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：计卓 1501

学　　号：U201514898

姓　　名：胡思勖

指导教师：辜希武

2018 年 5 月 10 日

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。

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

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

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

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

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。

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。

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。

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

对于病人的操作界面而言，需要有题目要求的挂号功能，包括人性化的过滤功能，需要足够的提示信息用于提示当前的操作状态，此外，还需要题目要求中没有提到的对于余额的操作功能；对于医生的操作界面而言，除了需要题目要求的两种统计功能外，还需要额外的过滤功能以供更于便捷的查询。此外，还需要一个统一的登录界面以供医生和病人登录。

对于程序的功能而言，不仅需要程序具有健壮性，在发生错误的时候不能崩溃，而且要求界面友好，支持多种操作方式，相应迅速。

二、 系统设计

1. 概要设计

整个系统分为 2 个部分：程序部分以及数据库部分。数据库部分用于存储数据并同时服

务多个客户端，而程序部分则负责相应用户的输入，与数据库沟通、处理数据并返回处理的结果。而程序部分总体上又分为 4 个模块：登录界面、医生操作模块，病人操作模块以及数据库连接器。总体的模块图如图 1 所示。其中，登录界面用于负责检查用户的登录信息是否正确，并负责唤醒医生操作界面或病人操作界面，而医生操作界面或病人操作界面则调用数据库模块与数据库进行沟通，处理数据并返回结果。

此外，程序入口也被封装为一个较小的 main 模块，用于执行包括加载数据库驱动、连接数据库、启动化图形界面引擎在内的初始化操作，以及实现唤醒登录界面的功能。如果不能成功连接数据库则此模块进行相应的处理并退出程序。由于此模块较小，且功能较为简单，因此不列入主要模块中。

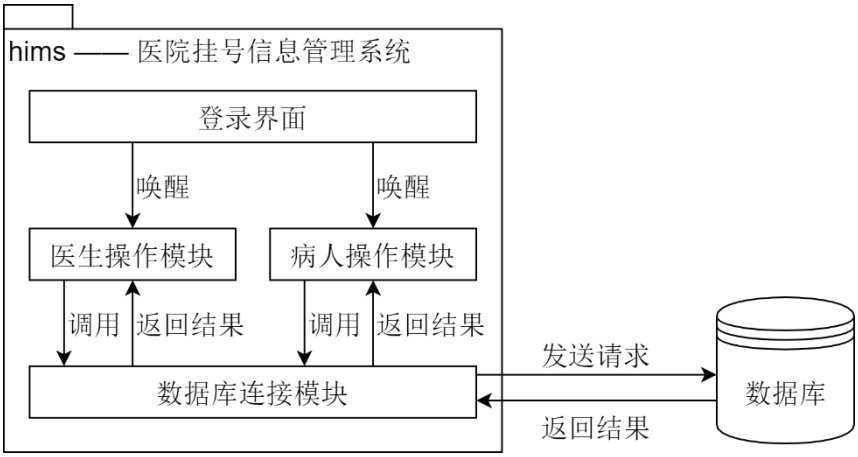

图 1 总体模块图

程序的总体状态转移图如图 2 所示，首先进入登录界面并等待用户输入登录信息，然后通过查询数据库判断登录信息是否匹配，如果匹配则登录，否则提示错误并等待用户重新输入登录信息。登录成功后通过判断用户点击的是医生登录按钮还是病人登录按钮来判断加载医生登录界面还是病人登录界面。此后进入图形界面引擎控制的事件循环。当有事件到来时（如用户输入、点击等）处理事件、显示返回结果并继续等待下一个事件。所有的状态都是可逆的，也就是说用户可以通过退出按钮来回到上一个界面，以此来增加界面的可操作性。

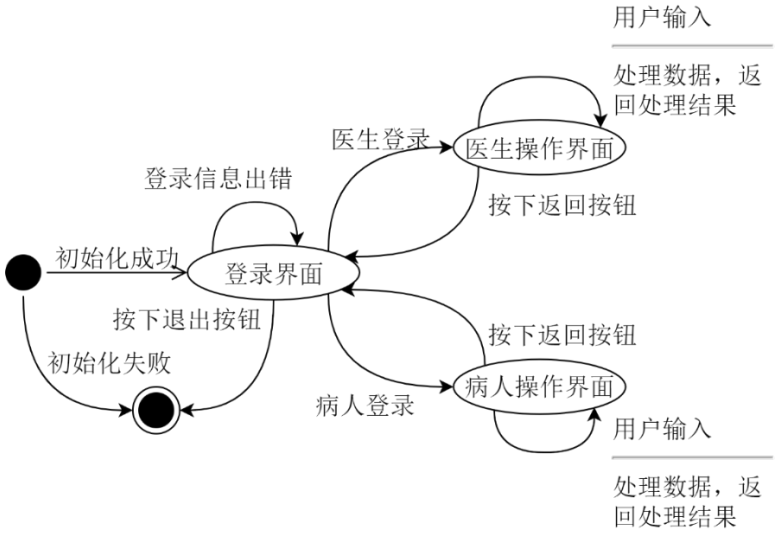

图 2 总体状态转移图

2. 详细设计

1) 登录界面设计

登录界面的设计较为简单，其功能为检测用户输入的登录信息是否与数据库中的登录信息相同。其应该包含一个用户名框，一个密码框，一个医生登录按钮、一个病人登录按钮以及一个退出按钮。由于图形化界面一般采用事件驱动，因此在按下按钮时进行对于用户输入的处理。按下医生登录按钮以及病人登录按钮的流程图如图 3 所示。如果点击的是医生登录按钮或病人登录按钮，则首先判断用户名和密码是否为空。由于设计不允许出现空的用户名和密码，因此提前进行这一步判断有助于减少用户的等待时间以及无效的数据库访问，如果不为空则通过事先建立好的连接在数据库中查询，并通过查询结果进行比对：如果不存在此用户（查询结果为空）或密码比对错误，则做出相应的提示并禁止用户登录，如果用户名和密码均正确则加载对应的登录界面。如果点击的是退出按钮，则执行清理工作并退出界面。值得注意的是，由于医生登录界面与病人登录界面为同一个界面，因此第一步的输入信息不为空的检查可以提取到一个函数中进行，在本程序中 validateUserNameAndPassword 函数执行这一操作。

