanel(layout),

companyPanel = new JPanel(layout),

telPanel = new JPanel(layout),

cardIdPanel = new JPanel(layout),

buttonPanel = new JPanel(new FlowLayout(FlowLayout.CENTER));

private JRadioButton maleRadio = new JRadioButton("male"), femaleRadio = new JRadioButton("female");

private ButtonGroup sexGroup = new ButtonGroup();

private JButton okButton = new JButton("确定"), cancelButton = new JButton("取消");

// 构造函数

public CustomerDialog(boolean isNew, JTable table) {

setTitle((isNew ? "新增" : "修改") + "客户"); // 设置窗口标题

setModal(true); // 已模态方式显示

this.isNew = isNew; // 是否为新建客户信息

this.table = table; // 相关的JTable表格

Appearance.setToCenter(this); // 窗口居中

setResizable(false); // 不可调整大小

setLayout(new GridLayout(6, 1)); // 窗口布局采用GridLayout

// 设置控件尺寸

nameText.setPreferredSize(new Dimension(160, 28));

maleRadio.setPreferredSize(new Dimension(80, 28));

femaleRadio.setPreferredSize(new Dimension(80, 28));

companyText.setPreferredSize(new Dimension(160, 28));

telText.setPreferredSize(new Dimension(160, 28));

cardIdText.setPreferredSize(new Dimension(160, 28));

Appearance.add(namePanel, nameLabel, nameText); // 将姓名标签和输入框添加到姓名面板

Appearance.add(sexPanel, sexLabel, maleRadio, femaleRadio); // 将性别标签和单选框添加到性别面板

sexGroup.add(maleRadio); // 将男性单选框添加到性别按钮组

sexGroup.add(femaleRadio); // 将女性单选框添加到性别按钮组

Appearance.add(companyPanel, companyLabel, companyText); // 将单位标签和输入框添加到单位面板

Appearance.add(telPanel, telLabel, telText); // 将电话标签和输入框添加到电话面板

Appearance.add(cardIdPanel, cardIdLabel, cardIdText); // 将贵宾卡号标签和输入框添加到贵宾卡号面板

Appearance.add(buttonPanel, okButton, cancelButton); // 将确定和取消按钮添加到按钮面板

Appearance.add(this, namePanel, sexPanel, companyPanel, telPanel, cardIdPanel, buttonPanel); // 将各个面板添加到窗口中

if (!isNew) { // 如果是修改已有客户信息，则从选中的表格行中获取相应数据并填入对应的文本框和单选框中

int row = table.getSelectedRow();

nameText.setText(table.getValueAt(row, 0).toString());

if (table.getValueAt(row, 1).equals("male")) {

maleRadio.setSelected(true);

} else {

femaleRadio.setSelected(true);

}

Object company = table.getValueAt(row, 2), tel = table.getValueAt(row, 3);

companyText.setText(company == null ? "" : company.toString());

telText.setText(tel == null ? "" : tel.toString());

cardIdText.setText(table.getValueAt(row, 4).toString());

}

Appearance.addListenerToDialog(this, applyAction, okButton, cancelButton); // 添加事件监听器

Appearance.setAllFont(this, Appearance.DEFAULT\_FONT); // 设置字体

pack(); // 窗口自适应大小

setVisible(true); // 显示窗口

}

private ActionListener applyAction = new ActionListener() { // 定义确定按钮的响应事件

@Override

public void actionPerformed(ActionEvent e) {

DataOperation.trimTextField(nameText, companyText, telText, cardIdText); // 清除文本框中的前后空格

if (nameText.getText().equals("") || (!maleRadio.isSelected() && !femaleRadio.isSelected())

|| cardIdText.getText().equals("")) { // 对输入的数据进行格式验证

// 如果有数据为空或者未选性别，则提示用户完善信息

JOptionPane.showMessageDialog(null, "请完善信息并确认信息格式无误！", "错误", JOptionPane.ERROR\_MESSAGE);

} else {

Customer customer = new Customer(

isNew ? -1 // 如果是新增客户信息，则设置ID为-1

: (int) table.getModel()

.getValueAt(table.convertRowIndexToModel(table.getSelectedRow()), 0), // 如果是修改已有客户信息，则从表格中获取相应的ID

nameText.getText(),

maleRadio.isSelected() ? "male" : "female",

companyText.getText(),

telText.getText(),

cardIdText.getText());

if (isNew) { // 新增客户信息

if (!CustomerOperation.isUnique(cardIdText.getText())) { // 检查贵宾卡号是否唯一

JOptionPane.showMessageDialog(null, "该贵宾卡号已存在！", "错误", JOptionPane.ERROR\_MESSAGE);

} else {

int id = CustomerOperation.add(customer); // 调用CustomerOperation中的add方法添加客户信息，返回新客户信息的ID

if (id != -1) {

((DefaultTableModel) table.getModel()).addRow(new Object[] {

id,

customer.getName(),

customer.getSex(),

customer.getCompany(),

customer.getTel(),

customer.getCardId()

}); // 将新添加的客户信息填入表格

dispose(); // 关闭窗口

} else {

JOptionPane.showMessageDialog(null, "添加失败！", "错误", JOptionPane.ERROR\_MESSAGE);

}

}

} else { // 修改客户信息

if (!CustomerOperation.isUnique(cardIdText.getText(),

(int) table.getModel().getValueAt(table.convertRowIndexToModel(table.getSelectedRow()),

0))) { // 检查贵宾卡号是否唯一

JOptionPane.showMessageDialog(null, "该贵宾卡号已有其他客户使用！", "错误", JOptionPane.ERROR\_MESSAGE);

} else {

if (CustomerOperation.change(customer)) { // 调用CustomerOperation中的change方法修改客户信息

int selectedRow = table.getSelectedRow();

table.setValueAt(customer.getName(), selectedRow, 0); // 更新表格中对应行的信息

table.setValueAt(customer.getSex(), selectedRow, 1);

table.setValueAt(customer.getCompany(), selectedRow, 2);

table.setValueAt(customer.getTel(), selectedRow, 3);

table.setValueAt(customer.getCardId(), selectedRow, 4);

dispose(); // 关闭当前窗口

} else {

JOptionPane.showMessageDialog(null, "修改失败！", "错误", JOptionPane.ERROR\_MESSAGE);

}

}

}

}

}

};

}

package gui.dialog;

import java.awt.Dimension;

import java.awt.FlowLayout;

import java.awt.GridLayout;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import java.util.List;

import javax.swing.JButton;

import javax.swing.JCheckBox;

import javax.swing.JComboBox;

import javax.swing.JDialog;

import javax.swing.JLabel;

import javax.swing.JOptionPane;

import javax.swing.JPanel;

import javax.swing.JTable;

import javax.swing.JTextField;

import javax.swing.table.DefaultTableModel;

import dao.DishOperation;

import dao.DishTypeOperation;

import entity.Dish;

import tool.Appearance;

import tool.DataOperation;

public class DishDialog extends JDialog {

private boolean isNew;

private JTable table;

private JLabel nameLabel = new JLabel("菜品名\*"), dishTypeLabel = new JLabel("类别\*"), codeLabel = new JLabel("菜品代码\*"),

unitLabel = new JLabel("单位"),

priceLabel = new JLabel("价格\*");

private JTextField nameText = new JTextField(), codeText = new JTextField(), unitText = new JTextField(),

priceText = new JTextField();

private FlowLayout layout = new FlowLayout(FlowLayout.RIGHT);

private JPanel namePanel = new JPanel(layout), dishTypePanel = new JPanel(layout), codePanel = new JPanel(layout),

unitPanel = new JPanel(layout), pricePanel = new JPanel(layout),

statusPanel = new JPanel(new FlowLayout(FlowLayout.CENTER)),

buttonPanel = new JPanel(new FlowLayout(FlowLayout.CENTER));

private JComboBox<String> dishTypeCombo = new JComboBox<String>();

private JCheckBox statusCheck = new JCheckBox("sold out");

private JButton okButton = new JButton("确定"), cancelButton = new JButton("取消");

public DishDialog(boolean isNew, JTable table) {

// 标题为新增或修改菜品

setTitle((isNew ? "新增" : "修改") + "菜品");

// 设置为模态对话框

setModal(true);

// 是否为新增菜品

this.isNew = isNew;

// 菜品信息表格

this.table = table;

// 窗口居中显示

Appearance.setToCenter(this);

// 禁止调整窗口大小

setResizable(false);

// 网格布局，7行1列

setLayout(new GridLayout(7, 1));

// 加载所有菜品类型名称

List<String> list = DishTypeOperation.loadDishTypeName();

for (String dishTypeName : list) {

// 将菜品类型名称添加到菜品类型组合框中

dishTypeCombo.addItem(dishTypeName);

}

// 设置各个文本框的尺寸

nameText.setPreferredSize(new Dimension(160, 28));

dishTypeCombo.setPreferredSize(new Dimension(160, 28));

codeText.setPreferredSize(new Dimension(160, 28));

unitText.setPreferredSize(new Dimension(160, 28));

priceText.setPreferredSize(new Dimension(160, 28));

// 将菜品名、菜品类型、菜品代码、计量单位、价格标签和对应的文本框添加到相应的面板上

Appearance.add(namePanel, nameLabel, nameText);

Appearance.add(dishTypePanel, dishTypeLabel, dishTypeCombo);

Appearance.add(codePanel, codeLabel, codeText);

Appearance.add(unitPanel, unitLabel, unitText);

Appearance.add(pricePanel, priceLabel, priceText);

// 将售罄状态勾选框添加到售罄状态面板上

statusPanel.add(statusCheck);

// 将确定和取消按钮添加到按钮面板上

Appearance.add(buttonPanel, okButton, cancelButton);

// 将菜品名、菜品类型、菜品代码、计量单位、价格、售罄状态、按钮面板依次添加到对话框中

Appearance.add(this, namePanel, dishTypePanel, codePanel, unitPanel, pricePanel, statusPanel, buttonPanel);

