# 華中科技大學

# 课 程 实 验 报 告

**课程名称：** Java 语言程序设计

**实验名称：** 医院简易挂号管理系统

院　　系：计算机科学与技术
专业班级：IOT1601
学　　号：U201614897
姓　　名：潘　越
指导教师：马光志

2019 年 4 月 21 日

# 目 录

# 一、需求分析

## 1.题目要求

采用桌面应用程序模式，开发一个医院挂号系统，管理包括人员、号种及其挂号费用，挂号退号等信息，完成登录、挂号、查询和统计打印功能。数据库表如下所示，建立索引的目的是加速访问，请自行确定每个索引要涉及哪些字段。

表 1.1 T_KSXX (科室信息表)

| 字段名称 | 字段类型 | 主键 | 索引 | 可空 | 备注 |
|---|---|---|---|---|---|
| KSBH | CHAR(6) | 是 | 是 | 否 | 科室编号，数字 |
| KSMC | CHAR(10) | 否 | 否 | 否 | 科室名称 |
| PYZS | CHAR(8) | 否 | 否 | 否 | 科室名称的拼音字首 |

表 1.2 T_BRXX (病人信息表)

| 字段名称 | 字段类型 | 主键 | 索引 | 可空 | 备注 |
|---|---|---|---|---|---|
| BRBH | CHAR(6) | 是 | 是 | 否 | 病人编号，数字 |
| BRMC | CHAR(10) | 否 | 否 | 否 | 病人名称 |
| DLKL | CHAR(8) | 否 | 否 | 否 | 登录口令 |
| YCJE | DECIMAL(10,2) | 否 | 否 | 否 | 病人预存金额 |
| DLRQ | DateTime | 否 | 否 | 是 | 最后一次登录日期及时间 |

表 1.3 T_KSYS (科室医生表)

| 字段名称 | 字段类型 | 主键 | 索引 | 可空 | 备注 |
|---|---|---|---|---|---|
| YSBH | CHAR(6) | 是 | 是 | 否 | 医生编号，数字，第 1 索引 |
| KSBH | CHAR(6) | 否 | 是 | 否 | 所属科室编号，第 2 索引 |
| YSMC | CHAR(10) | 否 | 否 | 否 | 医生名称 |
| PYZS | CHAR(4) | 否 | 否 | 否 | 医生名称的拼音字首 |
| DLKL | CHAR(8) | 否 | 否 | 否 | 登录口令 |
| SFZJ | BOOL | 否 | 否 | 否 | 是否专家 |
| DLRQ | DATETIME | 否 | 否 | 是 | 最后一次登录日期及时间 |

表 1.4 T_HZXX (号种信息表)

| 字段名称 | 字段类型 | 主键 | 索引 | 可空 | 备注 |
|---|---|---|---|---|---|
| HZBH | CHAR(6) | 是 | 是 | 否 | 号种编号，数字，第 1 索引 |
| HZMC | CHAR(12) | 否 | 否 | 否 | 号种名称 |
| PYZS | CHAR(4) | 否 | 否 | 否 | 号种名称的拼音字首 |
| KSBH | CHAR(6) | 否 | 是 | 否 | 号种所属科室，第 2 索引 |
| SFZJ | BOOL | 否 | 否 | 否 | 是否专家号 |
| GHRS | INT | 否 | 否 | 否 | 每日限定的挂号人数 |
| GHFY | DECIMAL(8,2) | 否 | 否 | 否 | 挂号费 |

表 1.5 T_GHXX (挂号信息表)

| 字段名称 | 字段类型 | 主键 | 索引 | 可空 | 备注 |
|---|---|---|---|---|---|
| GHBH | CHAR(6) | 是 | 是 | 否 | 挂号的顺序编号，数字 |
| HZBH | CHAR(6) | 否 | 是 | 否 | 号种编号 |
| YSBH | CHAR(6) | 否 | 是 | 否 | 医生编号 |
| BRBH | CHAR(6) | 否 | 是 | 否 | 病人编号 |
| GHRC | INT | 否 | 是 | 否 | 该病人该号种的挂号人次 |
| THBZ | BOOL | 否 | 否 | 否 | 退号标志=true 为已退号码 |
| GHFY | DECIMAL(8,2) | 否 | 否 | 否 | 病人的实际挂号费用 |
| RQSJ | DATETIME | 否 | 否 | 否 | 挂号日期时间 |

为了减少编程工作量，T_KSXX、T_BRXX、T_KSYS、T_HZXX 的信息手工录入数据库，每个表至少录入 6 条记录，所有类型为 CHAR(6)的字段数据从“000001”开始，连续编码且中间不得空缺。为病人开发的桌面应用程序要实现的主要功能具体如下：

（1）病人登录：输入自己的病人编号和密码，经验证无误后登录。

（2）病人挂号：病人处于登录状态，选择科室、号种和医生（非专家医生不得挂专家号，专家医生可以挂普通号）；输入缴费金额，计算并显示找零金额后完成挂号。所得挂号的编号从系统竞争获得生成，挂号的顺序编号连续编码不得空缺。

功能（2）的界面如下所示，在光标停在“科室名称”输入栏时，可在输入栏下方弹出下拉列表框，显示所有科室的“科室编号”、“科室名称”和“拼音字首”，此时可通过鼠标点击或输入科室名称的拼音字首两种输入方式获得“科室编号”，用于插入 T_GHXX 表。注意，采用拼音字首输入时可同时完成下拉

列表框的科室过滤，使得下拉列表框中符合条件的科室越来越少，例如，初始为“内一科”和“内二课”。其它输入栏，如“医生姓名”、“号种类别”、“号种名称”也可同时支持两种方式混合输入。

每种号种挂号限定当日人次，挂号人数超过规定数量不得挂号。一个数据一致的程序要保证：挂号总人数等于当日各号种的挂号人次之和，病人的账务应保证开支平衡。已退号码不得用于重新挂号，每个号重的 GHRC 数据应连续不间断，GHRC 从 1 开始。若病人有预存金额则直接扣除挂号费，此时“交款金额”和“找零金额”处于灰色不可操作状态。


图 1.1 门诊挂号界面（示例）

为医生开发的桌面应用程序要实现的主要功能具体如下：

（1）医生登录：输入自己的医生编号和密码，经验证无误后登录。

（2）病人列表：医生处于登录状态，显示自己的挂号病人列表，按照挂号编号升序排列。显示结果如下表所示。

表 1.6 病人列表 (示例)

| 挂号编号 | 病人名称 | 挂号日期时间 | 号种类别 |
|---|---|---|---|
| 000001 | 章紫衣 | 2018-12-30 11:52:26 | 专家号 |
| 000003 | 范冰冰 | 2018-12-30 11:53:26 | 普通号 |
| 000004 | 刘德华 | 2018-12-30 11:54:28 | 普通号 |

（3）收入列表：医生处于登录状态，显示所有科室不同医生不同号种起止日期内的收入合计，起始日期不输入时默认为当天零时开始，截止日期至当

前时间为止。时间输入和显示结果如下表所示。

表 1.7 收入列表 (示例)

起始时间：2018-12-30 00:00:00 截止时间：2018-12-30 12:20:00

| 科室名称 | 医生编号 | 医生名称 | 号种类别 | 挂号人次 | 收入合计 |
|---|---|---|---|---|---|
| 感染科 | 000001 | 李时珍 | 专家号 | 24 | 48 |
| 感染科 | 000001 | 李时珍 | 普通号 | 10 | 10 |
| 内一科 | 000002 | 扁鹊 | 普通号 | 23 | 23 |
| 保健科 | 000003 | 华佗 | 专家号 | 10 | 20 |

病人应用程序和医生应用程序可采用主窗口加菜单的方式实现。例如，医生应用程序有三个菜单项，分别为“病人列表”、“收入列表”和“退出系统”等。

考虑到客户端应用程序要在多台计算机上运行，而这些机器的时间各不相同，客户端程序每次在启动时需要同数据库服务器校准时间，可以建立一个时间服务程序或者直接取数据库时间校准。建议大家使用 MS SQL 数据库开发。

挂号时锁定票号可能导致死锁，为了防止死锁或系统响应变慢，建议大家不要锁死数据库表或者字段。程序编写完成后，同时启动两个挂号程序进行单步调试，以便测试两个病人是否会抢到同一个号、或者有号码不连续或丢号的现象。

系统考核目标：

（1）挂号后数据库数据包括挂号时间不会出现不一致或时序颠倒现象，以及挂号人次超过该号种当日限定数量的问题；

（2）挂号号码和挂号人次不会出现不连续或丢号问题；

（3）病人的开支应平衡，并应和医院的收入平衡；

（4）系统界面友好、操作简洁，能支持全键盘操作、全鼠标操作或者混合操作；

（5）能支持下拉列表框过滤输入；

（6）系统响应迅速，不会出现死锁；

（7）统计报表应尽可能不采用多重或者多个循环实现；

（8）若采用时间服务器程序校准时间，最好能采用心跳检测机制，显示客户端的上线和下线情况。

思考题：当病人晚上 11:59:59 秒取得某号种的挂号价格 10 元，当他确定保存时价格在第 2 天 00:00:00 已被调整为 20 元，在编程时如何保证挂号费用与当天价格相符？

## 2.需求分析

首先分析系统组成，整个系统分为五大部分，其架构如下图所示：


图 1.2 系统架构图

整个系统分为三大模块，MySQL 数据库用来存储，数据库操作层封装 SQL 语句，提供 API 供上层系统操作数据库，GUI 界面供用户操作，提供各种数据输入以及操作按钮。

GUI 界面又分为三个模块，登录界面提供医生登录和病人登录两种模式。病人界面提供病人挂号操作，医生界面提供查询挂号以及查询收入操作。在每个界面需要适当提供一些便于用户操作的功能，例如筛选，自动计算等功能。、

此外，考虑到程序的编写，额外提供了一个 Main 模块作为程序的入口，执行加载数据库驱动、连接数据库、启动登录界面的功能。

# 二、系统设计

## 1.概要设计

系统的整体框架前文中提到过，如图 2.1 所示，整个系统的业务流程图如图 2.1 所示，

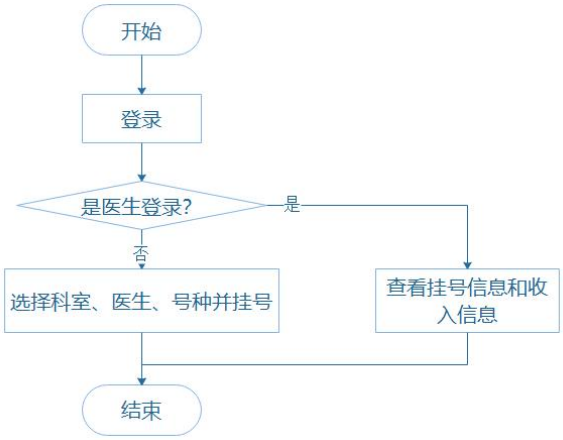


图 2.1 系统业务流程图

整个系统的模式比较简单，接下来描述五个主要部件的设计

## 2.详细设计

### （1）数据库设计

数据库的设计按照表 1.1 至表 1.5 来实现，通过手动添加的方式，在 MySQL 数据库中创建一个用户 zxcpyp 作为挂号系统在 MySQL 中的用户，创建数据库 java_lab2 作为挂号系统的数据库，采用硬写入的方式，创建五个表并在科室表中插入 6 个科室；在医生表中插入 12 个医生，每个科室一位专家一位普通医生；在病人表里插入 6 个病人；在号种表中插入 24 个号种，每个科室 4 个号，两个专家号两个普通号。

