Министерство образования Республики Беларусь

### Учреждение образования «[Институт информационных технологий](https://www.google.by/url?sa=t&rct=j&q=&esrc=s&source=web&cd=3&sqi=2&ved=0CDcQFjAC&url=http%3A%2F%2Fwww.iit-bsuir.by%2F&ei=unOvUpDmGemB4gSZ3IGQDQ&usg=AFQjCNHEr4the3QhkSSjmxbzcNJBZi5-Tg&sig2=GxV3hq7_8t34Csduk0HElg&bvm=bv.57967247,d.bGE&cad=rja) Белорусского государственного университета информатики и радиоэлектроники»

Факультет компьютерных технологий

Кафедра информационных систем и технологий

Дисциплина: Современные платформы программирования

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту

на тему

«Ювелирная мастерская»

БГУИР КП 1-40 01 01 5810027 ПЗ

Студент: гр.581061 Фут Д.С.

Руководитель: Бакунова О.М.

Минск 2017

**СОДЕРЖАНИЕ**

Введение 3

1 Анализ предметной области 4

1.1 Анализ существующих аналогов 4

1.2 Постановка задачи 4

1.3 Входные данные 4

1.4 Выходные данные 5

2 Разработка программного средства 6

2.1 Разработка спецификаций требований 6

2.2 Разработка архитектуры программного средства 6

2.3 Обоснование выбора языка программирования 7

2.4 Разработка схемы программы 8

2.5 Программная реализация программного средства 8

3 Тестирование программного средства 10

4 Методика использования прорграммного средства 13

Заключение 21

Список использованных источников 22

Приложение А. Исходный код программы 23

**ВВЕДЕНИЕ**

Основной задачей этого курсового проекта является разработка программного средства для ювелирной мастерской. Данное программное средство должно помогать администраторам и менеджерам ювелирной мастерской контролировать процесс работы в мастерской.

Мастерская работает с определенными материалами (платина, золото, серебро, различные драгоценные камни и т.д.). При обращении в мастерскую потенциального клиента, менеджер определяется с тем, какое именно изделие необходимо клиенту. Все изготавливаемые изделия принадлежат к некоторому типу (серьги, кольца, броши, браслеты), бывают выполнены из определенного материала и имеют цену (включающую стоимость материалов и работы).

**1 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ**

**1.1 Анализ существующих аналогов**

На данный момент времени существует множество аналогов программных средств по автоматизации рабочих мест сотрудников.

К основным недостаткам большинства таких программ можно отнести то, что для каждого рабочего места требуются свои функции, в зависимости от потребностей заказчика, соответственно программа, подходящая одному, не всегда может подойти другому, исходя из вышеизложенного причина создания программного средства обусловлена индивидуальными требованиями заказчика и невозможностью найти аналог под такие требования.

**1.2 Постановка задачи**

Анализ литературы и предметной области показали, что реализуемое программное средство имеет большое количество аналогов, но каждый из которых имеет свои недостатки.

Основной целью является – разработка программного средства для автоматизации рабочего места администратора ювелирной мастерской.

В данном программном средстве необходимо реализовать:

* Учет клиентов мастерской. Для каждого клиента необходимо хранить: Имя, Фамилию, Отчество, Телефон, Дату регистрации в системе.
* Учет заказов. Для каждого заказа необходимо хранить: дату оформления заказа, цену, статус заказа и статус оплаты.
* Учет изделий. Для каждого изделия необходимо хранить: тип изделия, цену, статус и описание изделия.
* Учёт материалов. Для каждого материала необходимо хранить: название материала и цену за грамм.
* Учёт работников. Для каждого работника необходимо хранить: фио работника и телефона.
* Учёт пользователей системы. Для каждого пользователя необходимо хранить: логин, пароль и информацию о том, является ли пользователь администратором.

**1.3 Входные данные**

Входной информацией является:

* Информация о клиенте;
* Информация о заказе;
* Информация об изделии;
* Информация о пользователе;
* Информация о материале;
* Информация о работнике;

**1.4 Выходные данные**

* Список всех клиентов;
* Список заказов;
* Список изделий;
* Список материалов;
* Список работников;
* Список пользователей;

**2 РАЗРАБОТКА ПРОГРАММНОГО СРЕДСТВА**

**2.1 Разработка спецификаций требований**

Программа будет разрабатываться, и функционировать в операционной системе Windows. В версиях от Windows xp до Windows 10.

Windows — семейство проприетарных операционных систем корпорации Microsoft, ориентированных на применение графического интерфейса при управлении. Изначально Windows была всего лишь графической надстройкой для MS-DOS. По состоянию на август 2014 года под управлением операционных систем семейства Windows по данным ресурса Net Applications работает около 89% персональных компьютеров. Это и было основание выбора данной операционной системы как основной рабочей системы проекта.

Минимальные требования:

* + - процессор: Intel Pentium II 400 MHz или аналогичный;
    - оперативная память: 512 Mб;
    - видеокарта: объем 256 Mб;
    - операционная система: Windows XP;
    - устройства ввода: клавиатура, мышь;
    - устройство вывода: монитор.

**2.2 Разработка архитектуры программного средства**

Программное средство должно будет хранить все имеющуюся информацию в базе данных. Для хранения информации будет использоваться база данных использующая реляционную модель.

В реляционной СУБД все обрабатываемые данные представляются в виде плоских таблиц. Информация об объектах определенного вида представляется в табличном виде: в столбцах таблицы сосредоточены различные атрибуты объектов, а строки предназначены для сведения описаний всех атрибутов к отдельным экземплярам объектов.

В качестве самой СУБД будет использоваться MySQL.

Гибкость СУБД MySQL обеспечивается поддержкой большого количества типов таблиц: пользователи могут выбрать как таблицы типа MyISAM, поддерживающие полнотекстовый поиск, так и таблицы InnoDB, поддерживающие транзакции на уровне отдельных записей. Более того, СУБД MySQL поставляется со специальным типом таблиц EXAMPLE, демонстрирующим принципы создания новых типов таблиц. Благодаря открытой архитектуре и GPL-лицензированию, в СУБД MySQL постоянно появляются новые типы таблиц.

**2.3 Обоснование выбора языка программирования**

Java представляет собой язык программирования и платформу вычислений, которая была впервые выпущена Sun Microsystems в 1995 г.

Существует множество приложений и веб-сайтов, которые не работают при отсутствии установленной Java, и с каждым днем число таких веб-сайтов и приложений увеличивается. Java отличается быстротой, высоким уровнем защиты и надежностью. От портативных компьютеров до центров данных, от игровых консолей до суперкомпьютеров, используемых для научных разработок, от сотовых телефонов до сети Интернет.

Программы на Java могут быть транслированы в байт-код, выполняемый на виртуальной java-машине (JVM) — программе, обрабатывающей байт-код и передающей инструкции оборудованию, как интерпретатор, но с тем отличием, что байт-код, в отличие от текста, обрабатывается значительно быстрее.

В Java используются практически идентичные соглашения для объявления переменных, передачи параметров, операторов и для управления потоком выполнением кода. В Java добавлены все хорошие черты C++.

Три ключевых элемента объединились в технологии языка Java

Java предоставляет для широкого использования свои апплеты (applets) — небольшие, надежные, динамичные, не зависящие от платформы активные сетевые приложения, встраиваемые в страницы Web. Апплеты Java могут настраиваться и распространяться потребителям с такой же легкостью, как любые документы HTML

Java высвобождает мощь объектно-ориентированной разработки приложений, сочетая простой и знакомый синтаксис с надежной и удобной в работе средой разработки. Это позволяет широкому кругу программистов быстро создавать новые программы и новые апплеты

Java предоставляет программисту богатый набор классов объектов для ясного абстрагирования многих системных функций, используемых при работе с окнами, сетью и для ввода-вывода. Ключевая черта этих классов заключается в том, что они обеспечивают создание независимых от используемой платформы абстракций для широкого спектра системных интерфейсов

**2.4 Разработка схемы программы**

Схема работы программного средства представлена на рисунке 2.1.