// 如果不是新增菜品，则将选中菜品的信息自动填充到对应的文本框和组合框中

if (!isNew) {

int row = table.getSelectedRow();

nameText.setText(table.getValueAt(row, 0).toString());

dishTypeCombo.setSelectedItem(table.getValueAt(row, 1).toString());

codeText.setText(table.getValueAt(row, 2).toString());

Object unit = table.getValueAt(row, 3);

unitText.setText(unit == null ? "" : unit.toString());

priceText.setText(table.getValueAt(row, 4).toString());

statusCheck.setSelected(table.getValueAt(row, 5).toString().equals("售罄"));

}

// 给确定和取消按钮添加监听器

Appearance.addListenerToDialog(this, applyAction, okButton, cancelButton);

// 设置所有控件的字体

Appearance.setAllFont(this, Appearance.DEFAULT\_FONT);

// 调整窗口大小以适应控件

pack();

// 显示对话框

setVisible(true);

}

private ActionListener applyAction = new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

// 去除菜品名、菜品代码、计量单位、价格文本框输入的前后空格

DataOperation.trimTextField(nameText, codeText, unitText, priceText);

boolean errorFormat = false;

double priceDouble = -1;

try {

// 将价格文本框的内容转换为double类型

priceDouble = Double.parseDouble(priceText.getText());

} catch (NumberFormatException ex) {

// 如果转换失败，则标记出错格式

errorFormat = true;

}

// 如果价格小于0，则标记出错格式

if (priceDouble < 0)

errorFormat = true;

// 如果菜品名、菜品类型、菜品代码、价格存在为空或者格式错误的情况，则弹出错误提示。

if (nameText.getText().equals("") || dishTypeCombo.getSelectedIndex() == -1 || codeText.getText().equals("")

|| errorFormat) {

JOptionPane.showMessageDialog(null, "请完善信息并确认信息格式无误！", "错误", JOptionPane.ERROR\_MESSAGE);

} else {

// 构造Dish对象，准备提交到数据库中

Dish dish = new Dish(

isNew ? -1

: (int) table.getModel()

.getValueAt(table.convertRowIndexToModel(table.getSelectedRow()), 0),

nameText.getText(),

DishTypeOperation.toDishTypeId(dishTypeCombo.getSelectedItem().toString()),

codeText.getText(),

unitText.getText(),

priceDouble,

statusCheck.isSelected() ? "sold out" : "");

if (isNew) { // 新增

// 如果菜品名重复，则弹出错误提示

if (!DishOperation.isUnique(nameText.getText())) {

JOptionPane.showMessageDialog(null, "该菜品名已存在！", "错误", JOptionPane.ERROR\_MESSAGE);

} else {

// 否则向数据库中添加菜品信息

int id = DishOperation.add(dish);

if (id != -1) {

// 将新增的菜品信息添加到表格中

((DefaultTableModel) table.getModel()).addRow(new Object[] {

id,

dish.getName(),

dish.getDishTypeId(),

dish.getCode(),

dish.getUnit(),

dish.getPrice(),

dish.getStatus()

});

// 关闭对话框

dispose();

} else {

JOptionPane.showMessageDialog(null, "添加失败！", "错误", JOptionPane.ERROR\_MESSAGE);

}

}

} else { // 修改

// 如果修改后的菜品名和其他菜品重复，则弹出错误提示

if (!DishOperation.isUnique(nameText.getText(),

(int) table.getModel().getValueAt(table.convertRowIndexToModel(table.getSelectedRow()),

0))) {

JOptionPane.showMessageDialog(null, "该菜品名已有其他菜品使用！", "错误", JOptionPane.ERROR\_MESSAGE);

} else {

// 否则更新数据库中的菜品信息，并更新表格中选中菜品行的信息

if (DishOperation.change(dish)) {

int selectedRow = table.getSelectedRow();

table.setValueAt(dish.getName(), selectedRow, 0);

table.setValueAt(DishTypeOperation.toDishTypeName(dish.getDishTypeId()), selectedRow, 1);

table.setValueAt(dish.getCode(), selectedRow, 2);

table.setValueAt(dish.getUnit(), selectedRow, 3);

table.setValueAt(dish.getPrice(), selectedRow, 4);

table.setValueAt(dish.getStatus(), selectedRow, 5);

// 关闭对话框

dispose();

} else {

JOptionPane.showMessageDialog(null, "修改失败！", "错误", JOptionPane.ERROR\_MESSAGE);

}

}

}

}

}

};

}

package gui.dialog;

import java.awt.Dimension;

import java.awt.FlowLayout;

import java.awt.GridLayout;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import javax.swing.JButton;

import javax.swing.JDialog;

import javax.swing.JLabel;

import javax.swing.JOptionPane;

import javax.swing.JPanel;

import javax.swing.JTable;

import javax.swing.JTextField;

import javax.swing.table.DefaultTableModel;

import dao.DishTypeOperation;

import entity.DishType;

import gui.MainFrame;

import tool.Appearance;

import tool.DataOperation;

public class DishTypeDialog extends JDialog {

private boolean isNew;

private JTable table;

private JLabel nameLabel = new JLabel("菜类名\*"), describeLabel = new JLabel("描述");

private JTextField nameText = new JTextField(), describeText = new JTextField();

private FlowLayout layout = new FlowLayout(FlowLayout.RIGHT);

private JPanel namePanel = new JPanel(layout), describePanel = new JPanel(layout),

buttonPanel = new JPanel(new FlowLayout(FlowLayout.CENTER));

private JButton okButton = new JButton("确定"), cancelButton = new JButton("取消");

public DishTypeDialog(boolean isNew, JTable table) { // 构造方法，接受isNew和table两个参数

setTitle((isNew ? "新增" : "修改") + "菜品分类"); // 设置对话框标题

setModal(true); // 设置模态对话框

this.isNew = isNew; // 将isNew参数赋值给成员变量isNew

this.table = table; // 将table参数赋值给成员变量table

Appearance.setToCenter(this); // 让对话框居中显示

setResizable(false); // 禁止对话框调整大小

setLayout(new GridLayout(3, 1)); // 设置对话框布局为网格布局，共分三行一列

// 设置控件尺寸

nameText.setPreferredSize(new Dimension(160, 28));

describeText.setPreferredSize(new Dimension(160, 28));

// 添加菜品分类名称输入面板

Appearance.add(namePanel, nameLabel, nameText);

// 添加菜品分类描述输入面板

Appearance.add(describePanel, describeLabel, describeText);

// 添加按钮面板

Appearance.add(buttonPanel, okButton, cancelButton);

// 添加所有面板到对话框中

Appearance.add(this, namePanel, describePanel, buttonPanel);

if (!isNew) { // 如果是修改操作

int row = table.getSelectedRow(); // 获取选中的行数

nameText.setText(table.getValueAt(row, 0).toString()); // 将选中行的菜品分类名称显示在文本框中

Object describe = table.getValueAt(row, 1);

describeText.setText(describe == null ? "" : describe.toString()); // 将选中行的菜品分类描述显示在文本域中

}

// 添加监听器到对话框中，以相应按钮点击事件

Appearance.addListenerToDialog(this, applyAction, okButton, cancelButton);

// 设置所有控件的字体为默认字体

Appearance.setAllFont(this, Appearance.DEFAULT\_FONT);

pack(); // 调整对话框大小

setVisible(true); // 显示对话框

}

// 确定按钮的响应事件，实现菜品分类的新增和修改操作

private ActionListener applyAction = new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

DataOperation.trimTextField(nameText, describeText); // 去掉文本框和文本域中的空格

if (nameText.getText().equals("")) { // 如果菜品分类名称为空

JOptionPane.showMessageDialog(null, "请完善信息并确认信息格式无误！", "错误", JOptionPane.ERROR\_MESSAGE); // 弹出错误提示

} else if (nameText.getText().equals("???")) // 如果菜品分类名称为“???”, 不允许使用

JOptionPane.showMessageDialog(null, "不允许使用“???”作为菜类名！", "错误", JOptionPane.ERROR\_MESSAGE); // 弹出错误提示

else {

// 创建一个DishType对象，用于表示新添加或修改后的菜品分类

DishType dishType = new DishType(

isNew ? -1 // 如果是新增操作，则设置id为-1

: (int) table.getModel() // 如果是修改操作，则从表格中获取当前选中行的id

.getValueAt(table.convertRowIndexToModel(table.getSelectedRow()), 0),

nameText.getText(), // 获取菜品分类名称

describeText.getText()); // 获取菜品分类描述

if (isNew) { // 新增操作

if (!DishTypeOperation.isUnique(nameText.getText())) { // 如果数据库中已经存在相同名称的菜品分类

JOptionPane.showMessageDialog(null, "该菜类已存在！", "错误", JOptionPane.ERROR\_MESSAGE); // 弹出错误提示

} else {

int id = DishTypeOperation.add(dishType); // 将菜品分类添加到数据库中，并返回自增的id值

if (id != -1) { // 如果添加成功

// 在表格中添加一行表示当前新增的菜品分类

((DefaultTableModel) table.getModel()).addRow(new Object[] {

id,

dishType.getName(),

dishType.getDescribe()

});

// 刷新主界面的菜单列表

((MainFrame) table.getRootPane().getParent()).dishPanel.refreshDishTable();

dispose(); // 关闭对话框

} else {

JOptionPane.showMessageDialog(null, "添加失败！", "错误", JOptionPane.ERROR\_MESSAGE); // 弹出错误提示

}

}

} else { // 修改操作

if (!DishTypeOperation.isUnique(nameText.getText(),

(int) table.getModel().getValueAt(table.convertRowIndexToModel(table.getSelectedRow()),

0))) { // 如果新的菜品分类名称与其他菜品分类名称冲突

JOptionPane.showMessageDialog(null, "该菜类名与其他菜类冲突！", "错误", JOptionPane.ERROR\_MESSAGE); // 弹出错误提示