上述操作可以先在 MySQL 中建好表，然后通过导出数据库的方式导出数据到 database.sql 文件中，然后对文件进行修改即可。

### （2）数据库操作层设计

数据库操作层的作用主要是封装 SQL 语句，为上层提供 API 接口。它是系统的基础部分，必须在 GUI 界面被初始化之前初始化。

这里使用一个类 DBConnecto 来封装实现，其基本设计如下（未列出异常处理部分）：

```
public class DBConnector {
        private static DBConnector instance = null;
        private Connection connection;
        private Connection transactionConnection;
        private Statement statement;
        private Statement transactionStatement;

        static public DBConnector getInstance() {
                // TODO
        }

        public void connectDataBase(
                // TODO
        }

        public ResultSet getWholeTable(String tableName) {
                // TODO
                }
        }
        // TODO
```

```
}
```

DBConnecto 类的基本框架如上，通过单例模式来保证在全局数据连接类只有一个实例，这样可以简化上层的逻辑，便于高层代码编写。

上面提供了三个接口示例，getInstance()用于获取实例，connectDataBase()用于连接数据库，在初始化时调用，getWholeTable()为 SQL 语句封装示例，作用是获取所有的表内容，其他的 SQL 语句也是类似，将不同的查询功能封装成不同的方法函数，这里不一一列举了。

**（3）登录界面设计**

登录界面如下图所示：


图 2.2 登录界面

登录界面的设计很简单，需要一个帐号输入栏，一个密码输入栏，三个功能按钮，分别是“医生登录”、“病人登录”、“退出”以及一个错误显示栏。界面采用 JFX 和 Sense Builder 工具来图形化的设计。

登录功能的流程图如下：

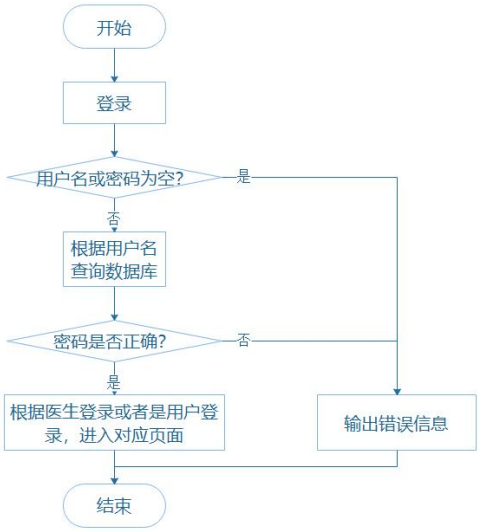


图 2.3 登录功能流程图

**（4）病人界面设计**

病人登录后的界面如下：


图 2.4 病人界面

在该界面中，右侧的两个输入栏是必要的，分别是挂号的医生姓名和挂的号种名称，当这两个栏中均有合法信息时，就可以执行挂号操作。

左侧的两个输入栏起筛选作用，筛选科室和号种类别，筛选的结果会直接反应到右侧两个输入栏选择时的可选项。这一功能通过嵌套 SQL 查询来实现。

付款功能有两个选项，使用余额付款或者现金支付。若使用余额付款，则当余额够用时，挂号按钮激活，点击之后完成挂号并自动扣除余额，并在下方显示挂号编号。若选择现金支付，则当输入现金不足时，下面会显示金额不足。在输入金额框中输入金额，找零金额会随着输入的变化自动显示，这里可以设置触发

时间为按下键盘之后。此外，勾选找零存入余额之后，使用现金付款后的找零会存入余额中。

**（5）医生界面设计**

医生登录后，可以选择两个功能“挂号列表”和“收入列表”，选择挂号列表后的界面如图 2.5 所示。

下方设置“开始时间”、“结束时间”、“全部时间”、“今天”四个筛选栏供筛选查询结果。整个实现基础为一个 SELECT 语句，根据这些条件来从数据库中查找结果并显示在列表中。


图 2.5 挂号列表界面

选择“收入列表”则进入如图 2.6 所示的界面，和“挂号列表”相同，也是通过筛选条件来执行查找，并将查找的结果显示在列表中，不再赘述。


图 2.6 收入列表界面

## 三、软件开发

本实验的开发与测试的环境如下：

（1）操作系统：Arch Linux x86_64 kernel version 5.0.7

（2）JRE：jre8-openjdk 8.u212-1

（3）JDK：jdk8-openjdk 8.u212-1

（4）IDE：IntelliJ IDEA 2019.1

（5）图形化 GUI 编辑工具：Sense Builde

（6）数据库：MariaDB version 10.3.14-1

实验程序的编译与调试均在 IDEA 中完成。

## 四、软件测试

首先打开界面，输入帐号和密码。在数据库中我规定了所有医生的密码为“123456”，所有病人的密码为“654321”，点击“病人登录”，可以进入病人界面。（若此处帐号和密码有一个没有输入，则会提示“请输入密码”，若用户名不存在，则会提示“用户名不存在”，若用户名存在但密码错误，则会提示“密码错误”）。


图 4.1 登录

接着如图 4.2 和 4.3 所示，对用户 001 分别使用“余额”挂号和使用“现金”挂号操作。


图 4.2 使用余额挂号


图 4.3 使用现金挂号

接着换医生 001 登录（即病人 001 挂号的医生），点击查询可以看到挂号列表为刚刚我们挂的两个号。点击“收入列表”也可以看到统计的收入，如图 4.4 和 4.5 所示。


图 4.4 查看挂号列表


图 4.5 查看收入列表

基本功能完善，接着测试挂号上限，登录用户 005，按照如图 4.6 所示的挂号一直挂号。当达到上限后，如图 4.7 所示，可以看到系统已经不允许挂号了。


图 4.6 一直挂号


图 4.7 挂号达到上限

# 五、特点与不足

## 1.技术特点

（1）将所有的数据库操作封装为了一个单例类，便于上层程序操作。

（2）筛选操作是互相影响的，比如病人界面的四个输入栏，里面的数据会对其他三个的可选项进行筛选，提高了效率。

（3）使用 Sense Builder 进行可视化 GUI 开发，高效便捷，细节之处再在 xml 文件上做修改，在全局使用 main.css 控制样式。

（4）程序打包之后可以在全平台运行，只要有 java 8 以上环境即可。

## 2.不足和改进的建议

（1）密码明文存储，安全性较低，可以考虑使用 md5 等手段加密。

（2）限于时间原因，实现的功能很简单，只有基本的挂号与查询功能，还可以考虑实现统计与输出报表等功能。

# 六、过程和体会

## 1.遇到的问题和主要解决方法

本次实验中遇到的主要问题就是 GUI 的编写了，一直以来我是最讨厌图形界面编程的，总是有着各种各样的麻烦。这次实验中，即使是用上了 Sense Builder 这样的图形化编辑工具，还是有着各种各样的麻烦，比如位置和大小难以控制，颜色更改无效，各种事件的触发，要调整好确实很难。最终经过几天的研究，还是努力去习惯使用 Sense Builder + XML + CSS 的开发方式，使用 Sense Builder

初步编辑框架，编辑 XML 来添加事件触发，修改简单的 style 等等，编辑 CSS 来设置一些全局的样式等，最后在别人提供的框架基础上，改造出了自己的系统。

### 2.课程设计的体会

由于以前学过了数据库原理，并且也写过了数据库的课程设计，所以在做本次实验时对数据库就很熟悉了，只不过当时是使用 HTML + PHP + JS 的方式来开发的，这次主要就是学习 Java 了。通过本次实验，我最大的收获是学习了使用 JavaFX 来搭建一个友好的用户界面，让 Java 来完成各种复杂的功能，不仅进一步熟悉了 Java 的语法及特性，巩固了上课学过的知识，还提升了我的设计能力与工程能力，对今后的学习生活也是一个很好的经验。

## 七、源码和说明

### 1.文件清单及其功能说明

本实验的文件清单如下图：

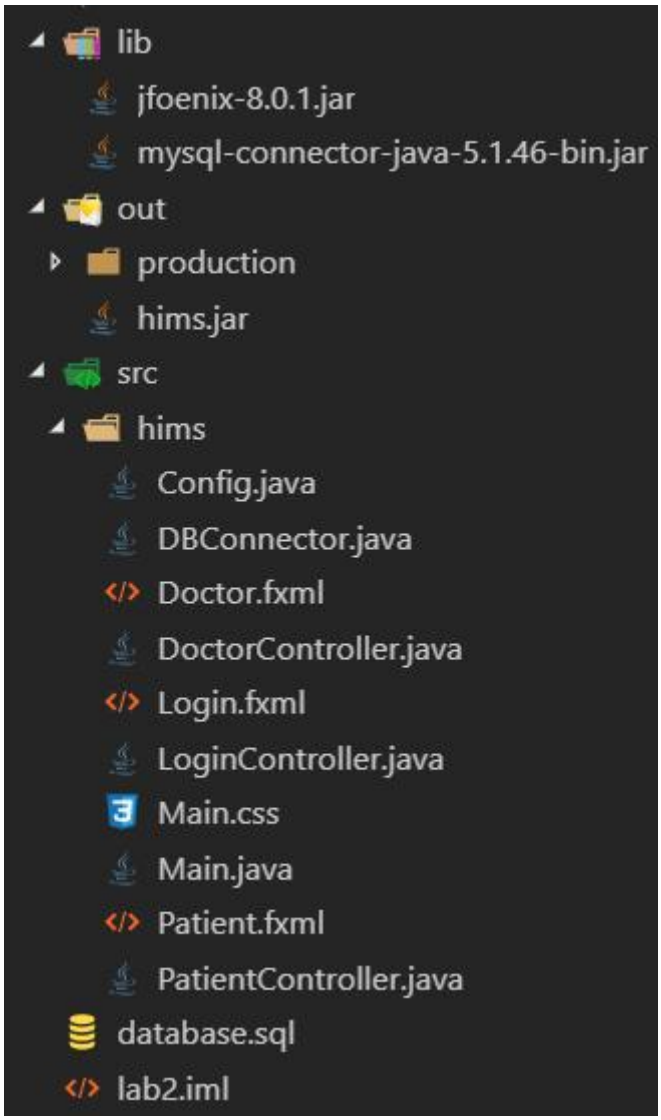

图 7.1 代码清单

其中 src 目录下为程序源码，功能分别如下：
Config.java：数据库表及项名称配置
DBConnection.java：数据库连接层
DoctorController.java：医生界面功能控制
PatientController.java：病人界面功能控制
LoginController.java：登录界面功能控制
Main.java：程序入口
Doctor.xml：医生界面 XML 文件
Patient.xml：病人界面 XML 文件
Login.xml：登录界面 XML 文件
Main.css：全局样式文件

out 目录下为输出的程序，hims.jar 为打包好的程序 jar 文件

lib 目录下为依赖库
jfoenix-8.0.1.jar：JFoenix 第三方库
mysql-connector-java-5.1.46-bin.jar：MySQL 第三方库

database.sql：初始化数据库文件

## 2.用户使用说明书

（1）创建数据库
create database java_lab2；
（2）创建用户，专门操作 java_lab2 数据库，并且赋予其所有权限
create user zxcpyp;
grant all on java_lab2.* to 'zxcpyp'@"localhost' identified by '123456';
（3）初始化数据
use java_lab2;
source database.sql;
之后就可以正常运行 hims.jar 程序了，使用
Java -jar hims.jar
即可启动医院挂号管理系统。