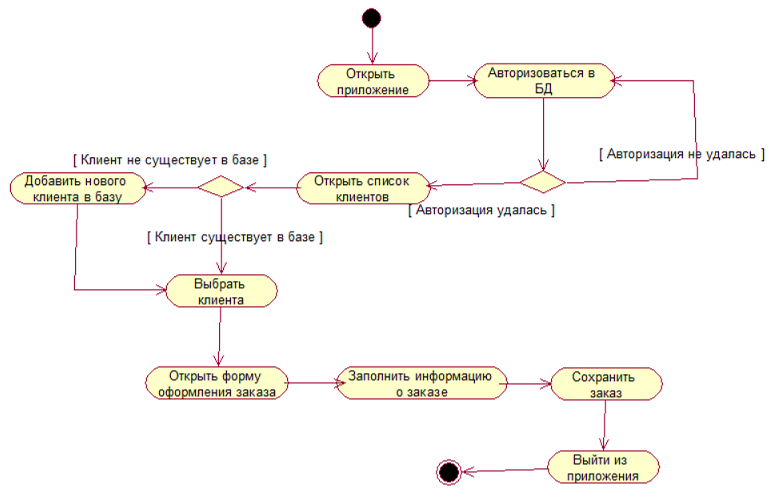


Рисунок 2.1 - Структура программного средства

**2.5 Программная реализация программного средства**

Программное средство предоставляет собой удобный визуальный интерфейс для доступа к необходимым данным. Приложение обеспечивает быстрый доступ к необходимой информации ювелирной мастерской.

Функции:

* отображение списка клиентов;
* фильтрация списка клиентов;
* добавление, редактирование, удаление информации о клиентах;
* отображение списка заказов;
* фильтрация списка заказов;
* добавление, редактирование, удаление информации о заказах;
* отображение списка изделий;
* фильтрация списка изделий;
* добавление, редактирование, удаление информации о изделиях;
* отображение списка материалов;
* фильтрация списка материалов;
* добавление, редактирование, удаление информации о материалах;
* отображение списка пользователей;
* фильтрация списка пользователей;
* добавление, редактирование, удаление информации о пользователях;

**3 ТЕСТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА**

Тесты существенно различаются по задачам, которые с их помощью решаются, и по используемой технике. Различие задач тестирования приводит, естественным образом, к необходимости использовать весьма разнообразные типы (виды) тестирования. Принято подразделять тестирование на виды по следующим категориям:

* по объектам (элементам) тестирования, часто разделение на виды тестов по данному критерию называют разделением тестирования на уровни;
* по глубине тестирования, то есть разделение тестовых испытаний на типы проводится в зависимости от количества времени и объема тестируемых компонент программного продукта.

1. Модульное тестирование (Автономное или Unit-тестирование).

На данном уровне тестируются по отдельности небольшие элементы системы, максимально отделенные от других элементов и, в то же время, пригодные для тестирования.

2. Комплексное тестирование (Сборочное тестирование, integration testing или interface testing).

На данном уровне тестируются объединенные элементы (компоненты или подсистемы) общей системы, чаще всего некоторая взаимодействующая между собой группа элементов

3. Системное тестирование (system testing).

После того, как система собрана из составляющих компонентов, она должна быть протестирована на соответствие “Системным спецификациям” – реализованы ли все функциональные и нефункциональные требования к разрабатываемой системе.

4. Приемочное тестирование (Приемо-сдаточное тестирование или acceptance testing).

На данном уровне завершенное приложение (система) тестируется Заказчиком, конечными пользователями или соответствующими уполномоченными с целью определения соответствия системы “Требованиям Заказчика” и готовности системы к внедрению

5. Операционное тестирование (Release Testing)

Тестирование в среде эксплуатации позволяет выявить и нефункциональные проблемы, такие как: конфликт с другими системами, смежными в области бизнеса или в программных и электронных окружениях; недостаточная производительность системы в среде эксплуатации и др.

Виды тестирования:

Инсталляционное тестирование (Installation testing). Основная цель состоит в том, чтобы убедиться, что продукт может быть установлен/деинсталлирован при различных условиях – таких как: новая инсталляция, усовершенствование системы (upgrade), установка по умолчанию, полная установка, установка по выбору.

Дымное тестирование (проверка на дым, Smoke testing). Как правило, используется для определения, готова ли программа для проведения более обширного тестирования.

Функциональное тестирование (Functional testing). Проверка соответствия продукта функциональным требованиям и спецификациям.

Функциональные требования включают в себя: Функциональную пригодность, точность, способность к взаимодействию, соответствие стандартам и правилам, защищённость.

Нефункциональное тестирование описывает тесты, необходимые для определения характеристик программного обеспечения, которые могут быть измерены различными величинами. Тестирование нефункциональных требований, то есть: тестирование производительности, защищенности, использования и восстановления.

Тестирование графического интерфейса пользователя (User Interface testing). Тестирование интерфейса – экранов, кнопок и т.д. Большая часть функциональности ПО реализуется, как правило, через пользовательский интерфейс.

Для начала проверим приложение на безопасность. Попробуем выполнить вход в систему с ошибочными данными.

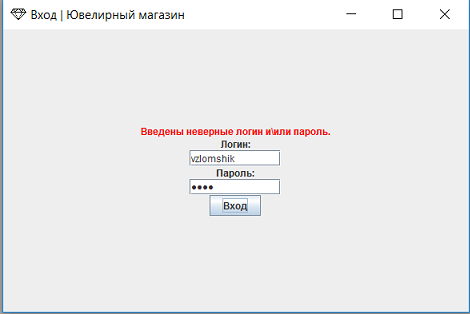


Рисунок 3.1 – Реакция приложения на ввод некорректных данных

Попробуем удалить материал, который в данный момент используется в системе. В случае если это удастся, то появятся некорректная информация в системе.

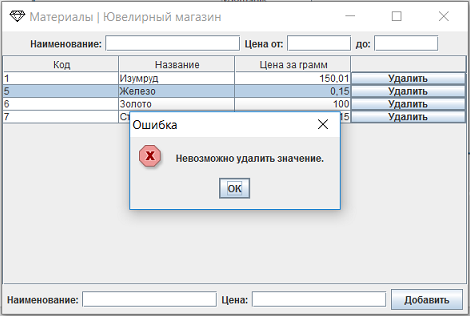


Рисунок 3.2 – Реакция приложения на попытку удаления используемого ресурса

Далее попробуем добавить материал, который уже есть в системе. Если это удастся, то в приложении будет иметь место дублирование уникальной информации, что будет явной ошибкой в системе.

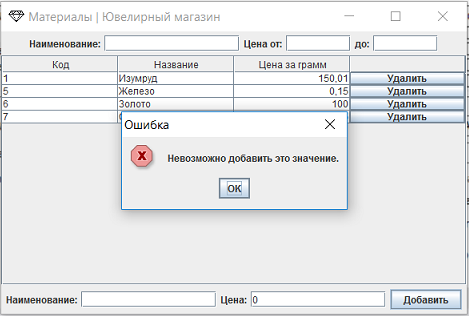


Рисунок 3.3 – Реакция приложения на попытку дублировать существующее значение

Приложение удачно прошло этап тестирования, имеет хороший уровень безопасности и не допускает пользователю внести некорректную информацию в базу.