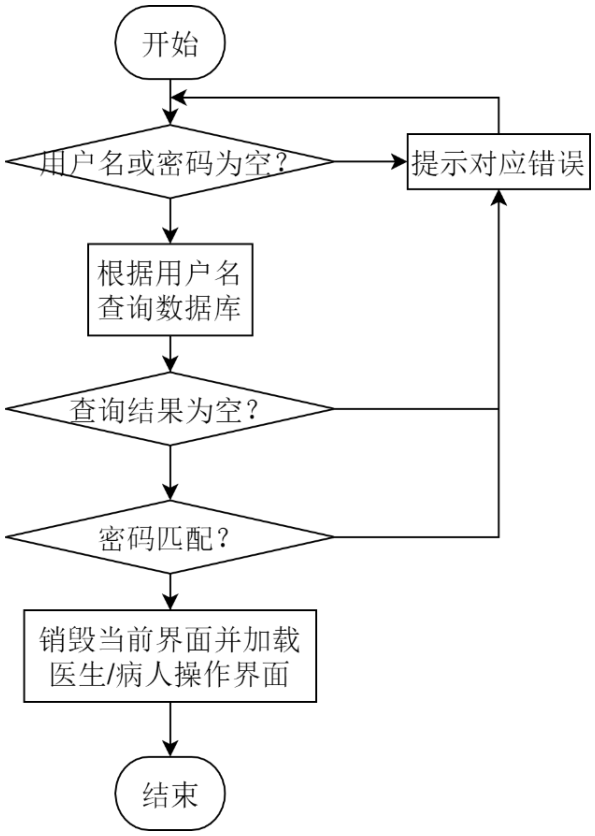

图 3 医生/病人登录流程图

此外，为了提高用户体验，在提示错误重新输入时需要清理掉上次提示的错误，而这些函数绑定在两个输入框的输入触发器上，这一绑定由此类的 initialize 方法完成。按照 JavaFX 的规范，initialize 方法会在界面初始化时被调用。由于这些函数主要对于 FXML 的 StyleSheet 进行修改，与主要逻辑相比重要性较低，因此不做赘述，详见源代码部分。

2) 数据库连接器设计

数据库连接器是整个程序的基础，它位于层次结构的最底层并被医生操作模块或病人操作模块所调用。这一模块在整个界面被初始化之前初始化，又由于没有它整个程序将无法运作，因此如数据库连接器不能正确被初始化，则程序将拒绝运行并退出。

数据库连接器作为一个单例，为医生操作模块和病人操作模块提供必要的服务。这样可以简化上层的逻辑，将部分逻辑移动到底层，从而降低了整个系统的耦合度，同时便于代码的修改。虽然数据库连接器在全局作为一个单例对象存在，但由于需要考虑到多个客户端同时连接同一个数据库的情况，依然要考虑数据库层面的并发安全性问题。

考虑到一个数据库仅供给一个人使用，为了简便起见，此类中仅维持一个连接实例，所有的 statement 均通过这一个连接进行。数据库对上层提供的服务（即公有方法）以及对应的执行语句如表 1 所示（以略去参数类型）。此外，还提供了 connectDataBase 方法供初始化使用以及 getInstance 方法供获得实例使用。

表 1 数据库连接器提供的服务以及对应的 sql 语句

方法 (参数)	说明	执行的 Sql 语句
getWholeTable (table)	获得表 table 中的全部内容	`SELECT *` ` FROM <table>`

getPatientPassword (number)	获得编号为 number 的病人的密码	`SELECT password`
		`FROM patient`
		` WHERE pid=<number>`

getPatientInfo (number)	获得编号为 number 的病人的全部信息	`SELECT *`
		`FROM patient`
		` WHERE pid=<number>`

getDoctorInfo (number)	获得编号为 number 的医生的全部信息	`SELECT *`
		`FROM doctor`
		`WHERE docid=<number>`