## 3.源代码

Main.java

```
package hims;

import javafx.application.Application;
import javafx.fxml.FXMLLoader;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.stage.Stage;

import java.sql.SQLException;

public class Main extends Application {
	@Override
	public void start(Stage primaryStage) throws Exception {
		// Connect to database
		try {
			DBConnector.getInstance().connectDataBase("localhost",
3306, "java_lab2", "zxcpyp", "123456");
		} catch (SQLException e) {
			System.err.println("failed to connect to sql database");
			System.exit(0);
		}
		// Start JFX
		Parent root = FXMLLoader.load(getClass().getResource("Login.fxml"));
		primaryStage.setTitle("医院简易挂号管理系统");
		primaryStage.setScene(new Scene(root));
		primaryStage.show();
	}

	public static void main(String[] args) {
		launch(args);
	}
}
```

LoginController.java

```
package hims;

import com.jfoenix.controls.*;
import com.sun.javafx.robot.impl.FXRobotHelper;
import javafx.fxml.FXML;
import javafx.fxml.FXMLLoader;
import javafx.scene.Scene;
import javafx.scene.control.*;
import javafx.scene.input.KeyCode;

import java.io.IOException;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;

import static java.lang.System.exit;

public class LoginController {
	@FXML
	JFXTextField inputUsername;
```

```
	@FXML
	JFXPasswordField inputPassword;
	@FXML
	JFXButton buttonLoginDoctor;
	@FXML
	JFXButton buttonLoginPatient;
	@FXML
	JFXButton buttonExit;
	@FXML
	Label labelStatus;

	@FXML
	void initialize() {
		buttonLoginDoctor.setOnKeyReleased(keyEvent -> {
			try {
				if (keyEvent.getCode() == KeyCode.ENTER)
					doctorLogin();
			} catch (IOException e) {
			}
		});

		buttonLoginPatient.setOnKeyReleased(keyEvent -> {
			try {
				if (keyEvent.getCode() == KeyCode.ENTER)
					patientLogin();
			} catch (IOException e) {
			}
		});

		buttonExit.setOnKeyReleased(keyEvent -> {
			if (keyEvent.getCode() == KeyCode.ENTER)
				exit(0);
		});
	}

	@FXML
	void doctorLogin() throws IOException {
		if (!validateUserNameAndPassword())
			return;

		ResultSet result =
DBConnector.getInstance().getDoctorInfo(inputUsername.getText().trim());
		if (result == null) {
			labelStatus.setText("读取数据库错误，请联系管理员。");
			labelStatus.setStyle("-fx-text-fill: red;");
		}

		try {
			if (!result.next()) {
				labelStatus.setText("用户不存在");
				labelStatus.setStyle("-fx-text-fill: red;");
				return;
			} else if
(!result.getString(Config.NameTableColumnDoctorPassword).equals(inputPassword.getText()
)) {
				labelStatus.setText("密码错误");
				labelStatus.setStyle("-fx-text-fill: red;");
```

```
                                        return;
                                }

                                DoctorController.doctorName =
result.getString(Config.NameTableColumnDoctorName);
                                DoctorController.doctorNumber =
result.getString(Config.NameTableColumnDoctorNumber);

                                DBConnector.getInstance().updateDoctorLoginTime(

                result.getString(Config.NameTableColumnDoctorNumber),

                LocalDateTime.now().format(DateTimeFormatter.ofPattern("yyyy-MM-dd
HH:mm:ss")));

                                Scene scene = new
Scene(FXMLLoader.load(getClass().getResource("Doctor.fxml")));

                scene.getStylesheets().add(getClass().getResource("Main.css").toExternalForm());
                                FXRobotHelper.getStages().get(0).setScene(scene);
                        } catch (SQLException e) {
                                e.printStackTrace();
                                return;
                        }
                }

                @FXML
                void patientLogin() throws IOException {
                        if (!validateUserNameAndPassword())
                                return;

                        ResultSet result =
DBConnector.getInstance().getPatientInfo(inputUsername.getText().trim());
                        if (result == null) {
                                labelStatus.setText("读取数据库错误，请联系管理员。");
                                labelStatus.setStyle("-fx-text-fill: red;");
                        }

                        try {
                                if (!result.next()) {
                                        labelStatus.setText("用户不存在");
                                        labelStatus.setStyle("-fx-text-fill: red;");
                                        return;
                                } else if
(!result.getString(Config.NameTableColumnPatientPassword).equals(inputPassword.getText()
)) {
                                        labelStatus.setText("密码错误");
                                        labelStatus.setStyle("-fx-text-fill: red;");
                                        return;
                                }
                                PatientController.patientName =
result.getString(Config.NameTableColumnPatientName);
                                PatientController.patientBalance =
result.getDouble(Config.NameTableColumnPatientBalance);
                                PatientController.patientNumber =
result.getString(Config.NameTableColumnPatientNumber);
```

```
                                DBConnector.getInstance().updatePatientLoginTime(

            result.getString(Config.NameTableColumnPatientNumber),

            LocalDateTime.now().format(DateTimeFormatter.ofPattern("yyyy-MM-dd
HH:mm:ss")));

                                Scene scene = new
Scene(FXMLLoader.load(getClass().getResource("Patient.fxml")));

            scene.getStylesheets().add(getClass().getResource("Main.css").toExternalForm());
                                FXRobotHelper.getStages().get(0).setScene(scene);
                        } catch (SQLException e) {
                                e.printStackTrace();
                                return;
                        }
                }

                private boolean validateUserNameAndPassword() {
                        if (inputUsername.getText().isEmpty()) {
                                inputUsername.setStyle("-fx-background-color: pink;");
                                labelStatus.setText("请输入用户名");
                                labelStatus.setStyle("-fx-text-fill: red;");
                                return false;
                        }
                        if (inputPassword.getText().isEmpty()) {
                                inputPassword.setStyle("-fx-background-color: pink;");
                                labelStatus.setText("请输入密码");
                                labelStatus.setStyle("-fx-text-fill: red;");
                                return false;
                        }

                        labelStatus.setText("登录中...");
                        labelStatus.setStyle("");
                        return true;
                }

                @FXML
                void onInputUsernameAction() {
                        inputUsername.setStyle("");
                }

                @FXML
                void onInputPasswordAction() {
                        inputPassword.setStyle("");
                }

                @FXML
                void buttonExitClicked() {
                        exit(0);
                }
}
```