**4 МЕТОДИКА ИСПОЛЬЗОВАНИЯ ПРОГРАММНОГО СРЕДСТВА**

После запуска приложения можно увидеть главную форму входа. На ней размещены элементы для ввода логина, пароля и кнопка «Вход».

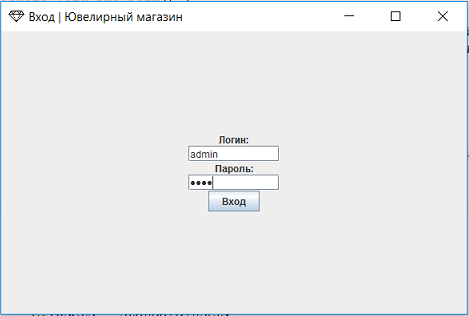


Рисунок 4.1 – Форма входа в приложение

После нажатия на кнопку «Вход» перед пользователем появиться форма, на которой расположены кнопки, позволяющие ему начать работу с нужным разделом системы. Для всех пользователей это будут кнопки: «Клиенты», «Работники», «Заказы», «Изделия», «Материалы». Для суперпользователей также появятся кнопки: «Менеджеры», «Статусы заказов», «Статусы изделий», «Типы изделий», «Статусы оплаты».

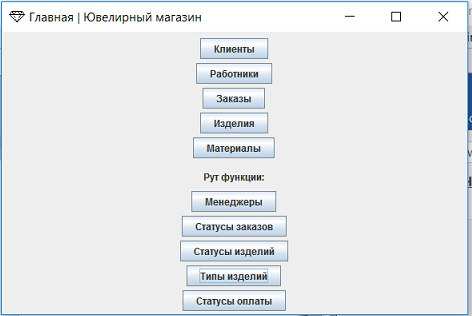


Рисунок 4.2 – Главная форма приложения для суперпользователя

При нажатии одну из кнопок «Статусы заказов», «Статусы изделий», «Типы изделий», «Статусы оплаты» откроется стандартная форма редактирования справочной сущности. Все они имеют одинаковый вид и функционал. Продемонстрирую функционал на форме «Типы изделий».

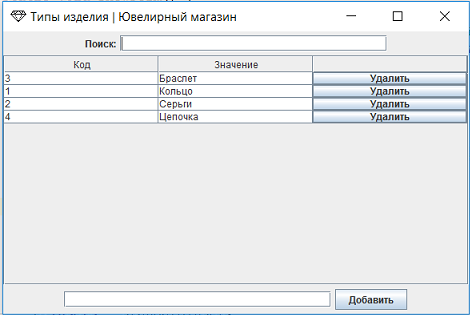


Рисунок 4.3 – Форма для управления типами изделий

На этой форме можно осуществить изменение значения путём редактирования его прямо в таблице. Также возможно удалить и добавить новое значение. В верхней части формы доступно поле для фильтрации списка по полю название.

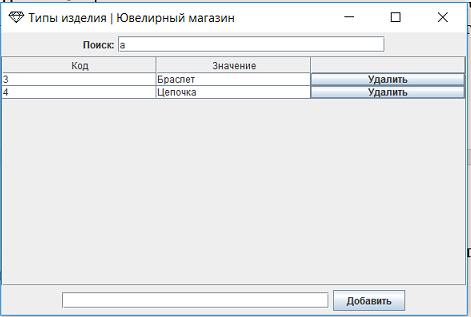


Рисунок 4.4 – Список изделий при активной фильтрации

Для управления менеджерами, пользователь может нажать на кнопку «Менеджеры» и приложение выдаст ему форму для управления.

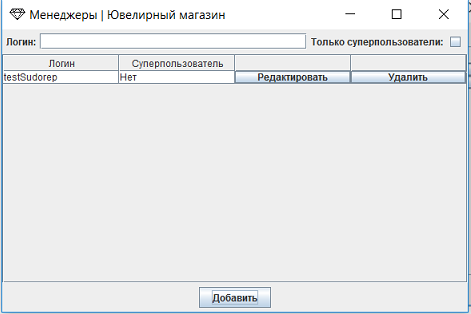


Рисунок 4.5 – Форма для управления менеджерами

На данной форме можно получить доступ к форме редактирования, добавления пользователя, удалить пользователя или использовать фильтрацию. При нажатии на кнопку «Добавить» откроется форма добавления нового пользователя.

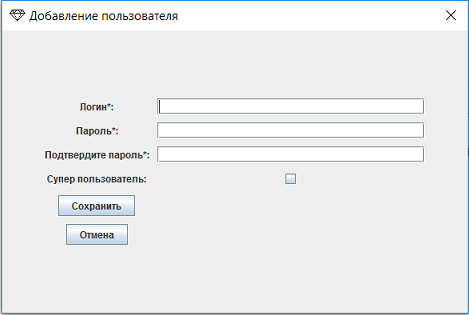


Рисунок 4.6 – Форма добавления пользователя

Для управления клиентами, пользователь может нажать на кнопку «Клиенты» и приложение выдаст ему форму для управления.

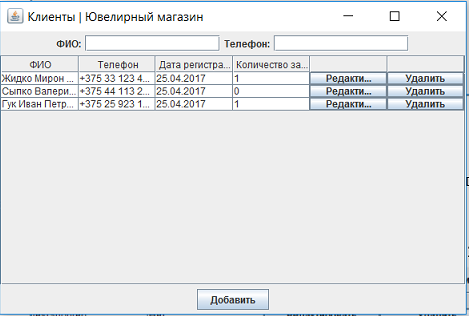


Рисунок 4.7 – Форма для управления клиентами

На данной форме можно получить доступ к форме редактирования, добавления клиента, удалить клиента или использовать фильтрацию. При нажатии на кнопку «Редактировать» откроется форма редактирование информации о клиенте.

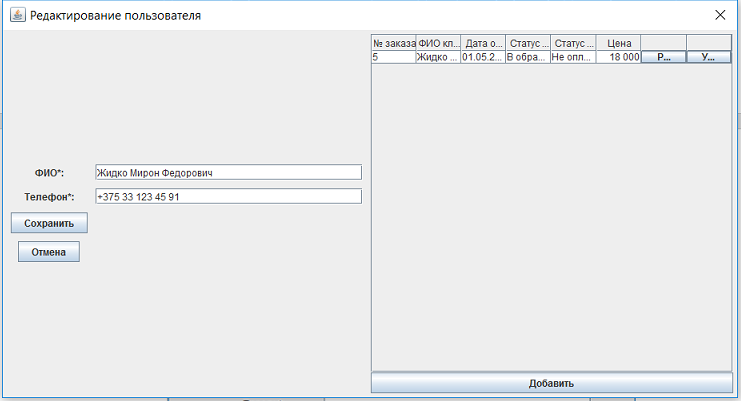


Рисунок 4.8 – Форма редактирования клиента

Для управления заказами, пользователь может нажать на кнопку «Заказы» и приложение выдаст ему форму для управления.

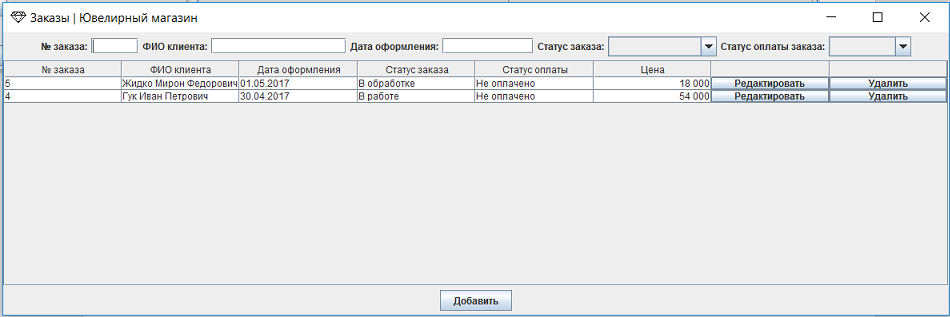


Рисунок 4.9 – Форма для управления заказами

На данной форме можно получить доступ к форме редактирования, добавления заказа, удалить заказ или использовать фильтрацию. При нажатии на кнопку «Редактировать» откроется форма редактирования информации о заказе.