getRegisterForDoctor (number, start, end)	获得起止时间分别为 start 和 end 的有关医生编号 number 的全部挂号信息	`SELECT reg.reg_id,pat.name,`
		`reg.reg_datetime,cat.speciallis`
		`t`
		`FROM (`
		`  SELECT`
		`reg_id,pid,reg_datetime,catid`
		`  FROM register`
		`  WHERE docid=<number> AND`
		`reg_datetime>=<start> AND`
		`reg_datetime<=<end>`
		`) as reg inner join (`
		`  SELECT pid,name`
		`  FROM patient`

		```) as pat on reg.pid=pat.pid inner join (SELECT reg_id, specialist FROM register_category) as cat on reg.catid=cat.catid ```
getIncomeInfo (start, end)	获得起止时间分别为 start 和 end 的所有医生的收入信息	``` SELECT dep.name as depname, reg.docid, doc.name as docname, cat.specialist, reg.current_reg_fount, SUM(reg.fee) as sum FROM (SELECT * FROM register WHERE reg_datetime>=<start> AND reg_datetime<=<end>) as reg inner join (SELECT docid,name,depid FROM doctor) as doc on reg.docid=doc.docid inner join(SELECT depid,name FROM department) as dep on doc.depid=dep.depid inner join(SELECT reg_id,specialist FROM register_category) as cat on reg.catid=cat.catid GROUP BY reg.docid, cat.specialist ```
updatePatientLoginTime (number, time)	更新编号为 number 病人的最近登录时间为 time	``` UPDATE patient SET last_login_datetime=<time> WHERE pid=<number> ```
updateDoctorLoginTime (number, time)	更新编号为 number 医生的最近登录时间为 time	``` UPDATE doctor SET last_login_datetime=<time> WHERE docid=<number> ```
tryRegister	根据所给参数尝试挂号（较为复杂，见下）	

tryRegister 方法由于不仅仅是执行一个简单的任务并返回结果而显得较为复杂：这个函数是为了将上层的部分逻辑转移到下层以降低系统耦合度而产生的。其流程图如图 4 所示。由于这一方法需要分为多步执行，并且需要考虑并发安全性，因此需要启动一个 transaction，在图中流程中任意一个地方发生错误则直接报错并回滚，回滚后不会对数据库造成任何影响。由于需要挂号编号是单调递增且连续的，因此 transaction 的隔离级别设置为 repeatable read（可重复读）。执行挂号的 sql 语句前执行的判断为当前挂号数是否超过最大挂号数，而执行之后的判断为两次对于是否更新余额的判断：一次是判断是否需要从余额扣款，一次判断是否将找零存入余额。在所有语句得以正确执行后提交 transaction 以持久化更改。

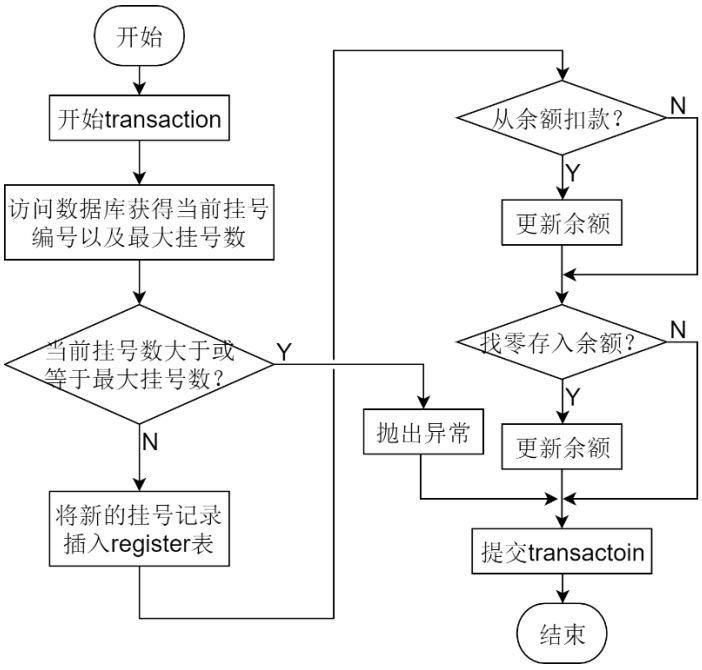

图 4 tryRegister 方法流程图

由于 tryRegister 返回的是成功挂号时的号码，占用了错误信息的返回渠道，因此为这一方法定义了一个异常 RegisterException 类来返回错误信息，其中包含错误代码可以指示出错的原因。

3) 病人操作界面设计

病人操作界面的设计是整个程序中最为复杂的部分，其大部分代码均为界面更改代码。界面包含 4 个输入/下拉框、一个滑块以及 2 个复选框，输入框分别用于选择科室名称、医生姓名、号种类别以及号种名称，滑动条用于选择交款金额，两个复选框用于选择是否使用余额付款以及是否将找零存入余额。此外，界面上还包含需要显示的信息，包括应缴金额、找零金额以及挂号号码、还包含挂号按钮和退出按钮用于开始执行挂号逻辑以及退出当前界面。用于搭建界面的逻辑此处不再赘述，下面主要描述核心部分较为重要的复选框过滤方法以及当挂号按钮按下后所执行的逻辑。

由于程序需要良好的用户体验，下拉框需要同时支持鼠标输入以及键盘输入，而当一个下拉框选择了一项内容后，其余的下拉框也应该更新其可选内容以匹配当前内容，这就涉及到过滤策略的问题：如何进行过滤以及在何时进行过滤。经过一番考虑与实践后决定采用如

下的过滤策略：为每一个下拉框定义一个原始列表与一个当前列表，原始列表用于存放此下拉框在没有其它限制的情况下所有的可选值，而当前列表用于存放经过过滤后下拉框的候选值。再为每一个下拉框定义 2 种操作：更改与提交。鼠标在候选值上滑动、键盘正在进行输入的时候下拉框为更改状态，而鼠标点击后、键盘换行键按下后下拉框显示当前列表中的某一个值的状态为提交状态。

于是过滤策略为：进行更改操作时仅过滤当前列表并将过滤后的内容加入下拉框的候选值中，而提交操作后对于所有的当前列表重新从原始列表进行按序更新，扩展到一般情况，假设共有 N 个列表，在提交任意一个下拉框 m 后对于当前列表 1~m-1, m+1~N 依次进行更新，而对于列表 k 的更新过程为：依次使用 1 ~ k-1、k+1 ~ N 的下拉框的已选择值以及此下拉框选择的内容类型对于第 k 个列表的语义限制对第 k 个列表进行更新。这一操作看似是具有时间复杂度$O(M^2)$的，而实际上进行了一定的优化后可以仅仅在 O(M)的时间内完成，其中 M 为平均原始列表长度（由于过程较为庞杂，此处不便于使用图形进行展示）。

这样设计的好处在于：能够在较短的时间内对于所有的列表进行动态的过滤而不会带来自定义的更新规则或带优先级的更新规则所带来的混乱，当用户选择下拉框 m 时 m 自动成为最高优先级（这也是合理的，因为用户当前的焦点在 m 处），并结合之前已选的内容对于所有下拉框进行过滤，在删除复选框内容时也不会造成其它更新策略可能带来的列表内容丢失。

在挂号按钮按下后首先进行挂号前检查：判断必要的下拉框是否已经有选择的值以及余额是否充足，然后调用数据库连接器中的 tryRegister 函数尝试挂号，并根据返回的结果显示相应的挂号号码或者错误信息。

4) 医生操作界面设计

医生操作界面则较为简单，其以查询、统计功能为主，并且统计功能可以集成在 sql 语句中完成，因此代码逻辑简短。值得注意的是需要使用 TreeTableColumn 来存储各列的值，并使用 DateTimeFormatter 对于时间格式进行转换，否则列表可能不能正常更新，时间也不能正常被数据库连接器读取。

医生操作界面界面包含一个挂号列表、一个收入列表，均可以按照任意一栏进行排序，包含 2 个时间选择器以及 2 个日期选择器，用于过滤起止日期，在 2 个表中均能过滤，并有 2 个复选框能够快捷的选定今天或全部时间，此处的界面更新逻辑为：但两个复选框中的任意一个被按下另一个都会取消选择，并且时间选择器与日期选择器会被禁用。最后，界面上包含 2 个按钮，分别用于发送数据库请求并更新界面以及退出当前界面。

三、 软件开发

本程序以及测试程序的编写均在 Arch Linux x64 系统下完成，并分别在 Linux 和 Windows 下进行了测试，具体开发以及测试环境见表 2。

表 2 软件开发环境

项目	版本

开发操作系统	Arch Linux 2018 年 4 月更新
JDK (Linux)	OpenJDK 1.8.0_172, Java 8
集成开发环境	IntelliJ IDEA 2018.1.3
测试操作系统 1	Arch Linux 2018 年 5 月更新
JRE (Linux)	OpenJDK Runtime Environment 1.8.0_172-b11
数据库 (Linux)	MariaDB 10.3.7.r77.gee5124d714e