} else {

if (DishTypeOperation.change(dishType)) { // 如果修改成功

int selectedRow = table.getSelectedRow(); // 获取选中的行数

// 更新表格中对应的菜品分类名称和描述

table.setValueAt(dishType.getName(), selectedRow, 0);

table.setValueAt(dishType.getDescribe(), selectedRow, 1);

// 刷新主界面的菜单列表

((MainFrame) table.getRootPane().getParent()).dishPanel.refreshDishTable();

dispose(); // 关闭对话框

} else {

JOptionPane.showMessageDialog(null, "修改失败！", "错误", JOptionPane.ERROR\_MESSAGE); // 弹出错误提示

}

}

}

}

}

};

}

package gui.dialog;

import java.awt.Dimension;

import java.awt.FlowLayout;

import java.awt.GridLayout;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import java.sql.Date;

import javax.swing.ButtonGroup;

import javax.swing.JButton;

import javax.swing.JCheckBox;

import javax.swing.JDialog;

import javax.swing.JLabel;

import javax.swing.JOptionPane;

import javax.swing.JPanel;

import javax.swing.JRadioButton;

import javax.swing.JTable;

import javax.swing.JTextField;

import javax.swing.table.DefaultTableModel;

import dao.EmployeeOperation;

import entity.Employee;

import tool.Appearance;

import tool.DataOperation;

//员工信息对话框类，用于新增或修改员工信息

public class EmployeeDialog extends JDialog {

private boolean isNew; // 是否为新增员工

private JTable table; // 操作的表格

// 控件变量

private JTextField nameText = new JTextField();

private JRadioButton maleRadio = new JRadioButton("男");

private JRadioButton femaleRadio = new JRadioButton("女");

private ButtonGroup sexGroup = new ButtonGroup();

private JTextField yearText = new JTextField();

private JLabel yearLabel = new JLabel("年");

private JTextField monthText = new JTextField();

private JLabel monthLabel = new JLabel("月");

private JTextField dayText = new JTextField();

private JLabel dayLabel = new JLabel("日");

private JTextField identityIdText = new JTextField();

private JTextField addressText = new JTextField();

private JTextField telText = new JTextField();

private JTextField positionText = new JTextField();

private JCheckBox freezeCheck = new JCheckBox("是否在职");

// 组件面板变量

private JPanel namePanel = new JPanel();

private JPanel sexPanel = new JPanel();

private JPanel birthdayPanel = new JPanel();

private JPanel identityIdPanel = new JPanel();

private JPanel addressPanel = new JPanel();

private JPanel telPanel = new JPanel();

private JPanel positionPanel = new JPanel();

private JPanel freezePanel = new JPanel();

private JPanel buttonPanel = new JPanel();

// 控件标签变量

private JLabel nameLabel = new JLabel("姓名：");

private JLabel sexLabel = new JLabel("性别：");

private JLabel birthdayLabel = new JLabel("生日：");

private JLabel identityIdLabel = new JLabel("身份证号：");

private JLabel addressLabel = new JLabel("地址：");

private JLabel telLabel = new JLabel("电话：");

private JLabel positionLabel = new JLabel("职位：");

private JLabel messageLabel = new JLabel("");

// 操作按钮

private JButton okButton = new JButton("确认");

private JButton cancelButton = new JButton("取消");

// 员工信息对话框构造方法

public EmployeeDialog(boolean isNew, JTable table) {

setTitle((isNew ? "新增" : "修改") + "员工"); // 设置对话框标题

setModal(true); // 设置对话框为模态

this.isNew = isNew; // 保存是否为新增员工

this.table = table; // 保存操作表格

Appearance.setToCenter(this); // 设置对话框在屏幕中心

setResizable(false); // 设置对话框不可改变大小

setLayout(new GridLayout(9, 1)); // 设置布局为9行1列，使用网格布局

// 设置文本框水平对齐方式

yearText.setHorizontalAlignment(JTextField.RIGHT);

monthText.setHorizontalAlignment(JTextField.RIGHT);

dayText.setHorizontalAlignment(JTextField.RIGHT);

// 设置控件尺寸

nameText.setPreferredSize(new Dimension(160, 28));

maleRadio.setPreferredSize(new Dimension(80, 28));

femaleRadio.setPreferredSize(new Dimension(80, 28));

yearText.setPreferredSize(new Dimension(48, 28));

yearLabel.setPreferredSize(new Dimension(20, 28));

monthText.setPreferredSize(new Dimension(26, 28));

monthLabel.setPreferredSize(new Dimension(20, 28));

dayText.setPreferredSize(new Dimension(26, 28));

dayLabel.setPreferredSize(new Dimension(20, 28));

identityIdText.setPreferredSize(new Dimension(160, 28));

addressText.setPreferredSize(new Dimension(160, 28));

telText.setPreferredSize(new Dimension(160, 28));

positionText.setPreferredSize(new Dimension(160, 28));

// 添加组件到相应的面板中

Appearance.add(namePanel, nameLabel, nameText);

Appearance.add(sexPanel, sexLabel, maleRadio, femaleRadio);

sexGroup.add(maleRadio);

sexGroup.add(femaleRadio);

Appearance.add(birthdayPanel, birthdayLabel, yearText, yearLabel, monthText, monthLabel, dayText, dayLabel);

Appearance.add(identityIdPanel, identityIdLabel, identityIdText);

Appearance.add(addressPanel, addressLabel, addressText);

Appearance.add(telPanel, telLabel, telText);

Appearance.add(positionPanel, positionLabel, positionText);

freezePanel.add(freezeCheck);

Appearance.add(buttonPanel, okButton, cancelButton);

Appearance.add(this, namePanel, sexPanel, birthdayPanel, identityIdPanel, addressPanel, telPanel, positionPanel,

freezePanel, buttonPanel);

// 若为修改员工，将原员工信息填充到对话框文本框中

if (!isNew) {

int row = table.getSelectedRow();

nameText.setText(table.getValueAt(row, 0).toString());

if (table.getValueAt(row, 1).equals("male")) {

maleRadio.setSelected(true);

} else {

femaleRadio.setSelected(true);

}

Object identityId = table.getValueAt(row, 3), address = table.getValueAt(row, 4),

tel = table.getValueAt(row, 5);

String[] date = DataOperation.dateToString((Date) table.getValueAt(row, 2));

yearText.setText(date[0]);

monthText.setText(date[1]);

dayText.setText(date[2]);

identityIdText.setText(identityId == null ? "" : identityId.toString());

addressText.setText(address == null ? "" : address.toString());

telText.setText(tel == null ? "" : tel.toString());

positionText.setText(table.getValueAt(row, 6).toString());

freezeCheck.setSelected(table.getValueAt(row, 7).toString().equals("yes"));

}

// 添加监听器

Appearance.addListenerToDialog(this, applyAction, okButton, cancelButton);

Appearance.setAllFont(this, Appearance.DEFAULT\_FONT); // 设置字体

pack(); // 自适应大小

setVisible(true); // 显示对话框

}

// 确认按钮的监听器

private ActionListener applyAction = new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

// 去除文本框中多余的空格

DataOperation.trimTextField(nameText, yearText, monthText, dayText, identityIdText, addressText, telText,

positionText);

// 将日期字符串转换为Date对象

Date date = DataOperation

.stringToDate(yearText.getText() + "-" + monthText.getText() + "-" + dayText.getText());

// 判断姓名、性别、生日和职位是否填写完整，日期是否有效

if (nameText.getText().equals("") || (!maleRadio.isSelected() && !femaleRadio.isSelected())

|| (date == null && (!yearText.getText().equals("") || !monthText.getText().equals("")

|| !dayText.getText().equals("")))// 日期无效，且年月日文本域至少有一个不为空

|| positionText.getText().equals("")) {

JOptionPane.showMessageDialog(null, "请完善信息并确认信息格式无误！", "错误", JOptionPane.ERROR\_MESSAGE);

} else {

// 构造员工对象

Employee employee = new Employee(

isNew ? -1

: (int) table.getModel()

.getValueAt(table.convertRowIndexToModel(table.getSelectedRow()), 0),

nameText.getText(),

maleRadio.isSelected() ? "male" : "female",

date,

identityIdText.getText(),

addressText.getText(),

telText.getText(),

positionText.getText(),

freezeCheck.isSelected() ? "yes" : "no");

if (isNew) { // 新增

int id = EmployeeOperation.add(employee); // 在数据库中添加员工，并返回id

if (id != -1) { // 添加成功

((DefaultTableModel) table.getModel()).addRow(new Object[] {

id,

employee.getName(),

employee.getSex(),

employee.getBirthday(),

employee.getIdentityId(),

employee.getAddress(),

employee.getTel(),

employee.getPosition(),

employee.getFreeze()

});

dispose(); // 销毁对话框

} else { // 添加失败

JOptionPane.showMessageDialog(null, "添加失败！", "错误", JOptionPane.ERROR\_MESSAGE);

}

} else { // 修改

if (EmployeeOperation.change(employee)) { // 在数据库中修改员工信息

int selectedRow = table.getSelectedRow();

// 更新对应表格行的数据

table.setValueAt(employee.getName(), selectedRow, 0);

table.setValueAt(employee.getSex(), selectedRow, 1);

table.setValueAt(employee.getBirthday(), selectedRow, 2);

table.setValueAt(employee.getIdentityId(), selectedRow, 3);

table.setValueAt(employee.getAddress(), selectedRow, 4);

table.setValueAt(employee.getTel(), selectedRow, 5);

table.setValueAt(employee.getPosition(), selectedRow, 6);

table.setValueAt(employee.getFreeze(), selectedRow, 7);

dispose(); // 销毁对话框

} else { // 修改失败

JOptionPane.showMessageDialog(null, "修改失败！", "错误", JOptionPane.ERROR\_MESSAGE);

}

}

}

}

};