DoctorController.java

```
package hims;
```

```
import com.jfoenix.controls.*;
import com.jfoenix.controls.datamodels.treetable.RecursiveTreeObject;
import com.sun.javafx.robot.impl.FXRobotHelper;
import javafx.beans.property.SimpleStringProperty;
import javafx.beans.property.StringProperty;
import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.fxml.FXML;
import javafx.fxml.FXMLLoader;
import javafx.scene.Scene;
import javafx.scene.control.*;
import javafx.util.StringConverter;
import javafx.event.*;

import java.io.IOException;
import java.sql.*;

import java.time.LocalDate;
import java.time.LocalDateTime;
import java.time.LocalTime;
import java.time.format.DateTimeFormatter;


public class DoctorController {
        private static final class Register extends RecursiveTreeObject<Register> {
                public StringProperty number;
                public StringProperty namePatient;
                public StringProperty dateTimeDisplay;
                public StringProperty isSpecialistDisplay;

                public Register(String number, String namePatient, Timestamp dateTime,
boolean isSpecialist) {
                        this.number = new SimpleStringProperty(number);
                        this.namePatient = new SimpleStringProperty(namePatient);
                        this.dateTimeDisplay = new
SimpleStringProperty(dateTime.toLocalDateTime().format(DateTimeFormatter.ofPattern("yy
yy-MM-dd HH:mm:ss")));
                        this.isSpecialistDisplay = new
SimpleStringProperty(isSpecialist ? "专家号" : "普通号");
                }
        }

        private static final class Income extends RecursiveTreeObject<Income> {
                public StringProperty departmentName;
                public StringProperty doctorNumber;
                public StringProperty doctorName;
                public StringProperty registerType;
                public StringProperty registerPopulation;
                public StringProperty incomeSum;

                public Income(String depName, String docNum, String docName,
boolean isSpec, int regNumPeople, Double incomSum) {
                        this.departmentName = new SimpleStringProperty(depName);
                        this.doctorNumber = new SimpleStringProperty(docNum);
                        this.doctorName = new SimpleStringProperty(docName);
```

```
                    this.registerType = new SimpleStringProperty(isSpec ? "专家号" : "普通号");
                    this.registerPopulation = new SimpleStringProperty(Integer.toString(regNumPeople));
                    this.incomeSum = new SimpleStringProperty(String.format("%.2f", incomSum));
                }
            }

            public static String doctorName;
            public static String doctorNumber;

            @FXML
            private Label labelWelcome;
            @FXML
            private JFXDatePicker pickerDateStart;
            @FXML
            private JFXDatePicker pickerDateEnd;
            @FXML
            private JFXTimePicker pickerTimeStart;
            @FXML
            private JFXTimePicker pickerTimeEnd;

            @FXML
            private JFXTabPane mainPane;
            @FXML
            private Tab tabRegister;
            @FXML
            private Tab tabIncome;

            @FXML
            private JFXTreeTableView<Register> tableRegister;
            @FXML
            private TreeTableColumn<Register, String> columnRegisterNumber;
            @FXML
            private TreeTableColumn<Register, String> columnRegisterPatientName;
            @FXML
            private TreeTableColumn<Register, String> columnRegisterDateTime;
            @FXML
            private TreeTableColumn<Register, String> columnRegisterType;
            private TreeItem<Register> rootRegister;

            @FXML
            private JFXTreeTableView<Income> tableIncome;
            @FXML
            private TreeTableColumn<Income, String> columnIncomeDepartmentName;
            @FXML
            private TreeTableColumn<Income, String> columnIncomeDoctorNumber;
            @FXML
            private TreeTableColumn<Income, String> columnIncomeDoctorName;
            @FXML
            private TreeTableColumn<Income, String> columnIncomeRegisterType;
            @FXML
            private TreeTableColumn<Income, String> columnIncomeRegisterPopulation;
            @FXML
            private TreeTableColumn<Income, String> columnIncomeSum;
            private TreeItem<Income> rootIncome;
```

```
        private ObservableList<Register> listRegister =
FXCollections.observableArrayList();
        private ObservableList<Income> listIncome =
FXCollections.observableArrayList();

        @FXML
        JFXCheckBox checkBoxAllTime;
        @FXML
        JFXCheckBox checkBoxToday;
        @FXML
        JFXButton buttonFilter;

        @FXML
        void initialize() {
                labelWelcome.setText(String.format("欢迎进入医院挂号系统，%s！",
doctorName));

                pickerDateStart.setConverter(new DateConverter());
                pickerDateEnd.setConverter(new DateConverter());
                pickerDateStart.setValue(LocalDate.now());
                pickerDateEnd.setValue(LocalDate.now());
                pickerTimeStart.setIs24HourView(true);
                pickerTimeEnd.setIs24HourView(true);
                pickerTimeStart.setValue(LocalTime.MIN);
                pickerTimeEnd.setValue(LocalTime.MAX);

                columnRegisterNumber.setCellValueFactory(
                                (TreeTableColumn.CellDataFeatures<Register,
String> param) -> param.getValue().getValue().number);
                columnRegisterPatientName.setCellValueFactory(
                                (TreeTableColumn.CellDataFeatures<Register,
String> param) -> param.getValue().getValue().namePatient);
                columnRegisterDateTime.setCellValueFactory(
                                (TreeTableColumn.CellDataFeatures<Register,
String> param) -> param.getValue().getValue().dateTimeDisplay);
                columnRegisterType.setCellValueFactory(
                                (TreeTableColumn.CellDataFeatures<Register,
String> param) -> param.getValue().getValue().isSpecialistDisplay);

                rootRegister = new RecursiveTreeItem<>(listRegister,
RecursiveTreeObject::getChildren);
                tableRegister.setRoot(rootRegister);

                columnIncomeDepartmentName.setCellValueFactory(
                                (TreeTableColumn.CellDataFeatures<Income,
String> param) -> param.getValue().getValue().departmentName);
                columnIncomeDoctorNumber.setCellValueFactory(
                                (TreeTableColumn.CellDataFeatures<Income,
String> param) -> param.getValue().getValue().doctorNumber);
                columnIncomeDoctorName.setCellValueFactory(
                                (TreeTableColumn.CellDataFeatures<Income,
String> param) -> param.getValue().getValue().doctorName);
                columnIncomeRegisterType.setCellValueFactory(
                                (TreeTableColumn.CellDataFeatures<Income,
String> param) -> param.getValue().getValue().registerType);
                columnIncomeRegisterPopulation.setCellValueFactory(
```

```
                                        (TreeTableColumn.CellDataFeatures<Income,
String> param) -> param.getValue().getValue().registerPopulation);
                        columnIncomeSum.setCellValueFactory(
                                        (TreeTableColumn.CellDataFeatures<Income,
String> param) -> param.getValue().getValue().incomeSum);

                        rootIncome = new RecursiveTreeItem<>(listIncome,
RecursiveTreeObject::getChildren);
                        tableIncome.setRoot(rootIncome);

                }

                @FXML
                private void buttomFilterPressed() {
                        if (mainPane.getSelectionModel().getSelectedItem() == tabRegister) {
                                ResultSet result;
                                if (checkBoxAllTime.isSelected()) {
                                        result =
DBConnector.getInstance().getRegisterForDoctor(
                                                        doctorNumber,
                                                        "0000-00-00 00:00:00",

                LocalDateTime.now().format(DateTimeFormatter.ofPattern("yyyy-MM-dd
HH:mm:ss"))
                                        );
                                } else if (checkBoxToday.isSelected()) {
                                        result =
DBConnector.getInstance().getRegisterForDoctor(
                                                        doctorNumber,

                LocalDate.now().format(DateTimeFormatter.ofPattern("yyyy-MM-dd")) + "
00:00:00",

                LocalDateTime.now().format(DateTimeFormatter.ofPattern("yyyy-MM-dd
HH:mm:ss"))
                                        );
                                } else {
                                        result =
DBConnector.getInstance().getRegisterForDoctor(
                                                        doctorNumber,

                pickerDateStart.getValue().format(DateTimeFormatter.ofPattern("yyyy-MM-dd"))
+

                pickerTimeStart.getValue().format(DateTimeFormatter.ofPattern(" HH:mm:ss")),

                pickerDateEnd.getValue().format(DateTimeFormatter.ofPattern("yyyy-MM-dd")) +

                pickerTimeEnd.getValue().format(DateTimeFormatter.ofPattern(" HH:mm:ss"))
                                        );
                                }

                                try {
                                        listRegister.clear();
                                        while (result.next()) {
                                                listRegister.add(new Register(
```

```
                result.getString(Config.NameTableColumnRegisterNumber),

                result.getString(Config.NameTableColumnPatientName),

                result.getTimestamp(Config.NameTableColumnRegisterDateTime),

                result.getBoolean(Config.NameTableColumnCategoryRegisterIsSpecialist)
                                                ));
                                        }
                                } catch (SQLException e) {
                                        e.printStackTrace();
                                        return;
                                }
                        } else if (mainPane.getSelectionModel().getSelectedItem() == tabIncome)
{
                                ResultSet result;
                                if (checkBoxAllTime.isSelected()) {
                                        result = DBConnector.getInstance().getIncomeInfo(
                                                        "0000-00-00 00:00:00",

                LocalDateTime.now().format(DateTimeFormatter.ofPattern("yyyy-MM-dd
HH:mm:ss"))
                                        );
                                } else if (checkBoxToday.isSelected()) {
                                        result = DBConnector.getInstance().getIncomeInfo(

                LocalDate.now().format(DateTimeFormatter.ofPattern("yyyy-MM-dd")) + "
00:00:00",

                LocalDateTime.now().format(DateTimeFormatter.ofPattern("yyyy-MM-dd
HH:mm:ss"))
                                        );
                                } else {
                                        result = DBConnector.getInstance().getIncomeInfo(

                pickerDateStart.getValue().format(DateTimeFormatter.ofPattern("yyyy-MM-dd"))
+

                pickerTimeStart.getValue().format(DateTimeFormatter.ofPattern(" HH:mm:ss")),

                pickerDateEnd.getValue().format(DateTimeFormatter.ofPattern("yyyy-MM-dd")) +

                pickerTimeEnd.getValue().format(DateTimeFormatter.ofPattern(" HH:mm:ss"))
                                        );
                                }

                                try {
                                        listIncome.clear();
                                        while (result.next()) {
                                                listIncome.add(new Income(

                result.getString("depname"),

                result.getString(Config.NameTableColumnDoctorNumber),

                result.getString("docname"),
```

```
                result.getBoolean(Config.NameTableColumnCategoryRegisterIsSpecialist),

                result.getInt(Config.NameTableColumnRegisterCurrentRegisterCount),

                result.getDouble("sum")
                                                                        )
                                                        );
                                        }
                                } catch (SQLException e) {
                                        e.printStackTrace();
                                        return;
                                }
                        }
                }

                @FXML
                private void tabSelectionChanged(Event event) {
                        if (((Tab) (event.getTarget()))).isSelected()) ;
                }

                @FXML
                private void buttonExitClicked() throws IOException {
                        Scene scene = new
Scene(FXMLLoader.load(getClass().getResource("Login.fxml")));
                        FXRobotHelper.getStages().get(0).setScene(scene);
                }

                @FXML
                void checkBoxAllTimeSelected() {
                        if (checkBoxAllTime.isSelected()) {
                                checkBoxToday.setSelected(false);
                                pickerDateStart.setDisable(true);
                                pickerDateEnd.setDisable(true);
                                pickerTimeStart.setDisable(true);
                                pickerTimeEnd.setDisable(true);
                        } else if (!checkBoxToday.isSelected()) {
                                pickerDateStart.setDisable(false);
                                pickerDateEnd.setDisable(false);
                                pickerTimeStart.setDisable(false);
                                pickerTimeEnd.setDisable(false);
                        }
                }

                @FXML
                void checkBoxTodaySelected() {
                        if (checkBoxToday.isSelected()) {
                                checkBoxAllTime.setSelected(false);
                                pickerDateStart.setDisable(true);
                                pickerDateEnd.setDisable(true);
                                pickerTimeStart.setDisable(true);
                                pickerTimeEnd.setDisable(true);
                        } else if (!checkBoxAllTime.isSelected()) {
                                pickerDateStart.setDisable(false);
                                pickerDateEnd.setDisable(false);
                                pickerTimeStart.setDisable(false);
                                pickerTimeEnd.setDisable(false);
```

```
                }
        }
}

class DateConverter extends StringConverter<LocalDate> {
        DateTimeFormatter formatter = DateTimeFormatter.ofPattern("yyyy-MM-dd");

        @Override
        public String toString(LocalDate localDate) {
                if (localDate != null) {
                        return formatter.format(localDate);
                } else {
                        return "";
                }
        }

        @Override
        public LocalDate fromString(String s) {
                if (s != null && !s.isEmpty()) {
                        return LocalDate.parse(s, formatter);
                } else {
                        return null;
                }
        }
}
```