测试操作系统 2	Windows 10 Pro x64 Version1803
JRE (Windows)	Oracle Java SE Runtime 1.8.0_161-b2
数据库 (Windows)	MariaDB 10.3.7 x64

四、 软件测试

本软件测试在 Arch Linux 下进行。经测试，程序在 Windows 下的行为和运行结果与在 Arch Linux 下完全相同。

1) 登录测试

首先不启动数据库，在无数据库连接的情况下启动程序，程序直接报错退出，如图 5 所示，这是预期行为，因为不能要求用户手动连接数据库，因此只能报错退出。

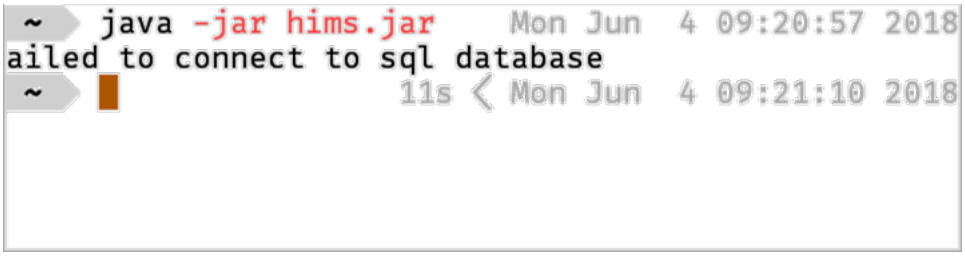

图 5 不能成功连接数据库

在数据库运作正常的情况下，启动后的界面如图 6 所示

图 6 登录界面

然后尝试不使用用户名、密码输入错误的用户名、密码登录，程序应该提示对应的错误，知己的程序输出如图 7 和图 8 所示。可以看出，与预期的结果相同。下方的显示区域分别显示“请输入用户名”、“请输入密码”、“用户不存在”以及“密码错误”。

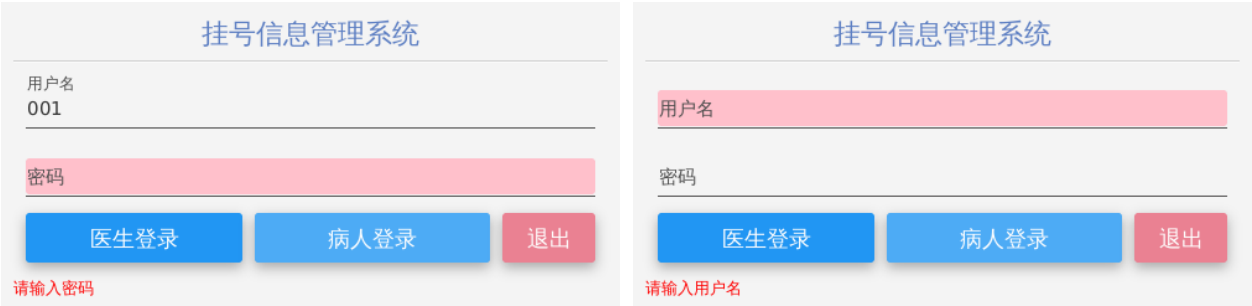

图 7 不输入用户名和密码的提示信息

图 8 用户名不存在或密码错误的提示信息

在使用用户名 003、密码 003003 并点击病人登录后，成功登录，登录后的界面如 所示。此处登录测试完成，进入病人操作界面测试阶段。

2) 病人操作界面测试

进入病人操作界面后如图 9 所示。其中所有的可交互控件均可同时使用鼠标和键盘进行操作。当号种类别以及号种名称被选择时，应缴金额以及找零金额会自动显示，并且找零金额会随着付款方式以及交款金额的变化而变化。当未满足挂号条件时，挂号按钮为未激活状态。

图 9 病人操作界面

接下来尝试进行挂号操作当余额低于应缴金额时，使用余额付款按钮为未激活状态，不可使用余额付款，否则可以选择使用余额付款或使用现金付款，当选择使用余额付款时，交款金额滑块为未激活状态以减少无效操作的可能性。正在使用现金挂号的界面以及挂号成功的界面分别如图 10 以及图 11 所示，在挂号成功后，底部状态栏显示了挂号成功的信息以及挂号号码。

图 10 正在使用现金挂号

图 11 挂号成功界面

为了测试程序的并发安全性，使用 IntelliJ IDEA 同时打开两个程序进行单步调试，调试结果显示无论两个程序以何种方式穿插挂号的操作都不会发生编号冲突或缺失的情况，但有可能造成某一个客户端不能挂号成功而发生数据库回滚。在这种情况下用户需要重新挂号。而当挂号数量达到限额时也会提示挂号失败，如图 12 所示，当多次点击挂号后，此号码达到了人数上限，因此在访问数据库的时候会抛出异常并禁止用户进行进一步的挂号操作。

此外，本程序还支持将找零存入余额以供下次使用。使用另一个号码进行挂号，并勾选将找零存入余额，挂号成功后余额有所增加，如图 13 所示。

图 12 挂号达到人数上限界面

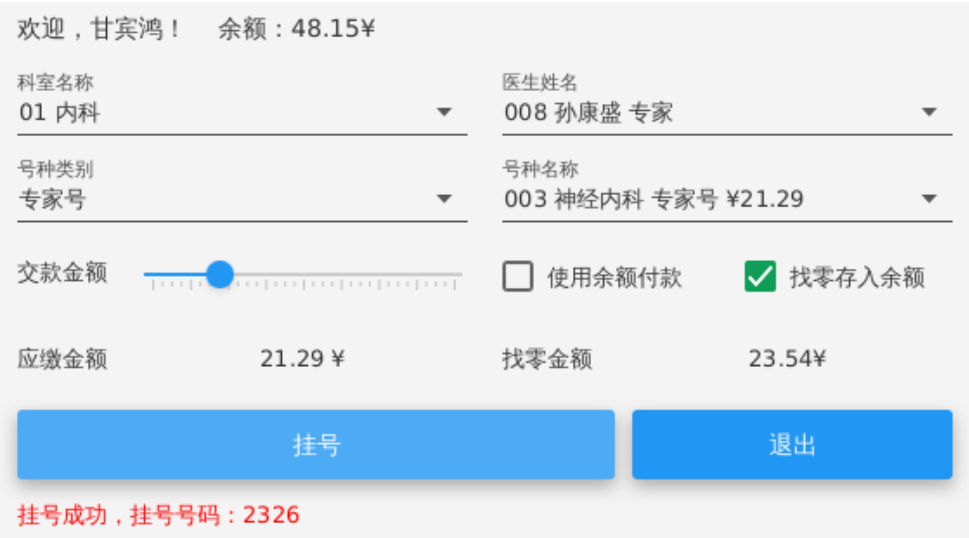

图 13 勾选找零存入余额后余额增加

在界面的友好性方面，所有的交互控件均能够同时支持鼠标和键盘操作，一个使用键盘通过输入拼音进行下拉框过滤的例子如图 14 所示。

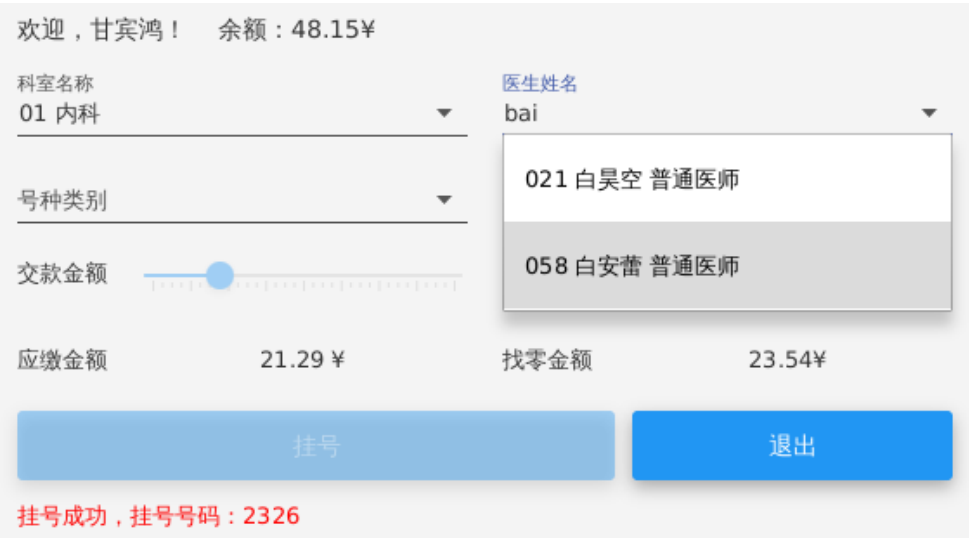

图 14 下拉框过滤支持拼音以及编号

3) 医生操作界面测试

退出病人操作界面后进入医生操作界面，图 15 展示了上面用于演示的 008 医生登录并查询所有挂号列表的操作界面。界面的下方为过滤器，可以自定义时间过滤，也可以使用“全部时间”和“今天”复选框进行快捷的过滤。使用这两个复选框的效果与手动设置时间的效果相同，但是免去了较为费时的选择操作。

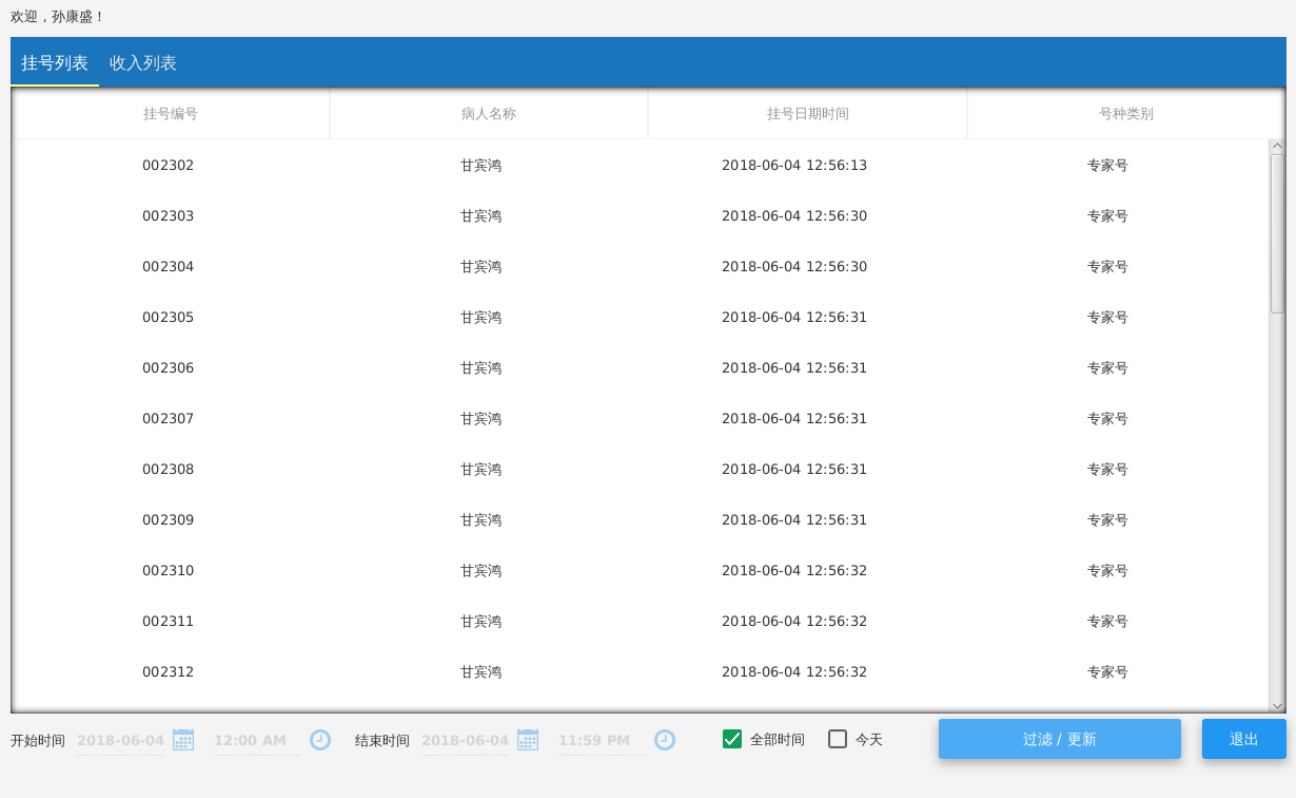

图 15 一个医生操作界面的示例 //TODO 替换

图 16 展示了手动输入时间进行过滤的效果，显示的值为上方病人测试中最后一次挂号的信息。可以看出与病人挂号时选择与返回的信息相同。

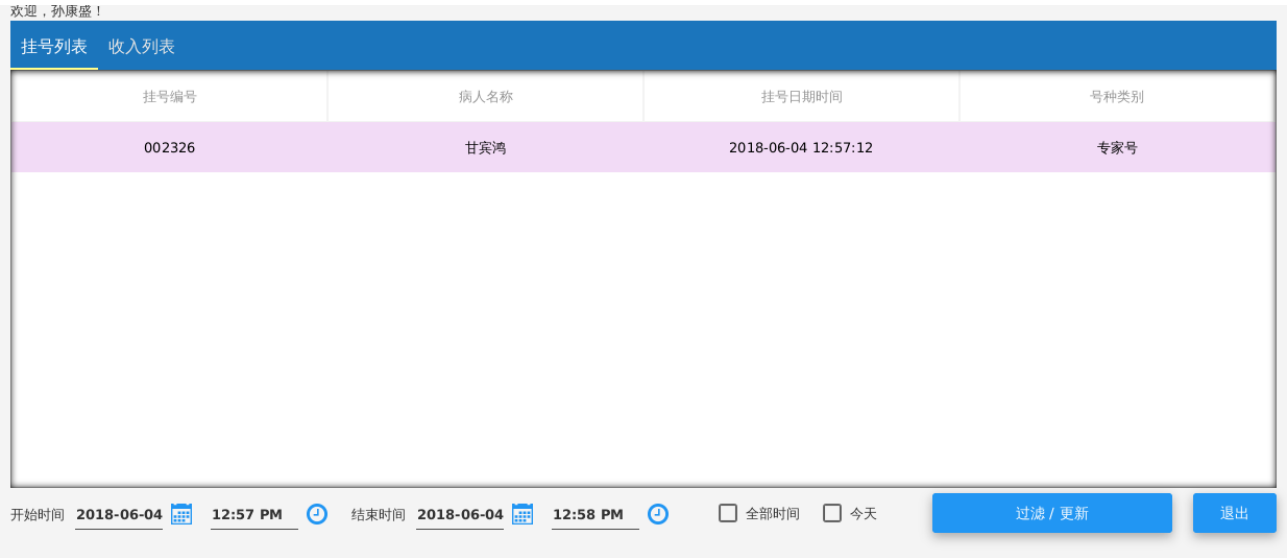

图 16 手动输入时间进行过滤

标签栏的另一个标签为收入列表，这一列表的过滤方式与挂号列表的过滤方式相同，只是显示的内容有所差异。

科室名称	医生编号	医生名称	号种类别	挂号人次	收入合计
妇产科	001	林鹏鲲	普通号	4	82.84
妇产科	001	林鹏鲲	专家号	1	244.01
精神心理科	002	金德元	普通号	2	79.48
精神心理科	002	金德元	专家号	3	176.21
生殖中心	003	邵弘济	普通号	1	120.01
生殖中心	003	邵弘济	专家号	5	203.37
烧伤科	004	熊向荣	普通号	3	62.38
烧伤科	004	熊向荣	专家号	3	130.70
肿瘤科	005	方成文	普通号	3	92.96
肿瘤科	005	方成文	专家号	4	373.04
皮肤美容	006	赖阳秋	普通号	6	87.13

图 17 收入列表在显示全部时间的信息

值得注意的是，无论是挂号列表还是收入列表，各栏的相对位置都可以移动，并且可以按照任意一栏排序。图 17 即为将挂号人次移动到第三栏并以之为基准进行排序的结果。

科室名称	医生编号	挂号人次	医生名称	号种类别	收入合计
妇产科	032	9	赵和璧	普通号	27.25
内科	026	9	万鹏鹍	专家号	322.71
中医科	115	9	阎悦宜	普通号	50.00
口腔科	015	8	漕瀚玥	专家号	158.53
眼科	140	8	黄胤文	专家号	315.91
精神心理科	040	8	周康顺	普通号	44.82
中医科	143	8	蒋幻香	普通号	77.34
儿科	083	7	龚念梦	专家号	233.26
眼科	128	7	武绮纶	专家号	234.67
眼科	054	7	戴玉英	专家号	264.62
儿科	116	7	罗郁蓝	专家号	332.16

图 18 收入列表样例

五、 特点与不足

1. 技术特点

- 病人操作界面使用了效率较高的过滤算法，以多个维度进行过滤，提高了程序的可用性和便捷性。
- 采用了移动端的控件风格构造了一个更为人性化的操作界面。
- 程序考虑到了多种异常以及并发的安全问题，健壮性强，并且程序效率较高。
- 将程序分解为多个模块，耦合度低、可扩展性强，易于修改。
- 编译时将所有的依赖打入一个 jar 包中，不存在依赖问题，全平台通用， 图 19 为本程序在 Window10 下的运行效果。可以看出，除了字体有些微的差别外其余并无不同。

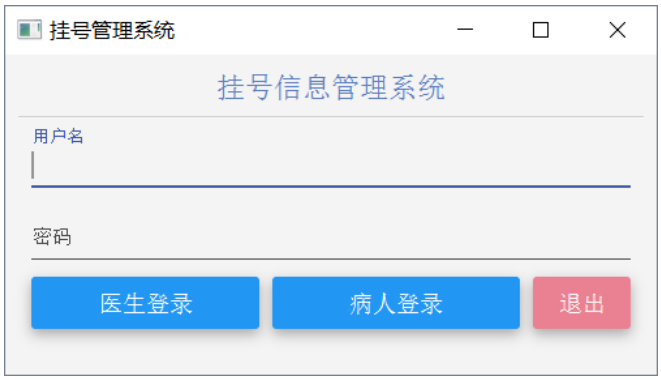

图 19 Windows 下的登录界面

2. 不足和改进的建议

- 由于数据库定义密码长度限制，所有的密码为明文存储，安全性较低。可以考虑将密码改为 hash 加密后的密码存储。
- 下拉框过滤算法以及医生界面的更新算法可以改为增量更新，并多加上一级 cache，这样可以减少数据库操作，并提升性能。
- 可以添加更多的统计功能供不同角度的查询使用。
- 可将 sql 用户名密码等全局变量提取到配置文件中，避免硬编码。

六、 过程和体会

1. 遇到的主要问题和解决方法

在医生见面设计中发生了统计表格不能自动更新的问题，通过阅读文档发现 TreeTable 类需要设置 CellValueFactory 才能实现自动更新，遂修改程序使之得以正常运行。在病人界面的设计时遇到了下拉框过滤输入时过滤算法失效的问题，经过仔细的检查后、发现原因在于应用过滤的优先级混乱造成了下拉框备选项丢失的错误，经过一番思考后想出了上面详细设计所述的过滤算法，从而从根本上解决了这一问题。此外，IntelliJ IDEA 的 FXML 编辑工具缺失部分功能，使得 FXML 文件不能指定控制类，并且容易导致 IDE 崩溃退出，这一问题可以使用独立的 SceneBuilder 解决。

2. 课程设计的体会

通过此次课程设计，我最大的收获是学习了如何使用 JavaFX 搭建一个现代化的、友好的界面，以及如何让 Java 程序与数据库协作来完成复杂的功能。对于第一次接触 JavaFX 库的我而言，这是一个挑战，但是经过近一个星期的不懈努力，我深入的了解了 JavaFX 的运作方式，并使用其构造了一个较大的项目，在这个过程中我学到了不少有用的知识。不仅如此，在这一综合性较强的实验中我还顺带复习了有关 Java 的语言特性、有关数据库的知识以及许多软件工程的经验与知识，这对于今后的学习生活奠定了不小的基础。

七、 源码和说明

1. 文件清单及其功能说明

文件（夹）结构及其说明如下：