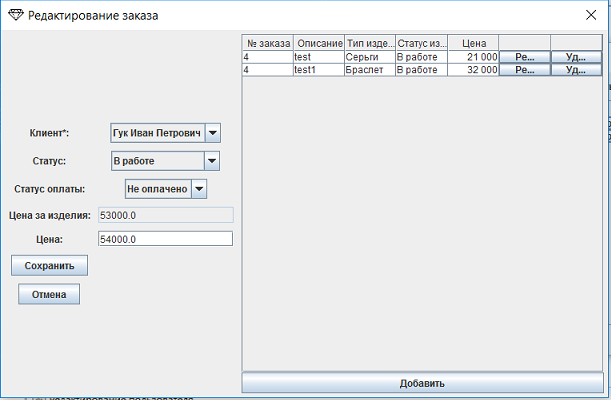


Рисунок 4.10 – Форма редактирования заказа

Из этой же формы у пользователя есть доступ к управлению изделиями, которые относятся к данному заказу. Пользователь может получить доступ к форме добавления, редактирования изделия или удалить его. При нажатии кнопки «Редактировать» напротив нужного изделия откроется форма редактирования изделия.

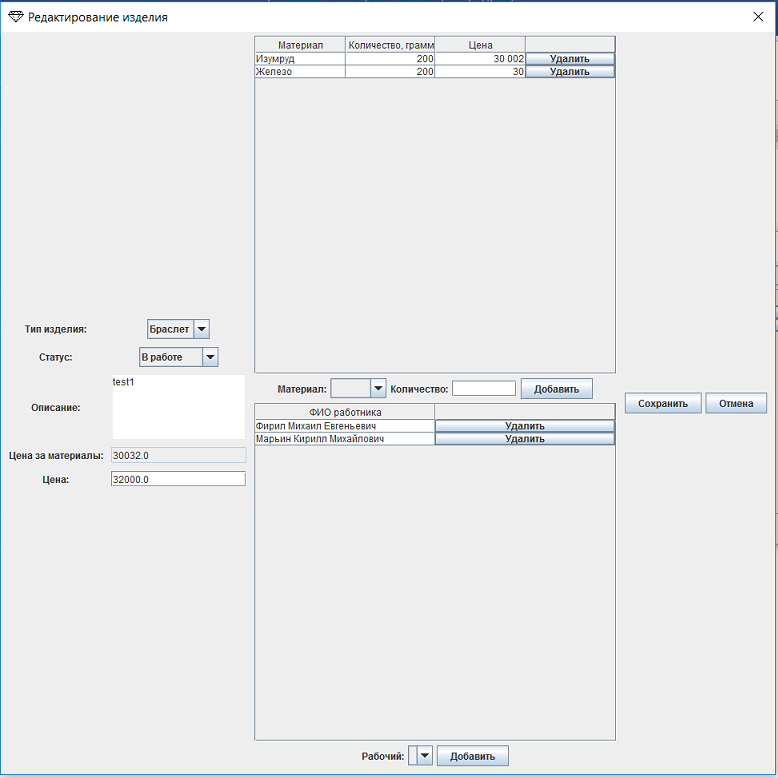


Рисунок 4.11 – Форма редактирования изделия

На данной форме нет доступа к непосредственному редактирования материалов и работников. Но пользователь может указать какие работники работают над текущим изделием, а также какие материалы и в каком количестве использовались при работе над изделием. Эта информация поможет пользователю указать цену за изделие.

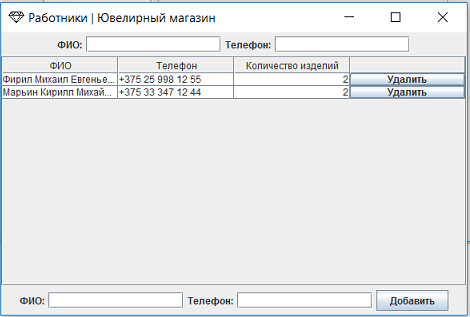


Рисунок 4.12 – Форма управления работниками

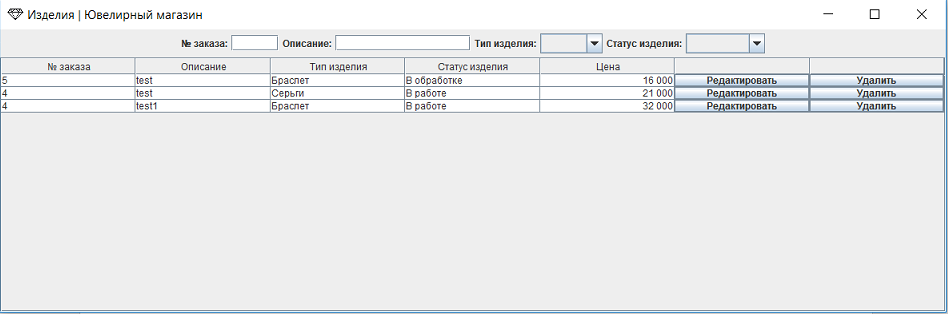


Рисунок 4.13 – Форма управления изделиями

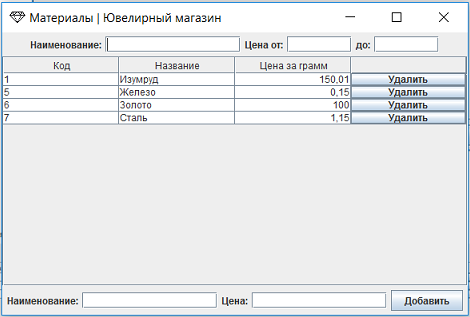


Рисунок 4.14 – Форма управления материалами

**ЗАКЛЮЧЕНИЕ**

В данном курсовом проекте разработано программное средство для автоматизации рабочего места администратора ювелирной мастерской.

В данном программном средстве реализовано:

* управление информацией о клиентах;
* управление информацией о заказах;
* управление информацией о изделиях;
* управление информацией о материалах;
* управление информацией о пользователях;
* управление информацией о работниках;

Курсовая работа была реализована на языке программирования Java с использованием среды разработки JetBrains IDEA и библиотек графического интерфейса Swing и AWT.

Все задачи, которые ставились в начале проектирования были реализованы и в некоторых местах дополнены.

В дальнейшем программное средство может быть модифицировано различными элементами управления и настройками, которые позволят сделать данное программное средство более универсальным, а также позволят решать другие более сложные задачи чем простое управление СУБД.

**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1. Свободная энциклопедия [Электронный ресурс] / центр информ. Режим доступа: http://ru.wikipedia.org/wiki/Windows\_10.
2. Программирование на языке Java: учеб. пособие / В.В. Фаронов; под ред. В.С. Кустов. – Питер, 2007. – 240с.
3. Язык Java: самоучитель / А.В. Фролов, Г.В. Фролов; под ред. О.А Фаронов. – Москва, 2003. – 560с.
4. Дьюсон Робин. для начинающих разработчиков. СПб.: БХВ-Питер, 2009. – 704 с.
5. Сайт разработчика [Электронный ресурс] центр информ. Режим доступа: http://www.protesting.ru/testing/. Дата доступа: 04.04.2017.
6. Энциклопедия [Электронный ресурс] центр информ. Режим доступа: http://habrahabr.ru/post/169381/ /ARM. Дата доступа: 06.04.2017.
7. Канер С., Фолк Дж., Енг Кек Нгуен. Тестирование программного обеспечения, 2004. – 73 с.