PatientController.java

```
package hims;

import com.jfoenix.controls.*;
import com.sun.javafx.robot.impl.FXRobotHelper;
import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.fxml.FXML;
import javafx.fxml.FXMLLoader;
import javafx.scene.Scene;
import javafx.scene.control.*;
import javafx.scene.input.KeyCode;
import javafx.scene.layout.GridPane;

import java.io.IOException;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Timestamp;

import javafx.concurrent.*;

abstract class ListItem {
        public String pronounce;

        @Override
        public abstract String toString();

        public abstract void fromSqlResult(ResultSet result) throws SQLException;
```

```
	public String getPronounce() {
		return pronounce;
	}
}

class ListItemNameDepartment extends ListItem {
	public String number;
	public String name;

	@Override
	public String toString() {
		return number + " " + name + " ";
	}

	@Override
	public void fromSqlResult(ResultSet result) throws SQLException {
		number =
result.getString(Config.NameTableColumnDepartmentNumber);
		name = result.getString(Config.NameTableColumnDepartmentName);
		pronounce =
result.getString(Config.NameTableColumnDepartmentPronounce);
	}
}

class ListItemNameDoctor extends ListItem {
	public String number;
	public String departmentNumber;
	public String name;
	public boolean isSpecialist;
	public Timestamp lastLogin;
	public String password;

	@Override
	public String toString() {
		return number + " " + name + " " + (isSpecialist ? "专家" : "普通医师");
	}

	@Override
	public void fromSqlResult(ResultSet result) throws SQLException {
		number = result.getString(Config.NameTableColumnDoctorNumber);
		departmentNumber =
result.getString(Config.NameTableColumnDoctorDepartmentNumber);
		name = result.getString(Config.NameTableColumnDoctorName);
		isSpecialist =
result.getBoolean(Config.NameTableColumnDoctorIsSpecialist);
		lastLogin =
result.getTimestamp(Config.NameTableColumnDoctorLastLogin);
		password = result.getString(Config.NameTableColumnDoctorPassword);
		pronounce =
result.getString(Config.NameTableColumnDoctorPronounce);
	}
}

class ListItemTypeRegister extends ListItem {
	public boolean isSpecialist;

	@Override
```

```
		public String toString() {
			return isSpecialist ? "专家号" : "普通号";
		}

		@Override
		public void fromSqlResult(ResultSet result) {

		}
}

class ListItemNameRegister extends ListItem {
		public String number;
		public String name;
		public Float fee;
		public String department;
		public boolean isSpecialist;
		public int maxNumber;

		@Override
		public String toString() {
			return number + " " + name + " " + (isSpecialist ? "专家号" : "普通号")
+ "¥ " + fee;
		}

		@Override
		public void fromSqlResult(ResultSet result) throws SQLException {
			number =
result.getString(Config.NameTableColumnCategoryRegisterNumber);
			name =
result.getString(Config.NameTableColumnCategoryRegisterName);
			pronounce =
result.getString(Config.NameTableColumnCategoryRegisterPronounce);
			department =
result.getString(Config.NameTableColumnCategoryRegisterDepartment);
			isSpecialist =
result.getBoolean(Config.NameTableColumnCategoryRegisterIsSpecialist);
			maxNumber =
result.getInt(Config.NameTableColumnCategoryRegisterMaxRegisterNumber);
			fee = result.getFloat(Config.NameTableColumnCategoryRegisterFee);
		}
}

public class PatientController {
		static public String patientName;
		static public String patientNumber;
		static public Double patientBalance;

		@FXML
		private GridPane mainPane;

		@FXML
		private Label labelWelcome;
		@FXML
		private JFXComboBox<String> inputNameDepartment;
		@FXML
		private JFXComboBox<String> inputNameDoctor;
		@FXML
```

```
        private JFXComboBox<String> inputTypeRegister;
        @FXML
        private JFXComboBox<String> inputNameRegister;
        @FXML
        private Label labelFee;
        @FXML
        private Label labelRefund;
        @FXML
        private Label labelStatus;
        @FXML
        private JFXTextField inputMoney;
        @FXML
        private JFXButton buttonRegister;
        @FXML
        private JFXButton buttonExit;
        @FXML
        private JFXCheckBox checkBoxUseBalance;
        @FXML
        private JFXCheckBox checkBoxAddToBalance;

        private int lastIndexInputNameDepartment = -1;
        private int lastIndexInputNameDoctor = -1;
        private int lastIndexInputTypeRegister = -1;
        private int lastIndexInputNameRegister = -1;

        private ObservableList<ListItemNameDepartment> listNameDepartment =
FXCollections.observableArrayList();
        private ObservableList<ListItemNameDoctor> listNameDoctor =
FXCollections.observableArrayList();
        private ObservableList<ListItemTypeRegister> listTypeRegister =
FXCollections.observableArrayList();
        private ObservableList<ListItemNameRegister> listNameRegister =
FXCollections.observableArrayList();

        private ObservableList<ListItemNameDepartment> listNameDepartmentFiltered =
FXCollections.observableArrayList();
        private ObservableList<ListItemNameDoctor> listNameDoctorFiltered =
FXCollections.observableArrayList();
        private ObservableList<ListItemTypeRegister> listTypeRegisterFiltered =
FXCollections.observableArrayList();
        private ObservableList<ListItemNameRegister> listNameRegisterFiltered =
FXCollections.observableArrayList();

        private <ItemType extends ListItem> boolean updateOneSetOfData(
                        String tableName,
                        ObservableList<ItemType> list,
                        Class<ItemType> clazz) {

                ResultSet result =
DBConnector.getInstance().getWholeTable(tableName);

                if (result != null) {
                        ObservableList<ItemType> tempList =
FXCollections.observableArrayList();
                        try {
                                while (result.next()) {
                                        ItemType item = clazz.newInstance();
```

```
                                        item.fromSqlResult(result);
                                        tempList.add(item);
                                }
                        } catch (Exception e) {
                                e.printStackTrace();
                                System.exit(-1);
                        }
                        list.clear();
                        list.addAll(tempList);
                        return true;
                }
                return true;
        }

        private int getMoneyData() {
                if (inputMoney.getText().isEmpty())
                        return 0;
                else
                        return Integer.parseInt(inputMoney.getText().trim());
        }

        public void updateData() {
                updateOneSetOfData(
                                Config.NameTableDepartment,
                                listNameDepartment,
                                ListItemNameDepartment.class
                );

                updateOneSetOfData(
                                Config.NameTableDoctor,
                                listNameDoctor,
                                ListItemNameDoctor.class
                );

                updateOneSetOfData(
                                Config.NameTableCategoryRegister,
                                listNameRegister,
                                ListItemNameRegister.class
                );
                ListItemTypeRegister itemSpecialist = new ListItemTypeRegister();
                ListItemTypeRegister itemNormal = new ListItemTypeRegister();
                itemSpecialist.isSpecialist = true;
                itemSpecialist.pronounce = "zhuanjiahao";
                itemNormal.isSpecialist = false;
                itemNormal.pronounce = "potonghao";
                listTypeRegister.clear();
                listTypeRegister.add(itemSpecialist);
                listTypeRegister.add(itemNormal);
        }


        private void updateRefund() {
                int index = inputNameRegister.getSelectionModel().getSelectedIndex();
                if (index != -1 && checkBoxUseBalance.isSelected()) {
                        labelRefund.setText("¥ 0");
                        labelRefund.setStyle("");
                        return;
```

```
                }

                if (index != -1 && getMoneyData() >
listNameRegisterFiltered.get(index).fee) {
                        labelRefund.setText(String.format("￥%.2f", getMoneyData() -
listNameRegisterFiltered.get(index).fee));
                        labelRefund.setStyle("");
                } else if (index != -1) {
                        labelRefund.setText("交款金额不足");
                        labelRefund.setStyle("-fx-text-fill: red;");
                }
        }

        private void updateUseBalance() {
                int index = inputNameRegister.getSelectionModel().getSelectedIndex();
                if (index != -1 && patientBalance <
listNameRegisterFiltered.get(index).fee) {
                        checkBoxUseBalance.setSelected(false);
                        inputMoney.setDisable(false);
                        checkBoxUseBalance.setText("余额不足");
                        checkBoxUseBalance.setDisable(true);
                } else {
                        checkBoxUseBalance.setDisable(false);
                        checkBoxUseBalance.setText("使用余额付款");
                        checkBoxUseBalance.setSelected(true);
                        inputMoney.setDisable(true);
                }
        }

        private void updateRegisterButton() {
                buttonRegister.setDisable(true);
                int index;
                if (inputNameDoctor.getSelectionModel().getSelectedIndex() != -1 &&
                                (index =
inputNameRegister.getSelectionModel().getSelectedIndex()) != -1 &&
                                ((checkBoxUseBalance.isSelected() &&
patientBalance >= listNameRegisterFiltered.get(index).fee) ||

        (!checkBoxUseBalance.isSelected() && getMoneyData() >=
listNameRegisterFiltered.get(index).fee))) {
                        buttonRegister.setDisable(false);
                }
        }

        @FXML
        void useBalanceClicked() {
                if (checkBoxUseBalance.isSelected()) {
                        inputMoney.setDisable(true);
                        updateRefund();
                } else {
                        inputMoney.setDisable(false);
                        updateRefund();
                }
                updateRegisterButton();
        }

        private void updateUserDisplayInfo() {
```

```
                labelWelcome.setText(String.format("欢迎进入医院挂号系统，%s！
余额：￥%.2f", patientName, patientBalance));
        }

        @FXML
        public void initialize() {
                updateUserDisplayInfo();
                updateData();
                inputNameDepartment.setItems(FXCollections.observableArrayList());
                inputNameDoctor.setItems(FXCollections.observableArrayList());
                inputTypeRegister.setItems(FXCollections.observableArrayList());
                inputNameRegister.setItems(FXCollections.observableArrayList());
                reFilterDepartment(false);
                reFilterDoctor(false);
                reFilterRegisterType(false);
                reFilterRegisterName(false);
                updateRegisterButton();
                inputNameDepartment.getEditor().setOnKeyReleased(keyEvent -> {
                        if (shouldSupressKeyCode(keyEvent.getCode()))
                                return;
                        reFilterDepartment(true);
                        reFilterDoctor(false);
                        reFilterRegisterType(false);
                        reFilterRegisterName(false);
                        if (!inputNameDepartment.isShowing()) {
                                inputNameDepartment.show();
                        } else {
                                inputNameDepartment.hide();
                                inputNameDepartment.show();
                        }
                });
                inputNameDepartment.addEventHandler(ComboBox.ON_HIDDEN, e ->
{
                        int index;
                        if ((index =
inputNameDepartment.getSelectionModel().getSelectedIndex())
                                        != lastIndexInputNameDepartment) {
                                lastIndexInputNameDepartment = index;
                                reFilterDoctor(false);
                                reFilterRegisterType(false);
                                reFilterRegisterName(false);
                        }
                        e.consume();
                });


                inputNameDoctor.getEditor().setOnKeyReleased(keyEvent -> {
                        if (shouldSupressKeyCode(keyEvent.getCode()))
                                return;
                        reFilterDoctor(true);
                        reFilterDepartment(false);
                        reFilterRegisterType(false);
                        reFilterRegisterName(false);
                        if (!inputNameDoctor.isShowing()) {
                                inputNameDoctor.show();
                        } else {
                                inputNameDoctor.hide();
```

```
                                        inputNameDoctor.show();
                                }
                        });
                        inputNameDoctor.addEventHandler(ComboBox.ON_HIDDEN, e -> {
                                int index;
                                if ((index =
inputNameDoctor.getSelectionModel().getSelectedIndex())
                                                != lastIndexInputNameDoctor) {
                                        lastIndexInputNameDoctor = index;
                                        reFilterDepartment(false);
                                        reFilterRegisterType(false);
                                        reFilterRegisterName(false);
                                }
                                inputNameDoctor.setStyle("");
                                updateRegisterButton();
                                e.consume();
                        });
                        inputNameDoctor.setOnMousePressed(mouseEvent -> {
                                inputNameDoctor.setStyle("");
                        });

                        inputTypeRegister.getEditor().setOnKeyReleased(keyEvent -> {
                                if (shouldSupressKeyCode(keyEvent.getCode()))
                                        return;
                                reFilterRegisterType(true);
                                reFilterDepartment(false);
                                reFilterDoctor(false);
                                reFilterRegisterName(false);
                                if (!inputTypeRegister.isShowing()) {
                                        inputTypeRegister.show();
                                } else {
                                        inputTypeRegister.hide();
                                        inputTypeRegister.show();
                                }
                        });
                        inputTypeRegister.addEventHandler(ComboBox.ON_HIDDEN, e -> {
                                int index;
                                if ((index =
inputTypeRegister.getSelectionModel().getSelectedIndex())
                                                != lastIndexInputTypeRegister) {
                                        lastIndexInputTypeRegister = index;
                                        reFilterDepartment(false);
                                        reFilterDoctor(false);
                                        reFilterRegisterName(false);
                                }
                                updateRegisterButton();
                                e.consume();
                        });

                        inputNameRegister.getEditor().setOnKeyReleased(keyEvent -> {
                                if (shouldSupressKeyCode(keyEvent.getCode()))
                                        return;
                                reFilterRegisterName(true);
                                reFilterDepartment(false);
                                reFilterDoctor(false);
                                reFilterRegisterType(false);
                                if (!inputNameRegister.isShowing()) {
```

```
                                        inputNameRegister.show();
                                } else {
                                        inputNameRegister.hide();
                                        inputNameRegister.show();
                                }
                        });
                        inputNameRegister.addEventHandler(ComboBox.ON_HIDDEN, e -> {
                                int index;
                                if ((index =
inputNameRegister.getSelectionModel().getSelectedIndex())
                                                != lastIndexInputNameRegister) {
                                        lastIndexInputNameRegister = index;
                                        reFilterDepartment(false);
                                        reFilterDoctor(false);
                                        reFilterRegisterType(false);
                                }
                                inputNameRegister.setStyle("");
                                if (index != -1) {
                                        float fee = listNameRegisterFiltered.get(index).fee;
                                        labelFee.setText("¥ " + fee);
                                }
                                updateUseBalance();
                                updateRefund();
                                updateRegisterButton();
                                e.consume();
                        });
                        inputNameRegister.setOnMousePressed(mouseEvent -> {
                                inputNameRegister.setStyle("");
                        });
                        buttonRegister.setOnKeyReleased(keyEvent -> {
                                if (keyEvent.getCode() == KeyCode.ENTER)
                                        buttonRegisterPressed();
                        });

                        buttonExit.setOnKeyReleased(keyEvent -> {
                                try {
                                        if (keyEvent.getCode() == KeyCode.ENTER)
                                                buttonExitClicked();
                                } catch (IOException e) {
                                }
                        });

                        checkBoxUseBalance.setOnKeyReleased(keyEvent -> {
                                if (keyEvent.getCode() == KeyCode.SPACE)
                                        useBalanceClicked();
                                else
                                        keyEvent.consume();
                        });

        }

        @FXML
        private void InputMoneyFinished() {
                updateRefund();
                updateRegisterButton();
        }
```

```
        @FXML
        private void buttonRegisterPressed() {
                if (inputNameDoctor.getSelectionModel().getSelectedIndex() == -1) {
                        statusError("请选择医生姓名");
                        inputNameDoctor.setStyle("-fx-background-color: pink;");
                        return;
                }
                if (inputNameRegister.getSelectionModel().getSelectedIndex() == -1) {
                        statusError("请选择号种名称");
                        inputNameRegister.setStyle("-fx-background-color: pink;");
                        return;
                }
                int index;
                if (!((index =
inputNameRegister.getSelectionModel().getSelectedIndex()) != -1 &&

        inputNameDoctor.getSelectionModel().getSelectedIndex() != -1 && (
                                        (checkBoxUseBalance.isSelected() &&
patientBalance >= listNameRegisterFiltered.get(index).fee) ||

        (!checkBoxUseBalance.isSelected() && getMoneyData() >=
listNameRegisterFiltered.get(index).fee))) {
                                statusError("缴费金额不足或余额不足");
                                return;
                }

                disableEverything();
                TryRegisterService service = new TryRegisterService(