}

package gui.dialog;

import java.awt.BorderLayout;

import java.awt.Color;

import java.awt.Dimension;

import java.awt.FlowLayout;

import java.awt.GridBagConstraints;

import java.awt.GridBagLayout;

import java.awt.GridLayout;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import java.awt.event.MouseAdapter;

import java.awt.event.MouseEvent;

import java.awt.event.WindowAdapter;

import java.awt.event.WindowEvent;

import java.sql.Timestamp;

import java.util.Iterator;

import java.util.LinkedHashMap;

import java.util.List;

import java.util.Map;

import javax.swing.JButton;

import javax.swing.JComboBox;

import javax.swing.JFrame;

import javax.swing.JLabel;

import javax.swing.JOptionPane;

import javax.swing.JPanel;

import javax.swing.JScrollPane;

import javax.swing.JTable;

import javax.swing.JTextField;

import javax.swing.ListSelectionModel;

import javax.swing.event.TableModelEvent;

import javax.swing.event.TableModelListener;

import javax.swing.table.DefaultTableModel;

import javax.swing.table.TableRowSorter;

import dao.CustomerOperation;

import dao.DishOperation;

import dao.DishTypeOperation;

import dao.OrderOperation;

import dao.TableOperation;

import entity.Dish;

import entity.Order;

import entity.OrderItem;

import entity.Table;

import gui.MainFrame;

import tool.Appearance;

import tool.DataOperation;

public class OrderDialog extends JFrame {

private int selectedRow, seating;

private double money;

private String tableNo;

private DefaultTableModel menuModel = new DefaultTableModel(new Object[][] {},

new String[] { "类别", "菜品", "单位", "价格" }) {

@Override

public Class<?> getColumnClass(int column) { // 使表格模型能正确识别类型以便排序

Class<?> result = Object.class;

if (getRowCount() > 0) {

Object value = getValueAt(0, column);

if (value != null)

result = value.getClass();

}

return result;

}

@Override

public boolean isCellEditable(int row, int column) { // 使表格不可编辑

return false;

}

};

private DefaultTableModel selectedModel = new DefaultTableModel(new Object[][] {},

new String[] { "菜品", "单价", "数量（双击更改）" }) {

@Override

public boolean isCellEditable(int row, int column) { // 使表格仅最后一列可编辑

return column == 2;

}

};

private DefaultTableModel tableModel, orderModel;

private MainFrame mainFrame;

private Map<String, String> cardId\_customerName = new LinkedHashMap<String, String>(); // 存放贵宾卡号和客户名，与下拉列表框一一对应

private JComboBox<String> customerCombo = new JComboBox<String>();

private JTable menuTable = new JTable(menuModel), selectedTable = new JTable(selectedModel);

private JLabel customerLabel = new JLabel("客户 - 贵宾卡号"), quantityLabel = new JLabel("人数"),

menuLabel = new JLabel("菜单（双击选择）"), selectedLabel = new JLabel("已选（双击退选）"), moneyLabel = new JLabel();

private JTextField quantityText = new JTextField();

private JButton orderButton = new JButton("下单");

private GridBagLayout gridBagLayout = new GridBagLayout();

private GridBagConstraints gridBagConstraints = new GridBagConstraints();

private JPanel customerPanel = new JPanel(), quantityPanel = new JPanel(new FlowLayout(FlowLayout.RIGHT)),

topPanel = new JPanel(new GridLayout(1, 2)), menuPanel = new JPanel(new BorderLayout()),

selectedPanel = new JPanel(new BorderLayout()), mainPanel = new JPanel(gridBagLayout);

public OrderDialog(JTable table) {

selectedRow = table.convertRowIndexToModel(table.getSelectedRow());

mainFrame = (MainFrame) table.getRootPane().getParent();

mainFrame.operationPanel.setVisible(false);

mainFrame.count++;

tableModel = (DefaultTableModel) table.getModel();

orderModel = (DefaultTableModel) mainFrame.orderPanel.table.getModel();

tableNo = (String) tableModel.getValueAt(selectedRow, 1);

seating = (int) tableModel.getValueAt(selectedRow, 2);

setTitle("点菜（餐台编号：" + tableNo + "）");

Appearance.setToCenter(this);

setSize(800, 600);

menuTable.setRowSorter(new TableRowSorter<DefaultTableModel>(menuModel)); // 表格排序器

// 设置控件尺寸

quantityText.setPreferredSize(new Dimension(64, 28));

customerCombo.setPreferredSize(new Dimension(192, 28));

// 网格包布局

gridBagConstraints.fill = GridBagConstraints.BOTH;

gridBagConstraints.anchor = GridBagConstraints.WEST;

gridBagConstraints.weightx = 0.6;

gridBagConstraints.weighty = 1;

gridBagLayout.setConstraints(menuPanel, gridBagConstraints);

gridBagConstraints.anchor = GridBagConstraints.EAST;

gridBagConstraints.weightx = 0.4;

gridBagLayout.setConstraints(selectedPanel, gridBagConstraints);

menuTable.setSelectionBackground(new Color(212, 251, 121));

selectedTable.setSelectionBackground(new Color(255, 252, 121));

menuTable.setSelectionForeground(Color.BLACK);

selectedTable.setSelectionForeground(Color.BLACK);

menuTable.getTableHeader().setReorderingAllowed(false);

menuTable.setSelectionMode(ListSelectionModel.SINGLE\_SELECTION);

menuTable.setRowHeight(24);

menuTable.getTableHeader().setFont(Appearance.DEFAULT\_TITLE\_FONT);

selectedTable.getTableHeader().setReorderingAllowed(false);

selectedTable.setSelectionMode(ListSelectionModel.SINGLE\_SELECTION);

selectedTable.setRowHeight(24);

selectedTable.getTableHeader().setFont(Appearance.DEFAULT\_TITLE\_FONT);

cardId\_customerName = OrderOperation.loadCustomerNameAndCardId();

for (Map.Entry<String, String> entry : cardId\_customerName.entrySet()) {

customerCombo.addItem(entry.getValue() + " - " + entry.getKey());

}

List<Dish> list = DishOperation.load();

for (Dish dish : list) {

if (dish.getStatus().equals("")) {

menuModel.addRow(new Object[] {

DishTypeOperation.toDishTypeName(dish.getDishTypeId()),

dish.getName(),

dish.getUnit(),

dish.getPrice()

});

}

}

Appearance.add(customerPanel, customerLabel, customerCombo);

Appearance.add(quantityPanel, moneyLabel, quantityLabel, quantityText, orderButton);

Appearance.add(topPanel, customerPanel, quantityPanel);

menuPanel.add(menuLabel, BorderLayout.NORTH);

menuPanel.add(new JScrollPane(menuTable), BorderLayout.CENTER);

selectedPanel.add(selectedLabel, BorderLayout.NORTH);

selectedPanel.add(new JScrollPane(selectedTable), BorderLayout.CENTER);

Appearance.add(mainPanel, menuPanel, selectedPanel);

add(topPanel, BorderLayout.NORTH);

add(mainPanel, BorderLayout.CENTER);

addWindowListener(closingAction);

menuTable.addMouseListener(addAction);

selectedTable.addMouseListener(rmAction);

selectedModel.addTableModelListener(changeAction);

orderButton.addActionListener(orderAction);

Appearance.fitTableColumns(table);

Appearance.setAllFont(this, Appearance.DEFAULT\_FONT);

moneyLabel.setFont(Appearance.DEFAULT\_BIG\_FONT);

setVisible(true);

}

// 窗口关闭事件

private WindowAdapter closingAction = new WindowAdapter() {

@Override

public void windowClosing(WindowEvent e) {

// 减少餐桌数

mainFrame.count--;

// 判断是否需要隐藏操作面板

mainFrame.operationPanel.setVisible(mainFrame.count == 0);

// 修改指定餐桌的状态为“free”

if (TableOperation.change(new Table((int) tableModel.getValueAt(selectedRow, 0),

(String) tableModel.getValueAt(selectedRow, 1), (int) tableModel.getValueAt(selectedRow, 2),

"free"))) {

tableModel.setValueAt("free", selectedRow, 3);

mainFrame.tablePanel.selectAction.valueChanged(null);

} else {

JOptionPane.showMessageDialog(null, "未能释放餐台的占用状态！", "错误", JOptionPane.ERROR\_MESSAGE);

}

}

};

// 双击菜单添加菜品

private MouseAdapter addAction = new MouseAdapter() {

@Override

public void mousePressed(MouseEvent e) {

int selection = menuTable.getSelectedRow();

if (e.getClickCount() == 2 && selection != -1) {

// 在已选菜品列表中添加所选中的菜品

selectedModel.addRow(

new Object[] { menuTable.getValueAt(selection, 1), menuTable.getValueAt(selection, 3), 1 });

}

}

};

// 双击已选菜品退选

private MouseAdapter rmAction = new MouseAdapter() {

@Override

public void mousePressed(MouseEvent e) {

int selection = selectedTable.getSelectedRow();

if (e.getClickCount() == 2 && selection != -1 && selectedTable.getSelectedColumn() != 2) {

// 将所选中的菜品从已选菜品列表中删除

selectedModel.removeRow(selection);

}

}

};

// 下单

private ActionListener orderAction = new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

boolean errorFormat = false;

int quantityInt = -1;

try {

// 获取填写的人数

quantityInt = Integer.parseInt(quantityText.getText());

} catch (NumberFormatException ex) {

errorFormat = true;

}

// 判断是否填写正确、选择了菜品并且人数不多于座位数

if (selectedTable.getRowCount() > 0 && !errorFormat && quantityInt > 0 && quantityInt <= seating) {

// 创建订单对象

Order order = new Order(-1,

DataOperation.generateOrderNo(),

TableOperation.toTableId(tableNo),

new Timestamp(System.currentTimeMillis()),

money,

CustomerOperation.toCustomerId(getKeyByIndex(customerCombo.getSelectedIndex())),

"not paid",

quantityInt);