**ПРИЛОЖЕНИЕ А.**

Исходный код приложения

**import** javax.imageio.ImageIO;  
**import** javax.swing.\*;  
**import** java.awt.\*;  
**import** java.awt.event.ActionEvent;  
**import** java.awt.event.ActionListener;  
**import** java.io.File;  
**import** java.io.IOException;  
**import** java.security.MessageDigest;  
**import** java.security.NoSuchAlgorithmException;  
**import** java.sql.DriverManager;  
**import** java.sql.ResultSet;  
**import** java.sql.SQLException;  
**import** java.util.Scanner;  
  
*/\*\*  
 \* Created by User on 16.04.2017.  
 \*/***public class** AuthForm {  
 **static void** showForm() {  
  
 JFrame startFrame = **new** JFrame(**"Вход | Ювелирный магазин"**);  
 startFrame.setDefaultCloseOperation(WindowConstants.***EXIT\_ON\_CLOSE***);  
 **try** {  
 startFrame.setIconImage(ImageIO.read(**new** File(**"out/production/jewerly/images/diamond\_ico.jpg"**)));  
 } **catch** (IOException ie) {  
 ie.printStackTrace();  
 }  
  
 startFrame.setSize(600,400);  
 startFrame.setLocationRelativeTo(**null**);  
  
 JPanel loginPanel = **new** JPanel(**new** GridBagLayout());  
 GridBagConstraints c = **new** GridBagConstraints();  
 c.**gridx** = 1;  
  
 JLabel errorFieldLabel = **new** JLabel();  
 errorFieldLabel.setForeground(Color.***RED***);  
 JLabel loginFieldLabel = **new** JLabel(**"Логин:"**);  
 JTextField loginField = **new** JTextField(**""**,10);  
 JLabel passwordFieldLabel = **new** JLabel(**"Пароль:"**);  
 JPasswordField passwordField = **new** JPasswordField(**""**,10);  
 JButton submitButton = **new** JButton(**"Вход"**);  
  
 submitButton.addActionListener(**new** ActionListener() {  
 @Override  
 **public void** actionPerformed(ActionEvent e) {  
 ResultSet loggedSet = **null**;  
  
 errorFieldLabel.setText(**""**);  
 String login = loginField.getText();  
 String password = String.*valueOf*(passwordField.getPassword());  
  
 **if** ((login.length() > 0) && (password.length() > 0)) {  
 **try** {  
 **try** {  
 MessageDigest mg = MessageDigest.*getInstance*(**"MD5"**);  
 mg.update(password.getBytes());  
  
 **byte** byteData[] = mg.digest();  
  
 *//convert the byte to hex format method 1* StringBuffer sb = **new** StringBuffer();  
 **for** (**int** i = 0; i < byteData.**length**; i++) {  
 sb.append(Integer.*toString*((byteData[i] & 0xff) + 0x100, 16).substring(1));  
 }  
 password = sb.toString();  
 } **catch** (NoSuchAlgorithmException ex){  
 ex.printStackTrace();  
 }  
  
 loggedSet = main.*db*.select(**"\*"**,**"managers"**,**"login = ? AND password = ?"**,**new** String[]{login,password},**""**,**"1"**);  
  
  
 **while** (loggedSet.next()) {  
 main.*activeManager* = **new** Manager(loggedSet);  
  
 startFrame.setVisible(**false**);  
 startFrame.dispose();  
  
 GeneralForm.*showForm*();  
 **break**;  
 }  
 } **catch** (SQLException sqlEx) {  
 sqlEx.printStackTrace();  
 } **finally** {  
 main.*db*.closeStatementSet();  
 }  
  
 }  
  
 errorFieldLabel.setText(**"Введены неверные логин и\\или пароль."**);  
 }  
 });  
  
  
 loginPanel.add(errorFieldLabel,c);  
 loginPanel.add(loginFieldLabel,c);  
 loginPanel.add(loginField,c);  
 loginPanel.add(passwordFieldLabel,c);  
 loginPanel.add(passwordField,c);  
 loginPanel.add(submitButton,c);  
  
  
 startFrame.add(loginPanel,BorderLayout.***CENTER***);  
 startFrame.setVisible(**true**);  
  
 }  
}

**import** javax.imageio.ImageIO;  
**import** javax.swing.\*;  
**import** java.awt.\*;  
**import** java.awt.event.ActionEvent;  
**import** java.awt.event.ActionListener;  
**import** java.io.File;  
**import** java.io.IOException;  
  
*/\*\*  
 \* Created by User on 16.04.2017.  
 \*/***public class** GeneralForm {  
 **static void** showForm() {  
  
 JFrame startFrame = **new** JFrame(**"Главная | Ювелирный магазин"**);  
 startFrame.setDefaultCloseOperation(WindowConstants.***EXIT\_ON\_CLOSE***);  
  
 startFrame.setSize(600,400);  
 startFrame.setLocationRelativeTo(**null**);  
  
 **try** {  
 startFrame.setIconImage(ImageIO.*read*(**new** File(**"out/production/jewerly/images/diamond\_ico.jpg"**)));  
 } **catch** (IOException ie) {  
 ie.printStackTrace();  
 }  
  
 JPanel mainPanel = **new** JPanel(**new** GridBagLayout());  
 GridBagConstraints c = **new** GridBagConstraints();  
 c.**gridx** = 1;  
 Insets standardInset = **new** Insets(5,0,0,0);  
 c.**insets** = standardInset;  
  
 JButton clientsButton = **new** JButton(**"Клиенты"**);  
 clientsButton.addActionListener(**new** ActionListener() {  
 @Override  
 **public void** actionPerformed(ActionEvent e) {  
 **new** ClientsForm();  
 }  
 });  
  
 JButton workersButton = **new** JButton(**"Работники"**);  
 workersButton.addActionListener(**new** ActionListener() {  
 @Override  
 **public void** actionPerformed(ActionEvent e) {  
 **new** WorkersForm();  
 }  
 });  
  
 JButton ordersButton = **new** JButton(**"Заказы"**);  
 ordersButton.addActionListener(**new** ActionListener() {  
 @Override  
 **public void** actionPerformed(ActionEvent e) {  
 **new** OrdersForm();  
 }  
 });  
  
 JButton productsButton = **new** JButton(**"Изделия"**);  
 productsButton.addActionListener(**new** ActionListener() {  
 @Override  
 **public void** actionPerformed(ActionEvent e) {  
 **new** ProductsForm();  
 }  
 });  
  
 JButton materialsButton = **new** JButton(**"Материалы"**);  
 materialsButton.addActionListener(**new** ActionListener() {  
 @Override  
 **public void** actionPerformed(ActionEvent e) {  
 **new** MaterialsForm();  
 }  
 });  
  
 mainPanel.add(clientsButton,c);  
 mainPanel.add(workersButton,c);  
 mainPanel.add(ordersButton,c);  
 mainPanel.add(productsButton,c);  
 mainPanel.add(materialsButton,c);  
  
 **if** (main.*activeManager*.getSudo()) {  
 JLabel sudoLabel = **new** JLabel(**"Рут функции:"**);  
  
 JButton managersButton = **new** JButton(**"Менеджеры"**);  
 managersButton.addActionListener(**new** ActionListener() {  
 @Override  
 **public void** actionPerformed(ActionEvent e) {  
 **new** ManagersForm();  
 }  
 });  
  