```
lab2                                          项目根目录
├── README.md                                 说明文件
├── lib                                       第三方库文件夹
│   ├── jfoenix-8.0.1.jar                     JFoenix第三方库
│   └── mysql-connector-java-5.1.46-bin.jar   MySQL第三方库
├── out                                       编译输出文件夹
│   └── hims.jar                              编译输出Jar包
├── src                                       源文件夹
│   └── hims                                  源文件夹（包级）
│       ├── Login.fxml                        登录界面FXML文件
│       ├── Doctor.fxml                       医生界面FXML文件
│       ├── Patient.fxml                      病人界面FXML文件
│       ├── LoginController.java              登录界面控制类源码
│       ├── PatientController.java            病人界面控制类源码
│       ├── DoctorController.java             医生界面控制类源码
│       ├── DBConnector.java                  数据库连接器源码
│       ├── Config.java                       配置文件
│       ├── Main.java                         主函数所在类源码
│       └── Main.css                          界面使用的层叠样式表文件
└── sample_data.sql                           数据库样例数据文件
```

2. 用户使用说明书

要使用本程序，须按照下列步骤进行环境搭建：

- 前提条件：
  - 操作系统：Windows/Linux/OS X
  - Java 运行环境：Jre 1.8.0_161 或其兼容版本
  - 数据库：MariaDB 10.3.7 或其兼容版本或 MySQL 对应的兼容版本

- 样例数据导入
  - 启动数据库服务
  - 使用 root 登录进入数据库，创建名为 java_lab2 的数据库