// 添加订单到数据库中，并获取添加后的订单ID

int orderId = OrderOperation.addOrder(order);

if (orderId != -1) {

// 添加订单到表格中

orderModel.addRow(new Object[] {

orderId,

order.getOrderNo(),

TableOperation.toTableNo(order.getTableId()),

order.getCreatetime().toString(),

order.getMoney(),

CustomerOperation.toCustomerName(order.getCustomerId()),

order.getStatus(),

order.getNumber()

});

for (int i = 0; i < selectedModel.getRowCount(); i++) {

// 添加订单明细到数据库中

OrderItem orderItem = new OrderItem(-1, orderId,

DishOperation.toDishId(selectedModel.getValueAt(i, 0).toString()),

Integer.parseInt(selectedModel.getValueAt(i, 2).toString()));

int orderItemId = OrderOperation.addOrderItem(orderItem);

if (orderItemId == -1) {

JOptionPane.showMessageDialog(null, "订单明细添加失败！", "错误", JOptionPane.ERROR\_MESSAGE);

break;

}

}

JOptionPane.showMessageDialog(null, "已下单，订单已生成。", "提示", JOptionPane.PLAIN\_MESSAGE);

// 减少餐桌数并判断是否需要隐藏操作面板

mainFrame.count--;

mainFrame.operationPanel.setVisible(mainFrame.count == 0);

dispose();

} else {

JOptionPane.showMessageDialog(null, "订单添加失败！", "错误", JOptionPane.ERROR\_MESSAGE);

}

} else {

JOptionPane.showMessageDialog(null, "请确保人数不多于餐台座位数。\n请确保已正确填写人数且有选择菜品！", "错误", JOptionPane.ERROR\_MESSAGE);

}

}

};

// 更改填写错误的数量，然后计算总价

private TableModelListener changeAction = new TableModelListener() {

@Override

public void tableChanged(TableModelEvent e) {

money = 0;

for (int i = 0; i < selectedTable.getRowCount(); i++) {

int quantityInt = -1;

try {

quantityInt = Integer.parseInt(selectedTable.getValueAt(i, 2).toString());

} catch (NumberFormatException ex) {

}

if (quantityInt < 1) { // 如果填写的数量小于1，则默认数量为1

quantityInt = 1;

selectedTable.setValueAt(quantityInt, i, 2);

return;

}

// 计算总价

money += (double) selectedTable.getValueAt(i, 1) \* quantityInt;

}

// 更新总价的显示

moneyLabel.setText("总价：" + money + "\t");

}

};

// 获取下拉列表框中选中项对应的键值

private String getKeyByIndex(int index) {

Iterator<Map.Entry<String, String>> iterator = cardId\_customerName.entrySet().iterator();

Map.Entry<String, String> entry = null;

for (int i = 0; i <= index; i++) {

entry = iterator.next();

}

return entry.getKey();

}

}

package gui.dialog;

import java.awt.Dimension;

import java.awt.FlowLayout;

import java.awt.GridLayout;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import java.awt.event.WindowAdapter;

import java.awt.event.WindowEvent;

import javax.swing.JButton;

import javax.swing.JDialog;

import javax.swing.JLabel;

import javax.swing.JOptionPane;

import javax.swing.JPanel;

import javax.swing.JTable;

import javax.swing.JTextField;

import javax.swing.table.DefaultTableModel;

import dao.OrderOperation;

import dao.TableOperation;

import entity.Table;

import gui.MainFrame;

import tool.Appearance;

public class PayDialog extends JDialog {

private DefaultTableModel model;

private MainFrame mainFrame;

private int selectedRow;

private boolean paid = false;

private JLabel requiredLabel = new JLabel("应付"), actualLabel = new JLabel("实付"), discountLabel = new JLabel("优惠"),

cashLabel = new JLabel("找零");

private JTextField requiredText = new JTextField(), actualText = new JTextField(),

discountText = new JTextField("0.0"),

cashText = new JTextField();

private FlowLayout layout = new FlowLayout(FlowLayout.RIGHT);

private JPanel requiredPanel = new JPanel(layout), actualPanel = new JPanel(layout),

discountPanel = new JPanel(layout), cashPanel = new JPanel(layout),

buttonPanel = new JPanel(new FlowLayout(FlowLayout.CENTER));

private JButton payButton = new JButton("支付");

public PayDialog(JTable table) {

setTitle("结账");

mainFrame = (MainFrame) table.getRootPane().getParent();

mainFrame.operationPanel.setVisible(false);

mainFrame.count++;

selectedRow = table.convertRowIndexToModel(table.getSelectedRow());

model = (DefaultTableModel) table.getModel();

model.setValueAt("paying", selectedRow, 6);

requiredText.setText(model.getValueAt(selectedRow, 4).toString());

Appearance.setToCenter(this);

setResizable(false);

setLayout(new GridLayout(5, 1));

// 设置控件尺寸

requiredText.setPreferredSize(new Dimension(256, 48));

actualText.setPreferredSize(new Dimension(256, 48));

discountText.setPreferredSize(new Dimension(256, 48));

cashText.setPreferredSize(new Dimension(256, 48));

requiredText.setEditable(false);

cashText.setEditable(false);

Appearance.add(requiredPanel, requiredLabel, requiredText);

Appearance.add(actualPanel, actualLabel, actualText);

Appearance.add(discountPanel, discountLabel, discountText);

Appearance.add(cashPanel, cashLabel, cashText);

buttonPanel.add(payButton);

Appearance.add(this, requiredPanel, actualPanel, discountPanel, cashPanel, buttonPanel);

addWindowListener(closingAction);

payButton.addActionListener(payAction);

actualText.addActionListener(payAction);

discountText.addActionListener(payAction);

Appearance.setAllFont(this, Appearance.DEFAULT\_HUGE\_FONT);

pack();

setVisible(true);

}

private WindowAdapter closingAction = new WindowAdapter() { // 窗口关闭事件

@Override

public void windowClosing(WindowEvent e) {

if (!paid) {

mainFrame.count--;

mainFrame.operationPanel.setVisible(mainFrame.count == 0);

model.setValueAt("not paid", selectedRow, 6);

mainFrame.orderPanel.selectAction.valueChanged(null);

}

}

};

private ActionListener payAction = new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

boolean errorFormat = false;

double requiredDouble = -1, actualDouble = -1, discountDouble = -1;

try {

requiredDouble = Double.parseDouble(requiredText.getText());

actualDouble = Double.parseDouble(actualText.getText());

discountDouble = Double.parseDouble(discountText.getText());

} catch (NumberFormatException ex) {

errorFormat = true;

}

if (errorFormat) {

JOptionPane.showMessageDialog(null, "请填写正确的数值！", "错误", JOptionPane.ERROR\_MESSAGE);

return;

}

if (requiredDouble - discountDouble > actualDouble) {

JOptionPane.showMessageDialog(null, "支付的金额不足！", "错误", JOptionPane.ERROR\_MESSAGE);

return;

}

DefaultTableModel tableModel = (DefaultTableModel) mainFrame.tablePanel.table.getModel();

int tableRow = -1;

for (int i = 0; i < tableModel.getRowCount(); i++) { // 在餐台表找到该条订单的餐台编号所在行

if (model.getValueAt(selectedRow, 2).toString().equals(tableModel.getValueAt(i, 1).toString())) {

tableRow = i;

break;

}

}

if (payButton.getText().equals("支付")) {

if (tableRow != -1)

if (TableOperation.change(new Table((int) tableModel.getValueAt(tableRow, 0),

(String) tableModel.getValueAt(tableRow, 1), (int) tableModel.getValueAt(tableRow, 2),

"free"))) {

tableModel.setValueAt("free", tableRow, 3);

mainFrame.tablePanel.selectAction.valueChanged(null);

} else

JOptionPane.showMessageDialog(null, "未能释放餐台的占用状态！", "错误", JOptionPane.ERROR\_MESSAGE);

if (OrderOperation.changeOrder((int) model.getValueAt(selectedRow, 0))) {

paid = true;

mainFrame.count--;

mainFrame.operationPanel.setVisible(mainFrame.count == 0);

model.setValueAt("paid", selectedRow, 6);

mainFrame.orderPanel.selectAction.valueChanged(null);

actualText.setEditable(false);

discountText.setEditable(false);

cashText.setText(actualDouble - (requiredDouble - discountDouble) + ""); // 将double转为String

payButton.setText("支付成功！点击关闭");

} else

JOptionPane.showMessageDialog(null, "支付失败！", "错误", JOptionPane.ERROR\_MESSAGE);

} else {

dispose();

}

}

};

}

**package** gui.dialog;

**import** java.awt.Dimension;

**import** java.awt.FlowLayout;

**import** java.awt.GridLayout;

**import** java.awt.event.ActionEvent;

**import** java.awt.event.ActionListener;

**import** javax.swing.JButton;

**import** javax.swing.JCheckBox;

**import** javax.swing.JDialog;

**import** javax.swing.JLabel;

**import** javax.swing.JOptionPane;

**import** javax.swing.JPanel;

**import** javax.swing.JTable;

**import** javax.swing.JTextField;

**import** javax.swing.table.DefaultTableModel;

**import** dao.TableOperation;

**import** entity.Table;

**import** tool.Appearance;

**import** tool.DataOperation;

**public** **class** TableDialog **extends** JDialog {

**private** **boolean** isNew; // 是否是新增操作

**private** JTable table; // 对应的表格

**private** JLabel noLabel = **new** JLabel("餐台编号\*"), seatingLabel = **new** JLabel("座位数\*"); // 控件标签

**private** JTextField noText = **new** JTextField(), seatingText = **new** JTextField(); // 输入框

**private** FlowLayout layout = **new** FlowLayout(FlowLayout.***RIGHT***); // 控件布局

**private** JPanel noPanel = **new** JPanel(layout), seatingPanel = **new** JPanel(layout), // 餐台编号和座位数面板

statusPanel = **new** JPanel(**new** FlowLayout(FlowLayout.***CENTER***)), // 繁忙状态面板

buttonPanel = **new** JPanel(**new** FlowLayout(FlowLayout.***CENTER***)); // 操作按钮面板