 JButton ordersStatusesButton = **new** JButton(**"Статусы заказов"**);  
 ordersStatusesButton.addActionListener(**new** ActionListener() {  
 @Override  
 **public void** actionPerformed(ActionEvent e) {  
 **new** ListAddForm(**"Статусы заказов"**,**"orders\_statuses"**);  
 }  
 });  
  
 JButton productsStatusesButton = **new** JButton(**"Статусы изделий"**);  
 productsStatusesButton.addActionListener(**new** ActionListener() {  
 @Override  
 **public void** actionPerformed(ActionEvent e) {  
 **new** ListAddForm(**"Статусы изделий"**, **"products\_statuses"**);  
 }  
 });  
  
 JButton productsTypesButton = **new** JButton(**"Типы изделий"**);  
 productsTypesButton.addActionListener(**new** ActionListener() {  
 @Override  
 **public void** actionPerformed(ActionEvent e) {  
 **new** ListAddForm(**"Типы изделия"**, **"products\_types"**);  
 }  
 });  
  
 JButton paymentsStatusesButton = **new** JButton(**"Статусы оплаты"**);  
 paymentsStatusesButton.addActionListener(**new** ActionListener() {  
 @Override  
 **public void** actionPerformed(ActionEvent e) {  
 **new** ListAddForm(**"Статусы оплаты"**, **"payment\_statuses"**);  
 }  
 });  
  
 c.**insets** = **new** Insets(15,0,5,0);  
 mainPanel.add(sudoLabel,c);  
  
 c.**insets** = standardInset;  
 mainPanel.add(managersButton,c);  
 mainPanel.add(ordersStatusesButton,c);  
 mainPanel.add(productsStatusesButton,c);  
 mainPanel.add(productsTypesButton,c);  
 mainPanel.add(paymentsStatusesButton,c);  
 }  
  
 startFrame.add(mainPanel,BorderLayout.***CENTER***);  
 startFrame.setVisible(**true**);  
 }  
}

**import** javax.swing.\*;  
**import** javax.swing.table.AbstractTableModel;  
**import** javax.swing.table.DefaultTableModel;  
**import** java.awt.\*;  
**import** java.awt.event.\*;  
**import** java.sql.ResultSet;  
**import** java.sql.SQLException;  
**import** java.util.ArrayList;  
**import** java.util.Date;  
  
*/\*\*  
 \* Created by User on 23.04.2017.  
 \*/***public class** ClientsForm {  
  
 **private** JFrame **startFrame**;  
 **private** JTable **listTable**;  
 **private** ClientTableModel **model**;  
  
 **private** JTextField **nameSearchField**;  
 **private** JTextField **phoneSearchField**;  
  
 **public** ClientsForm() {  
  
 UIManager.*put*(**"OptionPane.yesButtonText"**, **"Да"**);  
 UIManager.*put*(**"OptionPane.noButtonText"**, **"Нет"**);  
  
 **startFrame** = **new** JFrame(**"Клиенты | Ювелирный магазин"**);  
 **startFrame**.setDefaultCloseOperation(WindowConstants.***DISPOSE\_ON\_CLOSE***);  
  
  
 **startFrame**.setSize(600,400);  
 **startFrame**.setLocationRelativeTo(**null**);  
  
 ResultSet modelSet = **null**;  
 modelSet = main.db.select(**"\*"**,**"clients"**,**""**,**"id desc"**,**""**);  
  
 model = **new** ClientTableModel(modelSet);  
 model.addModelListener(**new** ModelUpdateListener() {  
 @Override  
 **public void** modelUpdated() {  
 updateModel();  
 }  
 });  
  
 listTable = **new** JTable(model);  
 main.db.closeStatementSet();  
  
 ButtonColumn editBtnColumn = **new** ButtonColumn(listTable, **new** AbstractAction() {  
 @Override  
 **public void** actionPerformed(ActionEvent e) {  
 **int** modelRow = Integer.valueOf( e.getActionCommand() );  
  
 Client client = model.getClient(modelRow);  
 JDialog clientDialog = **new** JDialog(startFrame,**"Редактирование пользователя"**, Dialog.ModalityType.APPLICATION\_MODAL);  
 ClientPanel clientPanel = **new** ClientPanel(client);  
  
 clientDialog.setSize(600,400);  
 clientDialog.getContentPane().add(clientPanel);  
 clientDialog.pack();  
 clientDialog.setLocationRelativeTo(**null**);  
  
 clientDialog.setVisible(**true**);  
  
 **if** (clientPanel.isSaved()) {  
 client = clientPanel.getClient();  
 **if** (!client.save())  
 JOptionPane.showMessageDialog(startFrame,**"Невозможно изменить данные пользователя."**,**"Ошибка"**,JOptionPane.ERROR\_MESSAGE);  
  
 updateModel();  
 }  
 }  
 },4);  
  
 **if** (main.activeManager.getSudo()) {  
 ButtonColumn deleteBtnColumn = **new** ButtonColumn(listTable, **new** AbstractAction() {  
 @Override  
 **public void** actionPerformed(ActionEvent e) {  
 JTable table = (JTable) e.getSource();  
 **int** modelRow = Integer.valueOf(e.getActionCommand());  
 ((ClientTableModel) table.getModel()).deleteValueAt(modelRow);  
 }  
 }, 5);  
 }  
  
 JScrollPane scroll = **new** JScrollPane(listTable);  
 startFrame.add(scroll, BorderLayout.CENTER);  
  
 JPanel controlPanel = **new** JPanel();  
 JPanel searchPanel = **new** JPanel();  
  
 nameSearchField = **new** JTextField(**""**,15);  
 phoneSearchField = **new** JTextField(**""**,15);  
  
 JButton addButton = **new** JButton(**"Добавить"**);  
  
  
 nameSearchField.addKeyListener(**new** KeyAdapter() {  
 @Override  
 **public void** keyReleased(KeyEvent e) {  
 **super**.keyReleased(e);  
 updateModel();  
 }  
 });  
 phoneSearchField.addKeyListener(**new** KeyAdapter() {  
 @Override  
 **public void** keyReleased(KeyEvent e) {  
 **super**.keyReleased(e);  
 updateModel();  
 }  
 });  
 nameSearchField.addFocusListener(**new** FocusAdapter() {  
 @Override  
 **public void** focusGained(FocusEvent e) {  
 **super**.focusGained(e);  
 **if** (nameSearchField.getText().length() > 0)  
 updateModel();  
 }  
 });  
 phoneSearchField.addFocusListener(**new** FocusAdapter() {  
 @Override  
 **public void** focusGained(FocusEvent e) {  
 **super**.focusGained(e);  
 **if** (phoneSearchField.getText().length() > 0)  
 updateModel();  
 }  
 });  
  
 addButton.addActionListener(**new** ActionListener() {  
 @Override  
 **public void** actionPerformed(ActionEvent e) {  
  
 JDialog clientDialog = **new** JDialog(startFrame,**"Добавление клиента"**, Dialog.ModalityType.APPLICATION\_MODAL);  
 ClientPanel clientPanel = **new** ClientPanel();  
  
 clientDialog.setSize(600,400);  
 clientDialog.getContentPane().add(clientPanel);  
 clientDialog.setLocationRelativeTo(**null**);  
  
 clientDialog.setVisible(**true**);  
  
 **if** (clientPanel.isSaved()) {  
 Client newClient = clientPanel.getClient();  
 **if** (!newClient.save())  
 JOptionPane.showMessageDialog(startFrame,**"Невозможно добавить клиента."**,**"Ошибка"**,JOptionPane.ERROR\_MESSAGE);  
 **else** updateModel();  
 }  
  