        listNameRegisterFiltered.get(inputNameRegister.getSelectionModel().getSelectedI
ndex()).number,

        listNameDoctorFiltered.get(inputNameDoctor.getSelectionModel().getSelectedInde
x()).number,
                                        patientNumber,

        listNameRegisterFiltered.get(inputNameRegister.getSelectionModel().getSelectedI
ndex()).fee,
                                        checkBoxUseBalance.isSelected(),
                                        ((!checkBoxUseBalance.isSelected() &&
checkBoxAddToBalance.isSelected()) ?
                                                        getMoneyData() -
listNameRegisterFiltered.get(index).fee : 0)
                );
                service.setOnSucceeded(workerStateEvent -> {
                        switch (service.returnCode) {
                                case registerNumberExceeded:
                                        statusError("此号已达到人数上限。");
                                        break;
                                case registerCategoryNotFound:
                                case sqlException:
                                        statusError("数据库错误，请联系管理员
");
                                        break;
                                case retryTimeExceeded:
                                        statusError("系统繁忙，请稍候再试");
                                        break;
```

```
                                        case noError:
                                                labelStatus.setText("挂号成功，挂号号
码：" + service.registerNumber);
                                                patientBalance = service.updatedBalance;
                                                updateUserDisplayInfo();
                                                break;
                                }
                                enableEverything();
                        });
                        service.start();
                }

                private void disableEverything() {
                        mainPane.setDisable(true);
                }

                private void enableEverything() {
                        mainPane.setDisable(false);
                }

                private void statusError(String error) {
                        labelStatus.setText(error);
                        labelStatus.setStyle("-fx-text-fill: red;");
                }

                private void reFilterDepartment(boolean withoutSelect) {
                        int index;
                        String previousKey = "";
                        if ((index =
inputNameDepartment.getSelectionModel().getSelectedIndex()) != -1)
                                previousKey =
listNameDepartmentFiltered.get(index).number;

                        ObservableList<ListItemNameDepartment> list0 =
FXCollections.observableArrayList();
                        ObservableList<ListItemNameDepartment> list1 =
FXCollections.observableArrayList();

                        for (ListItemNameDepartment listItemNameDepartment :
listNameDepartment) {
                                if
(listItemNameDepartment.toString().contains(inputNameDepartment.getEditor().getText().tri
m()) ||

                listItemNameDepartment.getPronounce().contains(inputNameDepartment.getEditor
().getText().trim())) {

                listNameDepartmentFiltered.add(listItemNameDepartment);
                                        list0.add(listItemNameDepartment);
                                }
                        }

                        if ((index =
inputNameDoctor.getSelectionModel().getSelectedIndex()) != -1) {
                                for (ListItemNameDepartment department : list0)
                                        if
(department.number.equals(listNameDoctorFiltered.get(index).departmentNumber))
```

```
                                                list1.add(department);
                                        list0 = list1;
                                }

                                boolean isCurrentInputLegal = false;
                                int counter = 0, newSelection = -1;
                                inputNameDepartment.getItems().clear();
                                listNameDepartmentFiltered.clear();
                                for (ListItemNameDepartment department : list0) {
                                        inputNameDepartment.getItems().add(department.toString());
                                        listNameDepartmentFiltered.add(department);
                                        if
(department.toString().contains(inputNameDepartment.getEditor().getText().trim()) ||

                department.getPronounce().contains(inputNameDepartment.getEditor().getText().tr
im()))
                                                isCurrentInputLegal = true;
                                        if (previousKey.equals(department.number))
                                                newSelection = counter;
                                        ++counter;
                                }

                                if (!withoutSelect) {
                                        if (!isCurrentInputLegal)
                                                inputNameDepartment.getEditor().clear();
                                        if (newSelection != -1) {

                inputNameDepartment.getSelectionModel().clearAndSelect(newSelection);

                inputNameDepartment.getEditor().setText(inputNameDepartment.getItems().get(ne
wSelection));
                                        }
                                }
                        }

                        private void reFilterDoctor(boolean withoutSelect) {
                                int index;
                                String previousKey = "";
                                if ((index =
inputNameDoctor.getSelectionModel().getSelectedIndex()) != -1)
                                        previousKey = listNameDoctorFiltered.get(index).number;

                                ObservableList<ListItemNameDoctor> list0 =
FXCollections.observableArrayList();
                                ObservableList<ListItemNameDoctor> list1 =
FXCollections.observableArrayList();

                                for (ListItemNameDoctor listItemNameDoctor : listNameDoctor)
                                        if
(listItemNameDoctor.toString().contains(inputNameDoctor.getEditor().getText().trim()) ||

                listItemNameDoctor.getPronounce().contains(inputNameDoctor.getEditor().getTex
t().trim()))
                                                list0.add(listItemNameDoctor);

                                if ((index =
inputNameDepartment.getSelectionModel().getSelectedIndex()) != -1) {
```

```
                                for (ListItemNameDoctor listItemNameDoctor : list0)
                                        if
(listItemNameDoctor.departmentNumber.equals(listNameDepartmentFiltered.get(index).num
ber))
                                                list1.add(listItemNameDoctor);
                                list0 = list1;
                        }

                        list1 = FXCollections.observableArrayList();
                        if ((index =
inputTypeRegister.getSelectionModel().getSelectedIndex()) != -1) {
                                for (ListItemNameDoctor doctor : list0)
                                        if (doctor.isSpecialist
|| !listTypeRegisterFiltered.get(index).isSpecialist)
                                                list1.add(doctor);
                                list0 = list1;
                        }

                        list1 = FXCollections.observableArrayList();
                        if ((index =
inputNameRegister.getSelectionModel().getSelectedIndex()) != -1) {
                                for (ListItemNameDoctor doctor : list0)
                                        if
(doctor.departmentNumber.equals(listNameRegisterFiltered.get(index).department))
                                                list1.add(doctor);
                                list0 = list1;
                        }

                        boolean isCurrentInputLegal = false;
                        int counter = 0, newSelection = -1;
                        inputNameDoctor.getItems().clear();
                        listNameDoctorFiltered.clear();
                        for (ListItemNameDoctor doctor : list0) {
                                listNameDoctorFiltered.add(doctor);
                                inputNameDoctor.getItems().add(doctor.toString());
                                if
(doctor.toString().contains(inputNameDoctor.getEditor().getText().trim()) ||

        doctor.getPronounce().contains(inputNameDoctor.getEditor().getText().trim()))
                                        isCurrentInputLegal = true;
                                if (previousKey.equals(doctor.number))
                                        newSelection = counter;
                                ++counter;
                        }

                        if (!withoutSelect) {
                                if (!isCurrentInputLegal)
                                        inputNameDoctor.getEditor().clear();
                                if (newSelection != -1) {

        inputNameDoctor.getSelectionModel().clearAndSelect(counter);

        inputNameDoctor.getEditor().setText(inputNameDoctor.getItems().get(newSelectio
n));
                                }
                        }
                }
```

```
        private void reFilterRegisterType(boolean withoutSelect) {
                int index;
                String previousKey = "";
                if ((index =
inputTypeRegister.getSelectionModel().getSelectedIndex()) != -1)
                        previousKey = listTypeRegisterFiltered.get(index).pronounce;

                ObservableList<ListItemTypeRegister> list0 =
FXCollections.observableArrayList();
                ObservableList<ListItemTypeRegister> list1 =
FXCollections.observableArrayList();

                for (ListItemTypeRegister listItemTypeRegister : listTypeRegister)
                        if
(listItemTypeRegister.toString().contains(inputTypeRegister.getEditor().getText().trim()) ||

        listItemTypeRegister.getPronounce().contains(inputTypeRegister.getEditor().getTe
xt().trim()))
                                list0.add(listItemTypeRegister);

                if ((index =
inputNameDoctor.getSelectionModel().getSelectedIndex()) != -1) {
                        for (ListItemTypeRegister listItemTypeRegister : list0)
                                if (listNameDoctorFiltered.get(index).isSpecialist
|| !listItemTypeRegister.isSpecialist)
                                        list1.add(listItemTypeRegister);
                        list0 = list1;
                }

                list1 = FXCollections.observableArrayList();
                if ((index =
inputNameRegister.getSelectionModel().getSelectedIndex()) != -1) {
                        for (ListItemTypeRegister register : list0)
                                if (register.isSpecialist ==
listNameRegisterFiltered.get(index).isSpecialist)
                                        list1.add(register);
                        list0 = list1;
                }

                boolean isCurrentInputLegal = false;
                int counter = 0, newSelection = -1;
                listTypeRegisterFiltered.clear();
                inputTypeRegister.getItems().clear();
                for (ListItemTypeRegister register : list0) {
                        listTypeRegisterFiltered.add(register);
                        inputTypeRegister.getItems().add(register.toString());
                        if
(register.toString().contains(inputTypeRegister.getEditor().getText().trim()) ||

        register.getPronounce().contains(inputTypeRegister.getEditor().getText().trim()))
                                isCurrentInputLegal = true;
                        if (previousKey.equals(register.pronounce))
                                newSelection = counter;
                        ++counter;
                }
```

```
                    if (!withoutSelect) {
                            if (!isCurrentInputLegal)
                                    inputTypeRegister.getEditor().clear();
                            if (newSelection != -1) {

            inputTypeRegister.getSelectionModel().clearAndSelect(newSelection);

            inputTypeRegister.getEditor().setText(inputTypeRegister.getItems().get(newSelecti
on));
                            }
                    }
            }

            private void reFilterRegisterName(boolean withoutSelect) {
                    int index;
                    String previousKey = "";
                    if ((index =
inputNameRegister.getSelectionModel().getSelectedIndex()) != -1)
                            previousKey = listNameRegisterFiltered.get(index).number;

                    ObservableList<ListItemNameRegister> list0 =
FXCollections.observableArrayList();
                    ObservableList<ListItemNameRegister> list1 =
FXCollections.observableArrayList();

                    for (ListItemNameRegister listItemNameRegister : listNameRegister)
                            if
(listItemNameRegister.toString().contains(inputNameRegister.getEditor().getText().trim()) ||

            listItemNameRegister.getPronounce().contains(inputNameRegister.getEditor().getT
ext().trim()))
                                    list0.add(listItemNameRegister);

                    if ((index =
inputNameDepartment.getSelectionModel().getSelectedIndex()) != -1) {
                            for (ListItemNameRegister listItemNameRegister : list0)
                                    if
(listItemNameRegister.department.equals(listNameDepartmentFiltered.get(index).number))
                                            list1.add(listItemNameRegister);
                            list0 = list1;
                    }

                    list1 = FXCollections.observableArrayList();
                    if ((index =
inputNameDoctor.getSelectionModel().getSelectedIndex()) != -1) {
                            for (ListItemNameRegister listItemNameRegister : list0)
                                    if (!listItemNameRegister.isSpecialist ||
listNameDoctorFiltered.get(index).isSpecialist)
                                            list1.add(listItemNameRegister);
                            list0 = list1;
                    }

                    list1 = FXCollections.observableArrayList();
                    if ((index =
inputTypeRegister.getSelectionModel().getSelectedIndex()) != -1) {
                            for (ListItemNameRegister listItemNameRegister : list0)
```

```
                                        if (listItemNameRegister.isSpecialist ==
listTypeRegisterFiltered.get(index).isSpecialist)
                                                list1.add(listItemNameRegister);
                                list0 = list1;
                        }

                        boolean isCurrentInputLegal = false;
                        int counter = 0, newSelection = -1;
                        listNameRegisterFiltered.clear();
                        inputNameRegister.getItems().clear();
                        for (ListItemNameRegister listItemNameRegister : list0) {
                                listNameRegisterFiltered.add(listItemNameRegister);

            inputNameRegister.getItems().add(listItemNameRegister.toString());
                                if
(listItemNameRegister.toString().contains(inputNameRegister.getEditor().getText().trim()) ||

            listItemNameRegister.getPronounce().contains(inputNameRegister.getEditor().getT
ext().trim()))
                                        isCurrentInputLegal = true;
                                if (previousKey.equals(listItemNameRegister.number))
                                        newSelection = counter;
                                ++counter;
                        }

                        if (!withoutSelect) {
                                if (!isCurrentInputLegal)
                                        inputNameRegister.getEditor().clear();
                                if (newSelection != -1) {

            inputNameRegister.getSelectionModel().clearAndSelect(newSelection);

            inputNameRegister.getEditor().setText(inputNameRegister.getItems().get(newSele
ction));
                                }
                        }
            }

            private boolean shouldSupressKeyCode(KeyCode code) {
                        return code == KeyCode.DOWN ||
                                        code == KeyCode.UP ||
                                        code == KeyCode.ENTER;
            }

            @FXML
            void buttonExitClicked() throws IOException {
                        Scene scene = new
Scene(FXMLLoader.load(getClass().getResource("Login.fxml")));
                        FXRobotHelper.getStages().get(0).setScene(scene);
            }
}

class TryRegisterService extends Service {
            String registerCategoryNumber;
            String doctorNumber;
            String patientNumber;
            double registerFee;
```

```
	boolean deductFromBalance;
	double addToBalance;
	int retry = 5;

	int registerNumber;
	RegisterException.ErrorCode returnCode;
	double updatedBalance;

	public void setRetry(int retry) {
		this.retry = retry;
	}

	TryRegisterService(
				String regCatNum,
				String docNum,
				String patientNum,
				double regFee,
				boolean deduct,
				double add) {
		registerCategoryNumber = regCatNum;
		doctorNumber = docNum;
		patientNumber = patientNum;
		registerFee = regFee;
		deductFromBalance = deduct;
		addToBalance = add;
	}

	@Override
	protected Task createTask() {
		return new Task() {
			@Override
			protected Object call() throws Exception {
				boolean retryFlag = false;
				do {
					try {
						registerNumber =
DBConnector.getInstance().tryRegister(

	registerCategoryNumber,

	doctorNumber,

	patientNumber,

	registerFee,

	deductFromBalance,

	addToBalance);
					} catch (RegisterException e) {
						retryFlag = true;
						switch (e.error) {
							case sqlException:

	returnCode = RegisterException.ErrorCode.sqlException;
							break;
```

```
                                                                    case
registerNumberExceeded:
                                                                    case
registerCategoryNotFound:
                                                                    case patientNotExist:

            returnCode = e.error;
                                                                                return
null;
                                                                    }
                                                        }
                                                } while (retryFlag && --retry > 0);

                                                if (retry == 0)
                                                        returnCode =
RegisterException.ErrorCode.retryTimeExceeded;
                                                else {
                                                        returnCode =
RegisterException.ErrorCode.noError;
                                                        try {
                                                                ResultSet patientInfo =
DBConnector.getInstance().getPatientInfo(patientNumber);
                                                                if (!patientInfo.next())
                                                                        returnCode =
RegisterException.ErrorCode.patientNotExist;
                                                                updatedBalance =
patientInfo.getDouble(Config.NameTableColumnPatientBalance);
                                                        } catch (SQLException e) {
                                                                returnCode =
RegisterException.ErrorCode.sqlException;
                                                                return null;
                                                        }
                                                }
                                                return null;
                                        }
                                };
                        }
}
```