**private** JCheckBox statusCheck = **new** JCheckBox("busy"); // 繁忙状态复选框

**private** JButton okButton = **new** JButton("确定"), cancelButton = **new** JButton("取消"); // 操作按钮

**public** TableDialog(**boolean** isNew, JTable table) {

setTitle(isNew ? "新增" : "修改" + "餐台");

setModal(**true**); // 设置为模态对话框

**this**.isNew = isNew;

**this**.table = table;

Appearance.*setToCenter*(**this**); // 设置对话框居中

setResizable(**false**); // 不可改变大小

setLayout(**new** GridLayout(4, 1)); // 设置布局为网格布局

// 设置控件尺寸

noText.setPreferredSize(**new** Dimension(160, 28));

seatingText.setPreferredSize(**new** Dimension(160, 28));

Appearance.*add*(noPanel, noLabel, noText); // 添加餐台编号面板

Appearance.*add*(seatingPanel, seatingLabel, seatingText); // 添加座位数面板

statusPanel.add(statusCheck); // 添加繁忙状态复选框

Appearance.*add*(buttonPanel, okButton, cancelButton); // 添加操作按钮面板

Appearance.*add*(**this**, noPanel, seatingPanel, statusPanel, buttonPanel); // 将所有面板添加到对话框中

**if** (!isNew) { // 如果是修改操作

**int** row = table.getSelectedRow(); // 获取选中的行

noText.setText(table.getValueAt(row, 0).toString()); // 将餐台编号的值设为所选行对应列的值

seatingText.setText(table.getValueAt(row, 1).toString()); // 将座位数的值设为所选行对应列的值

statusCheck.setSelected(table.getValueAt(row, 2).toString().equals("busy")); // 将繁忙状态设为所选行对应列的值

}

Appearance.*addListenerToDialog*(**this**, applyAction, okButton, cancelButton); // 为确定和取消按钮添加监听器

Appearance.*setAllFont*(**this**, Appearance.***DEFAULT\_FONT***); // 设置字体

pack(); // 调整对话框大小以适应控件

setVisible(**true**); // 显示对话框

}

// 确定按钮的监听器

**private** ActionListener applyAction = **new** ActionListener() {

@Override

**public** **void** actionPerformed(ActionEvent e) {

DataOperation.*trimTextField*(noText, seatingText); // 去除输入框中的空格

**boolean** errorFormat = **false**; // 标记输入格式是否错误

**int** seatingInt = -1; // 座位数的整型变量

**try** {

seatingInt = Integer.*parseInt*(seatingText.getText()); // 将座位数转为整型

} **catch** (NumberFormatException ex) { // 数字格式错误

errorFormat = **true**;

}

**if** (seatingInt <= 0) // 座位数必须大于0

errorFormat = **true**;

**if** (noText.getText().equals("") || seatingText.getText().equals("") || errorFormat) { // 如果输入框为空或格式错误

JOptionPane.*showMessageDialog*(**null**, "请完善信息并确认信息格式无误！", "错误", JOptionPane.***ERROR\_MESSAGE***);

} **else** **if** (noText.getText().equals("???")) // 不允许使用“???”作为餐台编号

JOptionPane.*showMessageDialog*(**null**, "不允许使用“???”作为餐台编号！", "错误", JOptionPane.***ERROR\_MESSAGE***);

**else** {

Table table\_ = **new** Table(

isNew ? -1 // 如果是新增操作，表格ID设为-1

: (**int**) table.getModel()

.getValueAt(table.convertRowIndexToModel(table.getSelectedRow()), 0), // 如果是修改操作，表格ID设为选中行对应列的值

noText.getText(), // 获取餐台编号

seatingInt, // 获取座位数

statusCheck.isSelected() ? "busy" : "free"); // 获取繁忙状态

**if** (isNew) { // 新增操作

**if** (!TableOperation.*isUnique*(noText.getText())) { // 判断餐台编号是否唯一

JOptionPane.*showMessageDialog*(**null**, "该餐台编号已存在！", "错误", JOptionPane.***ERROR\_MESSAGE***);

} **else** {

**int** id = TableOperation.*add*(table\_); // 将数据保存到数据库

**if** (id != -1) { // 如果保存成功

// 在表格中添加一行新的数据

((DefaultTableModel) table.getModel()).addRow(**new** Object[] {

id, // ID部分由数据库分配，不需要手动输入

table\_.getNo(),

table\_.getSeating(),

table\_.getStatus()

});

dispose(); // 关闭对话框

} **else** {

JOptionPane.*showMessageDialog*(**null**, "添加失败！", "错误", JOptionPane.***ERROR\_MESSAGE***);

}

}

} **else** { // 修改操作

**if** (!TableOperation.*isUnique*(noText.getText(),

(**int**) table.getModel().getValueAt(table.convertRowIndexToModel(table.getSelectedRow()),

0))) { // 判断餐台编号是否唯一

JOptionPane.*showMessageDialog*(**null**, "该餐台编号已有其他餐台使用！", "错误", JOptionPane.***ERROR\_MESSAGE***);

} **else** {

**if** (TableOperation.*change*(table\_)) { // 将数据保存到数据库

// 在表格中更新所选行对应的数据

**int** selectedRow = table.getSelectedRow();

table.setValueAt(table\_.getNo(), selectedRow, 0);

table.setValueAt(table\_.getSeating(), selectedRow, 1);

table.setValueAt(table\_.getStatus(), selectedRow, 2);

dispose(); // 关闭对话框

} **else** {

JOptionPane.*showMessageDialog*(**null**, "修改失败！", "错误", JOptionPane.***ERROR\_MESSAGE***);

}

}

}

}

}

};

}

**gui.panel包**

package gui.panel;

import java.awt.BorderLayout;

import java.util.List;

import javax.swing.JPanel;

import javax.swing.JScrollPane;

import javax.swing.JTable;

import javax.swing.table.DefaultTableModel;

import dao.CustomerOperation;

import entity.Customer;

import tool.Appearance;

public class CustomerPanel extends JPanel {

private DefaultTableModel model = new DefaultTableModel(new Object[][] {}, new String[] {

"序号", "姓名", "性别", "单位", "电话", "贵宾卡号"

}) {

@Override

public Class<?> getColumnClass(int column) { // 使表格模型能正确识别类型以便排序

Class<?> result = Object.class;

if (getRowCount() > 0) {

Object value = getValueAt(0, column);

if (value != null)

result = value.getClass();

}

return result;

}

@Override

public boolean isCellEditable(int row, int column) { // 使表格不可编辑

return false;

}

};

public JTable table = new JTable(model);

public CustomerPanel() {

setLayout(new BorderLayout());

Appearance.tableInit(table);

List<Customer> list = CustomerOperation.load();

for (int i = 0; i < list.size(); i++) {

Customer customer = list.get(i);

model.addRow(new Object[] {

customer.getId(),

customer.getName(),

customer.getSex(),

customer.getCompany(),

customer.getTel(),

customer.getCardId()

});

}

Appearance.fitTableColumns(table);

add(new JScrollPane(table));

}

}

package gui.panel;

import java.awt.BorderLayout;

import java.util.List;

import javax.swing.JPanel;

import javax.swing.JScrollPane;

import javax.swing.JTable;

import javax.swing.table.DefaultTableModel;

import dao.DishOperation;

import dao.DishTypeOperation;

import entity.Dish;

import tool.Appearance;

public class DishPanel extends JPanel {

private DefaultTableModel model = new DefaultTableModel(new Object[][] {}, new String[] {

"序号", "菜品名", "类别", "菜品代码", "单位", "价格", "状态"

}) {

@Override

public Class<?> getColumnClass(int column) { // 使表格模型能正确识别类型以便排序

Class<?> result = Object.class;

if (getRowCount() > 0) {

Object value = getValueAt(0, column);

if (value != null)

result = value.getClass();

}

return result;

}

@Override

public boolean isCellEditable(int row, int column) { // 使表格不可编辑

return false;

}

};

public JTable table = new JTable(model);

public DishPanel() {

setLayout(new BorderLayout());

Appearance.tableInit(table);

refreshDishTable();

Appearance.setCellColor(table, "状态", "");

Appearance.fitTableColumns(table);

add(new JScrollPane(table));

}

public void refreshDishTable() {

List<Dish> dishList = DishOperation.load();

model.getDataVector().clear();

for (int i = 0; i < dishList.size(); i++) {

Dish dish = dishList.get(i);

model.addRow(new Object[] {

dish.getId(),

dish.getName(),

DishTypeOperation.toDishTypeName(dish.getDishTypeId()),

dish.getCode(),

dish.getUnit(),

dish.getPrice(),

dish.getStatus()

});

}

}

}

package gui.panel;

import java.awt.BorderLayout;

import java.util.List;

import javax.swing.JPanel;

import javax.swing.JScrollPane;

import javax.swing.JTable;

import javax.swing.table.DefaultTableModel;

import dao.DishTypeOperation;

import entity.DishType;

import tool.Appearance;

public class DishTypePanel extends JPanel {

private DefaultTableModel model = new DefaultTableModel(new Object[][] {}, new String[] {

"序号", "名称", "描述"

}) {

@Override

public Class<?> getColumnClass(int column) { // 使表格模型能正确识别类型以便排序

Class<?> result = Object.class;

if (getRowCount() > 0) {

Object value = getValueAt(0, column);

if (value != null)

result = value.getClass();

}

return result;

}

@Override

public boolean isCellEditable(int row, int column) { // 使表格不可编辑

return false;

}

};

public JTable table = new JTable(model);

public DishTypePanel() {

setLayout(new BorderLayout());

Appearance.tableInit(table);

List<DishType> list = DishTypeOperation.load();

for (int i = 0; i < list.size(); i++) {

DishType dishType = list.get(i);

model.addRow(new Object[] {

dishType.getId(),

dishType.getName(),

dishType.getDescribe()

});

}

Appearance.fitTableColumns(table);

add(new JScrollPane(table));