```
CREATE DATABASE java_lab2;
```

  - 创建名为 java 的用户，设置其密码为 javajava，并授予其在 java_lab2 上的所有权限。

```
GRANT ALL ON java_lab2.* TO 'java'@'localhost' IDENTIFIED BY 'javajava';
FLUSH PREVILEGES;
```

  - 使用 java 用户登录 java_lab2 数据库，并运行 source sample_data.sql 来初始化数据。
  - 由于数据库名称、用户名以及密码均硬编码在程序中，因此上述信息不可更改。
- 使启动数据库服务维持启动状态
- 运行 out 文件夹下的 hims.jar，可以在命令行中运行 java -jar hims.jar，也可直接在文件管理器中点击（需要 java 在 PATH 环境变量中并正确的设置 MIME TYPE）。

3. 源代码

此处略去所有 FXML 文件和 CSS 文件，只列出 Java 文件。

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{
        // intialize connector
        try {

DBConnector.getInstance().connectDataBase("localhost", 3306,
"java_lab2", "java", "javajava");
        } catch (SQLException e) {
            System.err.println("failed to connect to sql
database");
            System.exit(0);
        }

        //Parent root =
FXMLLoader.load(getClass().getResource("Patient.fxml"));
```

```
        Parent root =
FXMLLoader.load(getClass().getResource("Login.fxml"));
        primaryStage.setTitle("挂号管理系统");
        //primaryStage.setScene(new Scene(root, 600, 200));
        primaryStage.setMinWidth(400);
        primaryStage.setMinHeight(190);
        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").t
oExternalForm());
            FXRobotHelper.getStages().get(0).setScene(scene);
        } catch (SQLException e){
            // progrom shouldn't came here
            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).equal
s(inputPassword.getText())) {
                labelStatus.setText("密码错误");
                labelStatus.setStyle("-fx-text-fill: red;");
                return;
            }

            // fill info and login to patient page
            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").t
oExternalForm());
            FXRobotHelper.getStages().get(0).setScene(scene);
        } catch (SQLException e) {
            // progrom shouldn't came here
            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);
    }
}
```