DBConnector.java

```
package hims;

import java.sql.*;

public class DBConnector {
        private static DBConnector instance = null;
        private Connection connection;
        private Connection transactionConnection;
        private Statement statement;
        private Statement transactionStatement;

        private DBConnector() throws ClassNotFoundException {
```

```
                Class.forName("com.mysql.jdbc.Driver");
        }

        static public DBConnector getInstance() {
                try {
                        if (instance == null)
                                instance = new DBConnector();
                } catch (ClassNotFoundException e) {
                        System.err.print("cannot load sql driver.");
                        System.exit(1);
                }
                return instance;
        }

        public void connectDataBase(
                        String hostName,
                        Integer port,
                        String dbName,
                        String userName,
                        String password) throws SQLException {
                String url = "jdbc:mysql://" + hostName +
                                ":" + port +
                                "/" + dbName +

        "?zeroDateTimeBehavior=convertToNull&autoReconnect=true&characterEncoding
=UTF-8&characterSetResults=UTF-8";
                connection = DriverManager.getConnection(url, userName, password);
                statement = connection.createStatement();
                transactionConnection = DriverManager.getConnection(url, userName,
password);
                transactionConnection.setAutoCommit(false);
                transactionStatement = transactionConnection.createStatement();
        }


        public ResultSet getWholeTable(String tableName) {
                try {
                        return statement.executeQuery("select * from " + tableName);
                } catch (SQLException e) {
                        return null;
                }
        }

        public ResultSet getPatientInfo(String number) {
                try {
                        return statement.executeQuery(
                                        "select * from " +
Config.NameTablePatient +
                                                        " where " +
Config.NameTableColumnPatientNumber + "=" + number);
                } catch (SQLException e) {
                        return null;
                }
        }

        public ResultSet getDoctorInfo(String number) {
                try {
```

```
                return statement.executeQuery(
                                "select * from " +
Config.NameTableDoctor +
                                                " where " +
Config.NameTableColumnDoctorNumber + "=" + number);
        } catch (SQLException e) {
                return null;
        }
    }

    public int tryRegister(
                String registerCategoryNumber,
                String doctorNumber,
                String patientNumber,
                Double registerFee,
                boolean deductFromBalance,
                Double addToBalance) throws RegisterException {
        try {
                ResultSet result = transactionStatement.executeQuery(
                                "select * from " +
Config.NameTableRegister +
                                                " order by " +
Config.NameTableColumnRegisterNumber +
                                                " desc limit 1"
                );
                int regNumber, currentCount;
                if (!result.next())
                        regNumber = 0;
                else
                        regNumber =
Integer.parseInt(result.getString(Config.NameTableColumnRegisterNumber)) + 1;

                result = transactionStatement.executeQuery(
                                "select * from " +
Config.NameTableRegister +
                                                " where " +
Config.NameTableColumnRegisterCategoryNumber +
                                                "=" +
registerCategoryNumber +
                                                " order by " +
Config.NameTableColumnCategoryRegisterNumber +
                                                " desc limit 1"
                );
                if (!result.next())
                        currentCount = 0;
                else
                        currentCount =
result.getInt(Config.NameTableColumnRegisterCurrentRegisterCount);

                result = transactionStatement.executeQuery(
                                "select * from " +
Config.NameTablePatient +
                                                " where " +
Config.NameTableColumnPatientNumber +
                                                "=" + patientNumber
                );
                if (!result.next())
```

```
                                        throw new RegisterException("patient does not
exist", RegisterException.ErrorCode.patientNotExist);

                                double balance =
result.getDouble(Config.NameTableColumnPatientBalance);

                                result = transactionStatement.executeQuery(
                                                "select " +
Config.NameTableColumnCategoryRegisterMaxRegisterNumber +
                                                        " from " +
Config.NameTableCategoryRegister +
                                                        " where " +
Config.NameTableColumnCategoryRegisterNumber +
                                                        "=" +
registerCategoryNumber
                                );
                                int maxRegCount;
                                if (!result.next())
                                        throw new RegisterException("illegal table entry",

        RegisterException.ErrorCode.registerCategoryNotFound);
                                maxRegCount =
result.getInt(Config.NameTableColumnCategoryRegisterMaxRegisterNumber);

                                if (currentCount > maxRegCount) {
                                        throw new RegisterException("max register number
reached",

        RegisterException.ErrorCode.registerNumberExceeded);
                                }

                                transactionStatement.executeUpdate(
                                                String.format(
                                                        "insert into %s
values (\"%06d\",\"%s\",\"%s\",\"%s\",%d,false,%s, current_timestamp)",

        Config.NameTableRegister,
                                                        regNumber,

        registerCategoryNumber,
                                                        doctorNumber,
                                                        patientNumber,
                                                        currentCount + 1,
                                                        registerFee
                                                )
                                );

                                if (deductFromBalance) {
                                        transactionStatement.executeUpdate(
                                                        String.format("update %s
set %s=%.2f where %s=%s",

        Config.NameTablePatient,

        Config.NameTableColumnPatientBalance,
                                                                (balance
-= registerFee),
```

```
            Config.NameTableColumnPatientNumber,

            patientNumber)
                                    );
                            }

                            if (addToBalance != 0) {
                                    transactionStatement.executeUpdate(
                                                    String.format("update %s
set %s=%.2f where %s=%s",

            Config.NameTablePatient,

            Config.NameTableColumnPatientBalance,
                                                                    (balance
+= addToBalance),

            Config.NameTableColumnPatientNumber,

            patientNumber)
                                    );
                            }

                            transactionConnection.commit();
                            return regNumber;
                    } catch (SQLException e) {
                            try {
                                    transactionConnection.rollback();
                            } catch (SQLException ee) {
                            }
                            throw new RegisterException("sql exception occurred",
RegisterException.ErrorCode.sqlException);
                    }
            }

            public ResultSet getRegisterForDoctor(String docNumber, String startTime, String
endTime) {
                    try {
                            String sql = "select reg." +
Config.NameTableColumnRegisterNumber +
                                            ",pat." +
Config.NameTableColumnPatientName +
                                            ",reg." +
Config.NameTableColumnRegisterDateTime +
                                            ",cat." +
Config.NameTableColumnCategoryRegisterIsSpecialist + (
                                            " from (select " +
Config.NameTableColumnRegisterNumber +
                                                            "," +
Config.NameTableColumnRegisterPatientNumber +
                                                            "," +
Config.NameTableColumnRegisterDateTime +
                                                            "," +
Config.NameTableColumnRegisterCategoryNumber +
                                                            " from " +
Config.NameTableRegister +
```

```
                                                                    " where " +
Config.NameTableColumnRegisterDoctorNumber +
                                                                    "=" + docNumber +
                                                                    " and " +
Config.NameTableColumnRegisterDateTime +
                                                                    ">=\"" + startTime +
                                                                    "\" and " +
Config.NameTableColumnRegisterDateTime +
                                                                    "<=\"" + endTime +
                                                                    "\") as reg") + (
                                                    " inner join (select " +
Config.NameTableColumnPatientNumber +
                                                                    "," +
Config.NameTableColumnPatientName +
                                                                    " from " +
Config.NameTablePatient +
                                                                    ") as pat") +
                                                    " on reg." +
Config.NameTableColumnRegisterPatientNumber +
                                                    "=pat." +
Config.NameTableColumnPatientNumber + (
                                                    " inner join (select " +
Config.NameTableColumnCategoryRegisterNumber +
                                                                    "," +
Config.NameTableColumnCategoryRegisterIsSpecialist +
                                                                    " from " +
Config.NameTableCategoryRegister +
                                                                    ") as cat") +
                                                    " on reg." +
Config.NameTableColumnRegisterCategoryNumber +
                                                    "=cat." +
Config.NameTableColumnCategoryRegisterNumber;
                                    return statement.executeQuery(sql);
                            } catch (SQLException e) {
                                    e.printStackTrace();
                                    return null;
                            }
                    }

                    public ResultSet getIncomeInfo(String startTime, String endTime) {
                            try {
                                    String sql = "select dep." +
Config.NameTableColumnDepartmentName +
                                                    " as depname,reg." +
Config.NameTableColumnRegisterDoctorNumber +
                                                    ",doc." +
Config.NameTableColumnDoctorName +
                                                    " as docname,cat." +
Config.NameTableColumnCategoryRegisterIsSpecialist +
                                                    ",reg." +
Config.NameTableColumnRegisterCurrentRegisterCount +
                                                    ",SUM(reg." +
Config.NameTableColumnRegisterFee +
                                                    ") as sum from" + (
                                                    " (select * from " +
Config.NameTableRegister +
```

```
                                                                    " where " +
Config.NameTableColumnRegisterDateTime +
                                                                    ">=\"" + startTime +
                                                                    "\" and " +
Config.NameTableColumnRegisterDateTime +
                                                                    "<=\"" + endTime +
                                                                    "\") as reg") +
                                                    " inner join" + (
                                                    " (select " +
Config.NameTableColumnDoctorNumber +
                                                                    "," +
Config.NameTableColumnDoctorName +
                                                                    "," +
Config.NameTableColumnDoctorDepartmentNumber +
                                                                    " from " +
Config.NameTableDoctor +
                                                                    ") as doc") +
                                                    " on reg." +
Config.NameTableColumnRegisterDoctorNumber +
                                                    "=doc." +
Config.NameTableColumnDoctorNumber +
                                                    " inner join" + (
                                                    " (select " +
Config.NameTableColumnDepartmentNumber +
                                                                    "," +
Config.NameTableColumnDepartmentName +
                                                                    " from " +
Config.NameTableDepartment +
                                                                    ") as dep") +
                                                    " on doc." +
Config.NameTableColumnDoctorDepartmentNumber +
                                                    "=dep." +
Config.NameTableColumnDepartmentNumber +
                                                    " inner join" + (
                                                    " (select " +
Config.NameTableColumnCategoryRegisterNumber +
                                                                    "," +
Config.NameTableColumnCategoryRegisterIsSpecialist +
                                                                    " from " +
Config.NameTableCategoryRegister +
                                                                    ") as cat") +
                                                    " on reg." +
Config.NameTableColumnRegisterCategoryNumber +
                                                    "=cat." +
Config.NameTableColumnCategoryRegisterNumber +
                                                    " group by reg." +
Config.NameTableColumnRegisterDoctorNumber +
                                                    ",cat." +
Config.NameTableColumnCategoryRegisterIsSpecialist;
                        return statement.executeQuery(sql);
                } catch (SQLException e) {
                        e.printStackTrace();
                        return null;
                }
        }

        public void updatePatientLoginTime(String patientId, String time) {
```

```
                try {
                        statement.executeUpdate(
                                        "update " + Config.NameTablePatient +
                                                        " set " +
Config.NameTableColumnPatientLastLogin +
                                                        "=\"" + time +
                                                        "\" where " +
Config.NameTableColumnPatientNumber +
                                                        "=" + patientId
                        );
                } catch (SQLException e) {
                        e.printStackTrace();
                        return;
                }
        }

        public void updateDoctorLoginTime(String doctorId, String time) {
                try {
                        statement.executeUpdate(
                                        "update " + Config.NameTableDoctor +
                                                        " set " +
Config.NameTableColumnDoctorLastLogin +
                                                        "=\"" + time +
                                                        "\" where " +
Config.NameTableColumnRegisterDoctorNumber +
                                                        "=" + doctorId
                        );
                } catch (SQLException e) {
                        e.printStackTrace();
                        return;
                }
        }
}

class RegisterException extends Exception {
        public enum ErrorCode {
                noError,
                registerCategoryNotFound,
                registerNumberExceeded,
                patientNotExist,
                sqlException,
                retryTimeExceeded,
        }

        ErrorCode error;

        RegisterException(String reason, ErrorCode err) {
                super(reason);
                error = err;
        }
}
```