 }  
 });  
  
 controlPanel.add(addButton);  
  
 searchPanel.add(**new** JLabel(**"ФИО:"**));  
 searchPanel.add(nameSearchField);  
  
 searchPanel.add(**new** JLabel(**"Телефон:"**));  
 searchPanel.add(phoneSearchField);  
  
 startFrame.add(searchPanel,BorderLayout.NORTH);  
 startFrame.add(controlPanel,BorderLayout.SOUTH);  
  
 startFrame.setVisible(**true**);  
 }  
  
 **private void** updateModel() {  
 ResultSet modelSet = **null**;  
  
 **if** (nameSearchField.getText().length() > 0 || phoneSearchField.getText().length() > 0) {  
 ArrayList<String> whereValuesList = **new** ArrayList<String>();  
 String where = **""**;  
  
 **if** (nameSearchField.getText().length() > 0) {  
 where += **"name LIKE ? "**;  
 whereValuesList.add(**"%"** + nameSearchField.getText() + **"%"**);  
 }  
 **if** (phoneSearchField.getText().length() > 0) {  
 where += (where.length() > 0 ? **"AND "** : **""**) + **"phone LIKE ?"**;  
 whereValuesList.add(**"%"** + phoneSearchField.getText() + **"%"**);  
 }  
  
 String[] whereValues = **new** String[whereValuesList.size()];  
 whereValuesList.toArray(whereValues);  
  
 modelSet = main.db.select(**"\*"**, **"clients"**, where, whereValues, **"id desc"**, **""**);  
 } **else** modelSet = main.db.select(**"\*"**,**"clients"**,**""**,**"id desc"**,**""**);  
  
 model.update(modelSet);  
 main.db.closeStatementSet();  
 }  
}  
  
  
**class** ClientTableModel **extends** AbstractTableModel {  
  
 **private** ArrayList<Client> clients;  
 **private** ModelUpdateListener listener;  
  
  
 **public** ClientTableModel(ResultSet rs) {  
 clients = **new** ArrayList<Client>();  
  
 **try** {  
 **while** (rs.next()) {  
 Client cl = **new** Client(rs);  
 cl.fillOrders();  
 **this**.clients.add(cl);  
 }  
 } **catch** (SQLException se) {  
 se.printStackTrace();  
 } **finally** {  
 main.db.closeStatementSet();  
 }  
 }  
  
 **public void** addModelListener(ModelUpdateListener listener) {  
 **this**.listener = listener;  
 }  
  
 **public void** update(ResultSet rs) {  
 clients.clear();  
  
 **try** {  
 **while** (rs.next()) {  
 Client cl = **new** Client(rs);  
 cl.fillOrders();  
 **this**.clients.add(cl);  
  
 }  
 } **catch** (SQLException se) {  
 se.printStackTrace();  
 } **finally** {  
 main.db.closeStatementSet();  
 }  
  
 fireTableDataChanged();  
 }  
  
 **public** Class<?> getColumnClass(**int** columnIndex) {  
 **switch** (columnIndex) {  
 **case** 0:  
 **case** 1:  
 **return** String.**class**;  
 **case** 2:  
 **return** Date.**class**;  
 **case** 3:  
 **return int**.**class**;  
 **case** 4:  
 **case** 5:  
 **return** JButton.**class**;  
 }  
  
 **return** String.**class**;  
 }  
  
 **public int** getColumnCount() {  
 **return** main.activeManager.getSudo() ? 6 : 5;  
 }  
  
 **public** String getColumnName(**int** columnIndex) {  
 **switch** (columnIndex) {  
 **case** 0:  
 **return "ФИО"**;  
 **case** 1:  
 **return "Телефон"**;  
 **case** 2:  
 **return "Дата регистрации"**;  
 **case** 3:  
 **return "Количество заказов"**;  
 }  
 **return ""**;  
 }  
  
 **public int** getRowCount() {  
 **if** (clients == **null**)  
 **return** 0;  
 **return** clients.size();  
 }  
  
 **public** Object getValueAt(**int** rowIndex, **int** columnIndex) {  
 Client client = clients.get(rowIndex);  
 **switch** (columnIndex) {  
 **case** 0:  
 **return** client.getName();  
 **case** 1:  
 **return** client.getPhone();  
 **case** 2:  
 **return** client.getRegistration();  
 **case** 3:  
 **return** client.getOrdersCount();  
 **case** 4:  
 **return "Редактировать"**;  
 **case** 5:  
 **return "Удалить"**;  
 }  
 **return ""**;  
 }  
  
 **public boolean** isCellEditable(**int** rowIndex, **int** columnIndex) {  
 **if** (columnIndex == 4 || columnIndex == 5)  
 **return true**;  
 **return false**;  
 }  
  
 **public void** setValueAt(Object value, **int** rowIndex, **int** columnIndex) {  
 }  
  
 **public** Client getClient(**int** rowIndex) {  
 **return** clients.get(rowIndex);  
 }  
  
 **public void** deleteValueAt (**int** rowIndex) {  
 **if** (JOptionPane.showConfirmDialog(**null**,**"Вы уверены, что хотите удалить клиента? Удалятся также и все заказы."**,**"Вы уверены?"**,JOptionPane.YES\_NO\_OPTION) == JOptionPane.YES\_OPTION) {  
 Client client = clients.get(rowIndex);  
 **if** (!client.delete())  
 JOptionPane.showMessageDialog(**null**,**"Невозможно удалить клиента."**,**"Ошибка"**, JOptionPane.ERROR\_MESSAGE);  
  
 **if** (listener != **null**)  
 listener.modelUpdated();  
 }  
 }  
  
}

**import** javax.swing.\*;  
**import** java.awt.\*;  
**import** java.awt.event.ActionEvent;  
**import** java.awt.event.ActionListener;  
**import** java.sql.ResultSet;  
**import** java.sql.SQLException;  
**import** java.util.ArrayList;  
  
*/\*\*  
 \* Created by User on 23.04.2017.  
 \*/***public class** ClientPanel **extends** JPanel {  
  
 **private** Client client;  
  
 **private** JTextField nameField = **new** JTextField(**""**,30);  
 **private** JTextField phoneField = **new** JTextField(**""**,30);  
  
 **private** JButton saveButton = **new** JButton(**"Сохранить"**);  
 **private** JButton cancelButton = **new** JButton(**"Отмена"**);  
  
 **private** JTable ordersTable;  
 **private** OrderTableModel ordersModel;  
  
 **private boolean** saved = **false**;  
  
 **public** ClientPanel() {  
 **this**.client = **new** Client();  
  
 setLayout(**new** GridBagLayout());  
 GridBagConstraints c = **new** GridBagConstraints();  
  
 c.insets = **new** Insets(5,5,5,5);  
  
 c.gridx = 0;  
 c.gridy = 0;  
 add(**new** JLabel(**"ФИО\*:"**),c);  
 c.gridx = 1;  
 add(nameField,c);  
  
 c.gridx = 0;  
 c.gridy = 1;  
 add(**new** JLabel(**"Телефон\*:"**),c);  
 c.gridx = 1;  
 add(phoneField,c);  
  
 c.gridx = 0;  
 c.gridy = 3;  
 add(saveButton,c);  
 c.gridy = 4;  
 add(cancelButton,c);  
  
  
 saveButton.addActionListener(**new** ActionListener() {  
 @Override  
 **public void** actionPerformed(ActionEvent e) {  
 saveUser();  
 }  
 });  
 cancelButton.addActionListener(**new** ActionListener() {  
 @Override  
 **public void** actionPerformed(ActionEvent e) {  
 cancelDialog();  
 }  
 });  
 }  
  