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&characte
rEncoding=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 void disconnectDataBase() throws SQLException {
        connection.close();
    }

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

```
    /**
     * get patient's password by patient id
     * @param number
     * @return null if patient not found, else the password
     */
    public String getPatientPassword(String number) {
        try {
            ResultSet res =  statement.executeQuery(
                    "select " +
Config.NameTableColumnPatientPassword +
                            " from " + Config.NameTablePatient +
                            " where " +
Config.NameTableColumnPatientNumber + "=" + number);
            if (!res.next())
                return null;
            return
res.getString(Config.NameTableColumnPatientPassword);
        } catch (SQLException e){
            e.printStackTrace();
            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;
        }
    }
```

```
    /**
     * try adding register info to the database
     * @param registerCategoryNumber register category number
     * @param doctorNumber doctor number
     * @param patientNumber patient number
     * @param registerFee register fee
     * @return register number if registeration is successful,
null otherwise.
     * @throws RegisterException if register failed
     */
    public int tryRegister(
            String registerCategoryNumber,
            String doctorNumber,
            String patientNumber,
            Double registerFee,
            boolean deductFromBalance,
            Double addToBalance) throws RegisterException {
        try{
            // decide the register id
            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.NameTableColumnRegiste
rNumber)) + 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
);

            // decide patient id
            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);

            // decide if exceeded the max register count
            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.NameTableColumnCategoryRegisterMaxRegisterN
umber);

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

```
RegisterException.ErrorCode.registerNumberExceeded);
            }

            // try insert
            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
                    )
            );

            // deduct from balance
            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)
```