Config.java

```
package hims;
```

```
public class Config {
	public static String NameTableDepartment = "department";
	public static String NameTableDoctor = "doctor";
	public static String NameTableCategoryRegister = "register_category";
	public static String NameTablePatient = "patient";
	public static String NameTableRegister = "register";

	public static String NameTableColumnDepartmentNumber = "depid";
	public static String NameTableColumnDepartmentName = "name";
	public static String NameTableColumnDepartmentPronounce = "py";

	public static String NameTableColumnDoctorNumber = "docid";
	public static String NameTableColumnDoctorDepartmentNumber = "depid";
	public static String NameTableColumnDoctorName = "name";
	public static String NameTableColumnDoctorPronounce = "py";
	public static String NameTableColumnDoctorPassword = "password";
	public static String NameTableColumnDoctorIsSpecialist = "speciallist";
	public static String NameTableColumnDoctorLastLogin = "last_login_datetime";

	public static String NameTableColumnCategoryRegisterNumber = "catid";
	public static String NameTableColumnCategoryRegisterName = "name";
	public static String NameTableColumnCategoryRegisterPronounce = "py";
	public static String NameTableColumnCategoryRegisterDepartment = "depid";
	public static String NameTableColumnCategoryRegisterIsSpecialist =
"speciallist";
	public static String NameTableColumnCategoryRegisterMaxRegisterNumber =
"max_reg_number";
	public static String NameTableColumnCategoryRegisterFee = "reg_fee";

	public static String NameTableColumnPatientNumber = "pid";
	public static String NameTableColumnPatientName = "name";
	public static String NameTableColumnPatientPassword = "password";
	public static String NameTableColumnPatientBalance = "balance";
	public static String NameTableColumnPatientLastLogin = "last_login_datetime";

	public static String NameTableColumnRegisterNumber = "reg_id";
	public static String NameTableColumnRegisterCategoryNumber = "catid";
	public static String NameTableColumnRegisterDoctorNumber = "docid";
	public static String NameTableColumnRegisterPatientNumber = "pid";
	public static String NameTableColumnRegisterCurrentRegisterCount =
"current_reg_count";
	public static String NameTableColumnRegisterFee = "reg_fee";
	public static String NameTableColumnRegisterDateTime = "reg_datetime";
}
```

# 参考文献

1 [美] Cay S. Horstmann 著，周立新，叶乃文，邝劲筠，杜永萍译. Java 核心技术 卷 I 基础知识（原书第 10 版）. 北京：机械工业出版社，2018.