}

}

package gui.panel;

import java.awt.BorderLayout;

import java.util.List;

import javax.swing.JPanel;

import javax.swing.JScrollPane;

import javax.swing.JTable;

import javax.swing.table.DefaultTableModel;

import dao.EmployeeOperation;

import entity.Employee;

import tool.Appearance;

public class EmployeePanel extends JPanel {

private DefaultTableModel model = new DefaultTableModel(new Object[][] {}, new String[] {

"序号", "姓名", "性别", "出生日期", "身份证号", "家庭住址", "电话", "职位", "yesno在职"

}) {

@Override

public Class<?> getColumnClass(int column) { // 使表格模型能正确识别类型以便排序

Class<?> result = Object.class;

if (getRowCount() > 0) {

Object value = getValueAt(0, column);

if (value != null)

result = value.getClass();

}

return result;

}

@Override

public boolean isCellEditable(int row, int column) { // 使表格不可编辑

return false;

}

};

public JTable table = new JTable(model);

public EmployeePanel() {

setLayout(new BorderLayout());

Appearance.tableInit(table);

List<Employee> list = EmployeeOperation.load();

for (int i = 0; i < list.size(); i++) {

Employee employee = list.get(i);

model.addRow(new Object[] {

employee.getId(),

employee.getName(),

employee.getSex(),

employee.getBirthday(),

employee.getIdentityId(),

employee.getAddress(),

employee.getTel(),

employee.getPosition(),

employee.getFreeze()

});

}

Appearance.fitTableColumns(table);

Appearance.setCellColor(table, "yesno在职", "yes");

add(new JScrollPane(table));

}

}

package gui.panel;

import java.awt.BorderLayout;

import java.awt.Dimension;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import java.util.List;

import javax.swing.JButton;

import javax.swing.JPanel;

import javax.swing.JScrollPane;

import javax.swing.JTable;

import javax.swing.JTextArea;

import javax.swing.event.ListSelectionEvent;

import javax.swing.event.ListSelectionListener;

import javax.swing.table.DefaultTableModel;

import dao.CustomerOperation;

import dao.OrderOperation;

import dao.TableOperation;

import entity.Order;

import gui.dialog.PayDialog;

import tool.Appearance;

import tool.DataOperation;

public class OrderPanel extends JPanel {

public JButton payButton = new JButton("结账");

private JTextArea infoArea = new JTextArea();

private JPanel borderPanel = new JPanel(new BorderLayout());

private DefaultTableModel model = new DefaultTableModel(new Object[][] {}, new String[] {

"序号", "订单编号", "餐台", "就餐日期时间", "金额", "客户", "状态", "就餐人数"

}) {

@Override

public Class<?> getColumnClass(int column) { // 使表格模型能正确识别类型以便排序

Class<?> result = Object.class;

if (getRowCount() > 0) {

Object value = getValueAt(0, column);

if (value != null)

result = value.getClass();

}

return result;

}

@Override

public boolean isCellEditable(int row, int column) { // 使表格不可编辑

return false;

}

};

public JTable table = new JTable(model);

public OrderPanel() {

setLayout(new BorderLayout());

Appearance.tableInit(table);

table.getSelectionModel().addListSelectionListener(selectAction);

infoArea.setEditable(false);

infoArea.setLineWrap(true); // 让文本域自动换行

infoArea.setWrapStyleWord(true);

payButton.setPreferredSize(new Dimension(128, 32));

payButton.addActionListener(payAction);

refreshOrderTable();

borderPanel.add(payButton, BorderLayout.NORTH);

borderPanel.add(new JScrollPane(infoArea), BorderLayout.CENTER);

add(borderPanel, BorderLayout.EAST);

Appearance.fitTableColumns(table);

Appearance.setCellColor(table, "状态", "paid");

add(new JScrollPane(table), BorderLayout.CENTER);

}

public ListSelectionListener selectAction = new ListSelectionListener() {

@Override

public void valueChanged(ListSelectionEvent e) {

int row = table.getSelectedRow();

if (row != -1) {

payButton.setEnabled(

table.getValueAt(row, 5).toString().equals("not paid"));

infoArea.setText(

OrderOperation.getDetails((int) model.getValueAt(table.convertRowIndexToModel(row), 0)));

} else {

payButton.setEnabled(false);

infoArea.setText("选取一行以查看订单明细信息。");

}

}

};

private ActionListener payAction = new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

new PayDialog(table);

}

};

public void refreshOrderTable() {

List<Order> list = OrderOperation.loadOrder();

model.getDataVector().clear();

for (int i = 0; i < list.size(); i++) {

Order order = list.get(i);

model.addRow(new Object[] {

order.getId(),

order.getOrderNo(),

TableOperation.toTableNo(order.getTableId()),

DataOperation.timestampToString(order.getCreatetime()),

order.getMoney(),

CustomerOperation.toCustomerName(order.getCustomerId()),

order.getStatus(),

order.getNumber()

});

}

}

}

package gui.panel;

import java.awt.BorderLayout;

import java.awt.Dimension;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import java.util.List;

import javax.swing.JButton;

import javax.swing.JOptionPane;

import javax.swing.JPanel;

import javax.swing.JScrollPane;

import javax.swing.JTable;

import javax.swing.JTextArea;

import javax.swing.event.ListSelectionEvent;

import javax.swing.event.ListSelectionListener;

import javax.swing.table.DefaultTableModel;

import dao.TableOperation;

import entity.Table;

import gui.dialog.OrderDialog;

import tool.Appearance;

public class TablePanel extends JPanel {

public JButton orderButton = new JButton("点菜");

private JTextArea infoArea = new JTextArea();

private JPanel borderPanel = new JPanel(new BorderLayout());

private DefaultTableModel model = new DefaultTableModel(new Object[][] {},

new String[] { "序号", "餐台编号", "座位数", "状态" }) {

@Override

public Class<?> getColumnClass(int column) { // 使表格模型能正确识别类型以便排序

Class<?> result = Object.class;

if (getRowCount() > 0) {

Object value = getValueAt(0, column);

if (value != null)

result = value.getClass();

}

return result;

}

@Override

public boolean isCellEditable(int row, int column) { // 使表格不可编辑

return false;

}

};

public JTable table = new JTable(model);

public TablePanel() {

setLayout(new BorderLayout());

Appearance.tableInit(table);

table.getSelectionModel().addListSelectionListener(selectAction);

orderButton.addActionListener(orderAction);

infoArea.setEditable(false);

infoArea.setLineWrap(true); // 让文本域自动换行

infoArea.setWrapStyleWord(true);

orderButton.setPreferredSize(new Dimension(128, 32));

List<Table> list = TableOperation.load();

for (int i = 0; i < list.size(); i++) {

Table table\_ = list.get(i);

model.addRow(new Object[] {

table\_.getId(),

table\_.getNo(),

table\_.getSeating(),

table\_.getStatus()

});

}

Appearance.fitTableColumns(table);

Appearance.setCellColor(table, "状态", "free");

selectAction.valueChanged(null);

borderPanel.add(orderButton, BorderLayout.NORTH);

borderPanel.add(new JScrollPane(infoArea), BorderLayout.CENTER);

add(borderPanel, BorderLayout.EAST);

add(new JScrollPane(table), BorderLayout.CENTER);

}

public ListSelectionListener selectAction = new ListSelectionListener() {

@Override

public void valueChanged(ListSelectionEvent e) {

int row = table.getSelectedRow();

if (row != -1) {

orderButton.setEnabled(table.getValueAt(row, 2).toString().equals("free"));

infoArea.setText(

TableOperation.getDetails((int) model.getValueAt(table.convertRowIndexToModel(row), 0)));

} else {

orderButton.setEnabled(false);

infoArea.setText("选取餐台以查看最近一次在该餐台就餐的客户。");

}

}

};

private ActionListener orderAction = new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

int row = table.convertRowIndexToModel(table.getSelectedRow());

if (TableOperation.change(new Table((int) model.getValueAt(row, 0), (String) model.getValueAt(row, 1),

(int) model.getValueAt(row, 2), "busy"))) {

model.setValueAt("busy", row, 3);

selectAction.valueChanged(null);

new OrderDialog(table);

} else

JOptionPane.showMessageDialog(null, "点菜失败！", "错误", JOptionPane.ERROR\_MESSAGE);

}

};

}

**tool包**

package tool;

import java.awt.Color;

import java.awt.Component;

import java.awt.Container;

import java.awt.Dimension;

import java.awt.Font;

import java.awt.Toolkit;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import java.awt.event.KeyAdapter;

import java.awt.event.KeyEvent;

import java.awt.event.MouseAdapter;

import java.awt.event.MouseEvent;

import java.util.Enumeration;

import javax.swing.JButton;

import javax.swing.JDialog;

import javax.swing.JTable;

import javax.swing.ListSelectionModel;

import javax.swing.UIManager;

import javax.swing.event.ListSelectionEvent;

import javax.swing.event.ListSelectionListener;

import javax.swing.table.DefaultTableCellRenderer;

import javax.swing.table.DefaultTableModel;

import javax.swing.table.JTableHeader;

import javax.swing.table.TableColumn;

import javax.swing.table.TableRowSorter;

import gui.MainFrame;

/\*\*

\* 包含与用户界面相关的方法等。

\*/

public abstract class Appearance {

public static final Font DEFAULT\_FONT = new Font("Microsoft YaHei UI", Font.PLAIN, 16);

public static final Font DEFAULT\_TITLE\_FONT = new Font("Microsoft YaHei UI", Font.BOLD, 12);

public static final Font DEFAULT\_BIG\_FONT = new Font("Microsoft YaHei UI", Font.BOLD, 24);

public static final Font DEFAULT\_HUGE\_FONT = new Font("Microsoft YaHei UI", Font.PLAIN, 32);

/\*\*

\* 依次将控件添加入容器。

\*

\* @param container 容器

\* @param components 控件

\*/

public static void add(Container container, Component... components) {

for (Component c : components) {

container.add(c);