```
                );
            }

            // all successfulHH
            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;
    }
}
```

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(DateTime
Formatter.ofPattern("yyyy-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));
        }
    }

    // set by LoginController
    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，%s！",
doctorName));

        // set two date converter (formatter)
        pickerDateStart.setConverter(new DateConverter());
        pickerDateEnd.setConverter(new DateConverter());
        // default to current date
        pickerDateStart.setValue(LocalDate.now());
        pickerDateEnd.setValue(LocalDate.now());

        // set time selector to 24h
        pickerTimeStart.setIs24HourView(true);
        pickerTimeEnd.setIs24HourView(true);
        // default to 00:00 to 23:59
        pickerTimeStart.setValue(LocalTime.MIN);
        pickerTimeEnd.setValue(LocalTime.MAX);

        // initiailze register list
        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);

        // template
        // listRegister.add(new Register("a", "b", new
Timestamp(1000),true));
        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("yy
yy-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.NameTableColumnCategoryRegisterIsSpecia
list)
                    ));
                }
            } 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("yy
yy-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.NameTableColumnCategoryRegisterIsSpecia
list),

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.NameTableColumnCategoryRegisterDepartmen
t);
        isSpecialist =
result.getBoolean(Config.NameTableColumnCategoryRegisterIsSpecia
list);
        maxNumber =
result.getInt(Config.NameTableColumnCategoryRegisterMaxRegisterN
umber);
        fee =
result.getFloat(Config.NameTableColumnCategoryRegisterFee);
    }
}

public class PatientController {
    // basic patient info, initialized by LoginController
    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 JFXSlider sliderPay;
    @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;

    // data list
    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();

    /**
     * @brief update data for one editable combobox
     * @param tableName name of database table related to the
combobox
     * @param list list to update/initialize
     * @param clazz dumb thing, see
https://stackoverflow.com/questions/11404086/how-could-i-
initialize-a-generic-array
     * @return if update is successful
     */
    private <ItemType extends ListItem> boolean
updateOneSetOfData(
            String tableName,
            ObservableList<ItemType> list,
            Class<ItemType> clazz) {

        // get entire table from database
        ResultSet result =
DBConnector.getInstance().getWholeTable(tableName);

        if (result != null) {
            ObservableList<ItemType> tempList =
FXCollections.observableArrayList();
            try {
                // iterate all entries in table
                while (result.next()) {
                    // get a entry and convert it to ListItem
                    ItemType item = clazz.newInstance();
```

```
                    item.fromSqlResult(result);
                    // add to table
                    tempList.add(item);
                }
            } catch (Exception e) {
                e.printStackTrace();
                System.exit(-1);
            }

            // if everything is correct, copy them into result
            list.clear();
            list.addAll(tempList);
            return true;
        }
        return true;
    }

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

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

        updateOneSetOfData(
                Config.NameTableCategoryRegister,
                listNameRegister,
                ListItemNameRegister.class
        );

        // initialize/update register Type info
        // note that this part must be placed under the update
of catetgory register
        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 && sliderPay.getValue() >
listNameRegisterFiltered.get(index).fee) {
            labelRefund.setText(String.format("%.2f¥",
sliderPay.getValue() -
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){
            // fore to use cash
            checkBoxUseBalance.setSelected(false);
            sliderPay.setDisable(false);
            checkBoxUseBalance.setText("余额不足");
            checkBoxUseBalance.setDisable(true);
        } else {
            // prefer to use balance
            checkBoxUseBalance.setDisable(false);
            checkBoxUseBalance.setText("使用余额付款");
            checkBoxUseBalance.setSelected(true);
            sliderPay.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() &&
sliderPay.getValue() >=
listNameRegisterFiltered.get(index).fee)) ) {
            buttonRegister.setDisable(false);
        }
    }

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

    private void updateUserDisplayInfo() {
        labelWelcome.setText(String.format("欢迎，%s！     余
额：%.2f¥", patientName, patientBalance));
    }

    @FXML
    public void initialize() {
        updateUserDisplayInfo();
        // initialize datas
        updateData();

        // initialize combobox datas
```

```
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();

        // re-filter content on key typed

inputNameDepartment.getEditor().setOnKeyReleased(keyEvent -> {
            // pass up/down and enter keys
            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
-> {
            // pass up/down and enter keys
            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
-> {
            // pass up/down and enter keys
```

```
            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
-> {
            // pass up/down and enter keys
            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 sliderPayDragged(){
        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() &&
sliderPay.getValue() >=
listNameRegisterFiltered.get(index).fee)))) {
            // program should not run here
            statusError("缴费金额不足或余额不足");
            return;
        }

        // wait until sql response
        disableEverything();
```

```
        TryRegisterService service = new TryRegisterService(

listNameRegisterFiltered.get(inputNameRegister.getSelectionModel
().getSelectedIndex()).number,

listNameDoctorFiltered.get(inputNameDoctor.getSelectionModel().
getSelectedIndex()).number,
                patientNumber,

listNameRegisterFiltered.get(inputNameRegister.getSelectionModel
().getSelectedIndex()).fee,
                checkBoxUseBalance.isSelected(),
                ( (!checkBoxUseBalance.isSelected() &&
checkBoxAddToBalance.isSelected() ) ?
                        sliderPay.getValue() -
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();

        // filter Department Name
        for (ListItemNameDepartment listItemNameDepartment :
listNameDepartment) {
            if
(listItemNameDepartment.toString().contains(inputNameDepartment
.getEditor().getText().trim()) ||

listItemNameDepartment.getPronounce().contains(inputNameDepartm
ent.getEditor().getText().trim())) {

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

        // filter again according to doctor
        if ((index =
inputNameDoctor.getSelectionModel().getSelectedIndex()) != -1)
{
            for (ListItemNameDepartment department : list0)
```

```
                if
(department.number.equals(listNameDoctorFiltered.get(index).dep
artmentNumber))
                    list1.add(department);
            list0 = list1;
        }

        // add to filtered list and combobox
        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.getEdito
r().getText().trim()))
                isCurrentInputLegal = true;
            if(previousKey.equals(department.number))
                newSelection = counter;
            ++counter;
        }

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

inputNameDepartment.getSelectionModel().clearAndSelect(newSelec
tion);

inputNameDepartment.getEditor().setText(inputNameDepartment.get
Items().get(newSelection));
            }
        }
    }

    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();

        // filter Doctor Name
        for (ListItemNameDoctor listItemNameDoctor :
listNameDoctor)
            if
(listItemNameDoctor.toString().contains(inputNameDoctor.getEdit
or().getText().trim()) ||

listItemNameDoctor.getPronounce().contains(inputNameDoctor.getE
ditor().getText().trim()))
                list0.add(listItemNameDoctor);

        // filter by department
        if ((index =
inputNameDepartment.getSelectionModel().getSelectedIndex() )!=
-1) {
            for (ListItemNameDoctor listItemNameDoctor : list0)
                if
(listItemNameDoctor.departmentNumber.equals(listNameDepartmentF
iltered.get(index).number))
                    list1.add(listItemNameDoctor);
            list0 = list1;
        }

        // filter by register type
        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;
        }
```

```
        // filter by register name
        list1 = FXCollections.observableArrayList();
        if ((index =
inputNameRegister.getSelectionModel().getSelectedIndex()) != -
1) {
            for (ListItemNameDoctor doctor : list0)
                if
(doctor.departmentNumber.equals(listNameRegisterFiltered.get(in
dex).department))
                    list1.add(doctor);
            list0 = list1;
        }

        // add to filtered list and combobox
        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().getTe
xt().trim()) ||

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

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

inputNameDoctor.getSelectionModel().clearAndSelect(counter);

inputNameDoctor.getEditor().setText(inputNameDoctor.getItems().
get(newSelection));
            }
```

```
        }
    }

    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();

        // filter Register Type
        for (ListItemTypeRegister listItemTypeRegister :
listTypeRegister)
            if
(listItemTypeRegister.toString().contains(inputTypeRegister.get
Editor().getText().trim()) ||

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

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

        // filter by register name
        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;
        }

        // add to filtered list and combobox list
        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().g
etText().trim()) ||

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

        // delete illegal input
        if(!withoutSelect) {
            if (!isCurrentInputLegal)
                inputTypeRegister.getEditor().clear();
            if (newSelection != -1) {

inputTypeRegister.getSelectionModel().clearAndSelect(newSelecti
on);

inputTypeRegister.getEditor().setText(inputTypeRegister.getItem
s().get(newSelection));
            }
        }
    }
```

```
    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();

        // filter Register Name
        for (ListItemNameRegister listItemNameRegister :
listNameRegister)
            if
(listItemNameRegister.toString().contains(inputNameRegister.get
Editor().getText().trim()) ||

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

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

        // filter by doctor name
        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;
        }

        // filter by register type
        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;
        }

        // add to filtered list and combobox list
        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.g
etEditor().getText().trim()) ||

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

        // delete illegal input
```

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

inputNameRegister.getSelectionModel().clearAndSelect(newSelecti
on);

inputNameRegister.getEditor().setText(inputNameRegister.getItem
s().get(newSelection));
            }
        }
    }

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

    @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;

    // return value
    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 {
                        // try register
                        registerNumber =
DBConnector.getInstance().tryRegister(
                                registerCategoryNumber,
                                doctorNumber,
                                patientNumber,
                                registerFee,
                                deductFromBalance,
                                addToBalance);
                    } catch (RegisterException e) {
                        // retry on fail
                        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;
                }
            };
        }
}
```