}

}

/\*\*

\* 给表格进行外观初始化及常用监听器的添加。

\*

\* @param table 表格

\*/

public static void tableInit(JTable table) {

table.setSelectionBackground(new Color(204, 236, 255));

table.setSelectionForeground(Color.BLACK);

table.removeColumn(table.getColumnModel().getColumn(0)); // 隐藏表格视图的序号列

table.getTableHeader().setReorderingAllowed(false);

table.setSelectionMode(ListSelectionModel.SINGLE\_SELECTION);

table.setRowHeight(24);

table.setRowSorter(new TableRowSorter<DefaultTableModel>((DefaultTableModel) table.getModel())); // 表格排序器

table.getTableHeader().setFont(DEFAULT\_TITLE\_FONT);

table.getSelectionModel().addListSelectionListener(new ListSelectionListener() {

@Override

public void valueChanged(ListSelectionEvent e) { // 让表格无选中行时，禁用删改按钮

boolean isSelected = table.getSelectedRow() != -1;

MainFrame mainFrame = (MainFrame) table.getRootPane().getParent();

mainFrame.delButton.setEnabled(isSelected);

mainFrame.changeButton.setEnabled(isSelected);

}

});

table.addMouseListener(new MouseAdapter() { // 让表格双击某一行可以直接打开修改对话框

@Override

public void mousePressed(MouseEvent e) {

MainFrame mainFrame = (MainFrame) table.getRootPane().getParent();

if (e.getClickCount() == 2 && table.getSelectedRow() != -1 && mainFrame.operationPanel.isVisible()

&& mainFrame.changeButton.isEnabled()) {

mainFrame.updateAction.actionPerformed(new ActionEvent(e, 0, null));

}

}

});

table.addKeyListener(new KeyAdapter() { // 让表格可以通过回车键和删除键快捷删改某一行

@Override

public void keyPressed(KeyEvent e) {

MainFrame mainFrame = (MainFrame) table.getRootPane().getParent();

if (table.getSelectedRow() != -1 && mainFrame.operationPanel.isVisible()) {

switch (e.getKeyCode()) {

case KeyEvent.VK\_ENTER:

mainFrame.updateAction.actionPerformed(new ActionEvent(e, 0, null));

break;

case KeyEvent.VK\_DELETE:

mainFrame.delAction.actionPerformed(null);

break;

}

}

}

});

}

/\*\*

\* 给对话框内所有控件添加按键监听器以实现键盘快捷操作。

\*

\* @param component 控件

\* @param dialog 对话框

\* @param actionListener 对话框的确认按钮的动作监听器

\*/

private static void addKeyListenerToComponents(Component component, JDialog dialog, ActionListener actionListener) {

component.addKeyListener(new KeyAdapter() {

@Override

public void keyPressed(KeyEvent e) {

switch (e.getKeyCode()) {

case KeyEvent.VK\_ENTER:

actionListener.actionPerformed(null);

break;

case KeyEvent.VK\_ESCAPE:

dialog.dispose();

break;

}

}

});

if (component instanceof Container) {

for (Component c : ((Container) component).getComponents()) {

addKeyListenerToComponents(c, dialog, actionListener);

}

}

}

/\*\*

\* 给对话框添加监听器，以进行快捷操作。

\*

\* @param dialog 对话框

\* @param actionListener 确认按钮的动作监听器

\* @param okButton 确认按钮

\* @param cancelButton 取消按钮

\*/

public static void addListenerToDialog(JDialog dialog, ActionListener actionListener, JButton okButton,

JButton cancelButton) {

addKeyListenerToComponents(dialog, dialog, actionListener);

okButton.addActionListener(actionListener);

cancelButton.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

dialog.dispose();

}

});

}

/\*\*

\* 设置表格一列的前景色为红色或绿色。

\*

\* @param table 表格

\* @param columnName 欲更改前景色的那一列的列名

\* @param trueStr 是该字符串则改为绿色，否则改为红色

\*/

public static void setCellColor(JTable table, String columnName, String trueStr) {

DefaultTableCellRenderer renderer = new DefaultTableCellRenderer() {

@Override

public Component getTableCellRendererComponent(JTable table, Object value, boolean isSelected,

boolean hasFocus, int row, int column) {

Component cell = super.getTableCellRendererComponent(table, value, isSelected, hasFocus, row, column);

if (table.getValueAt(row, column).equals(trueStr)) {

cell.setForeground(new Color(0, 204, 102));

} else {

cell.setForeground(new Color(255, 51, 0));

}

return cell;

}

};

table.getColumn(columnName).setCellRenderer(renderer);

}

/\*\*

\* 隐藏表格的指定列。

\*

\* @param table 表格

\* @param column 要隐藏的一列

\*/

public static void hideIdColumn(JTable table, int column) {

table.removeColumn(table.getColumnModel().getColumn(column));

}

/\*\*

\* 让表格列宽根据内容自动调整

\* @param table

\*/

public static void fitTableColumns(JTable table) {

JTableHeader header = table.getTableHeader();

int rowCount = table.getRowCount();

Enumeration<TableColumn> columns = table.getColumnModel().getColumns();

while (columns.hasMoreElements()) {

TableColumn column = (TableColumn) columns.nextElement();

int col = header.getColumnModel().getColumnIndex(

column.getIdentifier());

int width = (int) table.getTableHeader().getDefaultRenderer()

.getTableCellRendererComponent(table,

column.getIdentifier(), false, false, -1, col)

.getPreferredSize().getWidth();

for (int row = 0; row < rowCount; row++) {

int preferedWidth = (int) table.getCellRenderer(row, col)

.getTableCellRendererComponent(table,

table.getValueAt(row, col), false, false,

row, col)

.getPreferredSize().getWidth();

width = Math.max(width, preferedWidth);

}

header.setResizingColumn(column);

column.setWidth(width \* 2 + table.getIntercellSpacing().width);

}

}

/\*\*

\* 将窗体的主题更改为系统默认主题。

\*/

public static void setToSystemTheme() {

try {

UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());

} catch (Exception e) {

e.printStackTrace();

}

}

/\*\*

\* 将容器及其内包含的所有控件更改为指定字体。

\*

\* @param component 容器

\* @param font 字体

\*/

public static void setAllFont(Component component, Font font) {

component.setFont(font);

if (component instanceof Container) {

for (Component c : ((Container) component).getComponents()) {

setAllFont(c, font);

}

}

}

/\*\*

\* 将窗体或对话框的位置设置为屏幕中央。

\*

\* @param component 窗体或对话框

\*/

public static void setToCenter(Component component) {

Toolkit toolkit = Toolkit.getDefaultToolkit();

Dimension screenSize = toolkit.getScreenSize();

component.setLocation((screenSize.width - component.getWidth()) / 2,

(screenSize.height - component.getHeight()) / 2);

}

}

package tool;

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.SQLException;

import java.sql.Timestamp;

import java.text.DateFormat;

import java.text.SimpleDateFormat;

import java.util.Random;

import javax.swing.JOptionPane;

import javax.swing.JTextField;

/\*\*

\* 包含数据转换、数据库连接等各种数据操作方法。

\*/

public abstract class DataOperation {

public static Connection connection;

/\*\*

\* 清除文本域首尾的空格。

\*

\* @param textFields 文本域对象

\*/

public static void trimTextField(JTextField... textFields) {

for (JTextField textField : textFields) {

textField.setText(textField.getText().trim());

}

}

/\*\*

\* 将异常以错误信息框的形式显示出来。

\*

\* @param e 异常

\*/

public static void showException(Exception e) {

JOptionPane.showMessageDialog(null, e.getMessage(), "Exception", JOptionPane.ERROR\_MESSAGE);

e.printStackTrace();

}

/\*\*

\* 将字符串转换为SQL日期。

\*

\* @param str 字符串

\* @return SQL日期

\*/

public static java.sql.Date stringToDate(String str) {

DateFormat df = new SimpleDateFormat("yyyy-M-d");

java.util.Date date = null;

try {

date = df.parse(str);

} catch (Exception e) {

}

if (date == null)

return null;

return new java.sql.Date(date.getTime());

}

/\*\*

\* 将SQL日期转换为字符串数组，分别为年月日。

\*

\* @param date SQL日期

\* @return 日期字符串数组，若SQL日期为null，则为空字符串

\*/

public static String[] dateToString(java.sql.Date date) {

if (date == null)

return new String[] { "", "", "" };

java.util.Date utilDate = new java.util.Date(date.getTime());

return new String[] { new SimpleDateFormat("yyyy").format(utilDate),

new SimpleDateFormat("M").format(utilDate),

new SimpleDateFormat("d").format(utilDate) };

}

/\*\*

\* 将SQL时间戳转换为yyyy-MM-dd HH:mm:ss格式的字符串

\*

\* @param timestamp SQL时间戳

\* @return 日期字符串数组，若SQL日期为null，则为空字符串

\*/

public static String timestampToString(Timestamp timestamp) {

return new SimpleDateFormat("yyyy-MM-dd HH:mm:ss").format(new java.util.Date(timestamp.getTime()));

}

/\*\*

\* 生成订单编号

\*

\* @return 订单编号

\*/

public static String generateOrderNo() {

SimpleDateFormat df = new SimpleDateFormat("yyyyMMddHHmmss");

String timeStr = df.format(new java.util.Date());

Random random = new Random();

return timeStr + random.nextInt(10) + random.nextInt(10) + random.nextInt(10) + random.nextInt(10);

}

/\*\*

\* 获取SQL数据库连接。

\*

\* @param username SQL用户名

\* @param password SQL密码

\* @return SQL连接

\* @throws SQLException SQL异常

\*/

public static Connection getConnection(String username, String password) throws SQLException {

try {

Class.forName("com.mysql.cj.jdbc.Driver");

} catch (ClassNotFoundException e) {

e.printStackTrace();

}

connection = DriverManager.getConnection("jdbc:mysql://localhost:3306/restaurant\_management", username, password);

return connection;

}

}