 **public** ClientPanel (Client cln) {  
 client = cln;  
  
 JPanel fieldsPanel = **new** JPanel(**new** GridBagLayout());  
 GridBagConstraints c = **new** GridBagConstraints();  
  
 JPanel tablePanel = **new** JPanel(**new** BorderLayout());  
  
 c.insets = **new** Insets(5,5,5,5);  
  
 c.gridx = 0;  
 c.gridy = 0;  
 fieldsPanel.add(**new** JLabel(**"ФИО\*:"**),c);  
 c.gridx = 1;  
 nameField.setText(client.getName());  
 fieldsPanel.add(nameField,c);  
  
 c.gridx = 0;  
 c.gridy = 1;  
 fieldsPanel.add(**new** JLabel(**"Телефон\*:"**),c);  
 c.gridx = 1;  
 phoneField.setText(client.getPhone());  
 fieldsPanel.add(phoneField,c);  
  
 c.gridx = 0;  
 c.gridy = 3;  
 fieldsPanel.add(saveButton,c);  
 c.gridy = 4;  
 fieldsPanel.add(cancelButton,c);  
  
  
 saveButton.addActionListener(**new** ActionListener() {  
 @Override  
 **public void** actionPerformed(ActionEvent e) {  
 saveUser();  
 }  
 });  
 cancelButton.addActionListener(**new** ActionListener() {  
 @Override  
 **public void** actionPerformed(ActionEvent e) {  
 cancelDialog();  
 }  
 });  
  
  
 ordersModel = **new** OrderTableModel(client.getOrders());  
 ordersModel.addModelListener(**new** ModelUpdateListener() {  
 @Override  
 **public void** modelUpdated() {  
 updateModel();  
 }  
 });  
  
 ordersTable = **new** JTable(ordersModel);  
 JScrollPane ordersScroll = **new** JScrollPane(ordersTable);  
  
 **new** ButtonColumn(ordersTable, **new** AbstractAction() {  
 @Override  
 **public void** actionPerformed(ActionEvent e) {  
 **int** modelRow = Integer.valueOf( e.getActionCommand() );  
  
 Order order = ordersModel.getOrder(modelRow);  
 editOrder(order);  
  
 }  
 },6);  
  
 **if** (main.*activeManager*.getSudo()) {  
 **new** ButtonColumn(**ordersTable**, **new** AbstractAction() {  
 @Override  
 **public void** actionPerformed(ActionEvent e) {  
 JTable table = (JTable) e.getSource();  
 **int** modelRow = Integer.*valueOf*(e.getActionCommand());  
 ((OrderTableModel) table.getModel()).deleteValueAt(modelRow);  
 }  
 }, 7);  
 }  
  
 JButton addOrder = **new** JButton(**"Добавить"**);  
 Window win = SwingUtilities.*getWindowAncestor*(**this**);  
 addOrder.addActionListener(**new** ActionListener() {  
 @Override  
 **public void** actionPerformed(ActionEvent e) {  
 JDialog orderDialog = **new** JDialog(win,**"Добавление заказа"**, Dialog.ModalityType.***APPLICATION\_MODAL***);  
 OrderPanel orderPanel = **new** OrderPanel(**client**);  
  
 orderDialog.setSize(600,400);  
 orderDialog.getContentPane().add(orderPanel);  
 orderDialog.setLocationRelativeTo(**null**);  
  
 orderDialog.setVisible(**true**);  
  
 **if** (orderPanel.isSaved()) {  
 Order newOrder = orderPanel.getOrder();  
 **if** (!newOrder.save())  
 JOptionPane.*showMessageDialog*(win,**"Невозможно добавить заказ."**,**"Ошибка"**,JOptionPane.***ERROR\_MESSAGE***);  
 **else** {  
 updateModel();  
 editOrder(newOrder);  
 }  
 }  
 }  
 });  
  
 tablePanel.add(ordersScroll,BorderLayout.NORTH);  
 tablePanel.add(addOrder,BorderLayout.SOUTH);  
  
 add(fieldsPanel,BorderLayout.NORTH);  
 add(tablePanel,BorderLayout.SOUTH);  
 }  
  
 **protected void** updateModel() {  
 client.fillOrders();  
 ordersModel.update(client.getOrders());  
 }  
  
 **private boolean** editOrder(Order o) {  
 Window win = SwingUtilities.getWindowAncestor(**this**);  
  
 JDialog orderDialog = **new** JDialog(win,**"Редактирование заказа"**, Dialog.ModalityType.APPLICATION\_MODAL);  
 OrderPanel orderPanel = **new** OrderPanel(o);  
  
 orderDialog.setSize(600,400);  
 orderDialog.getContentPane().add(orderPanel);  
 orderDialog.pack();  
 orderDialog.setLocationRelativeTo(**null**);  
  
  
 orderDialog.setVisible(**true**);  
  
 **if** (orderPanel.isSaved()) {  
 o = orderPanel.getOrder();  
 Boolean res = o.save();  
 **if** (!res)  
 JOptionPane.showMessageDialog(win,**"Невозможно изменить данные заказа."**,**"Ошибка"**,JOptionPane.ERROR\_MESSAGE);  
  
 updateModel();  
  
 **return** res;  
 }  
  
 **return false**;  
 }  
  
 **protected boolean** isSaved() {  
 **return** saved;  
 }  
  
 **protected** Client getClient() {  
 **return this**.client;  
 }  
  
 **private void** saveUser() {  
 Window win = SwingUtilities.getWindowAncestor(**this**);  
 ArrayList<String> errors = **new** ArrayList<String>();  
  
 **if** (nameField.getText().length() < 2)  
 errors.add(**"Введите ФИО"**);  
 **if** (phoneField.getText().length() < 2)  
 errors.add(**"Введите телефон"**);  
  
 **if** (errors.size() > 0) {  
 String errorMessage = **""**;  
 **for** (String error : errors) {  
 errorMessage += error+**"\n"**;  
 }  
  
 JOptionPane.showMessageDialog(win,errorMessage,**"Ошибка"**,JOptionPane.ERROR\_MESSAGE);  
 } **else** {  
 client.setName(nameField.getText());  
 client.setPhone(phoneField.getText());  
 **if** (client.getId() == 0)  
 client.setRegistration(System.currentTimeMillis());  
  
 saved = **true**;  
 win.dispose();  
 }  
 }  
  
 **private void** cancelDialog() {  
 Window win = SwingUtilities.getWindowAncestor(**this**);  
 **if** (win != **null**)  
 win.dispose();  
 }  
}

**import** javax.swing.\*;  
**import** java.awt.\*;  
**import** java.sql.\*;  
**import** java.util.Locale;  
  
*/\*\*  
 \* Created by User on 16.04.2017.  
 \*/***public class** main {  
 **private static final** String ***db\_url*** = **"jdbc:mysql://localhost:3306/jewerly?useUnicode=true&characterEncoding=UTF-8"**;  
 **private static final** String ***db\_user*** = **"root"**;  
 **private static final** String ***db\_password*** = **"123"**;  
  
 **public static** DatabaseController *db*;  
  
 **public static** Manager *activeManager*;  
  
 **public static void** main(String args[]){  
 *db* = **new** DatabaseController(***db\_url***,***db\_user***,***db\_password***);  
  
 AuthForm.*showForm*();  
 }  
}

**import** javax.swing.tree.RowMapper;  
**import** java.sql.\*;  
**import** java.util.ArrayList;  
**import** java.util.List;  
**import** java.util.Stack;  
  
*/\*\*  
 \* Created by User on 18.04.2017.  
\*\*/***public class** DatabaseController **implements** AutoCloseable {  
  
 **private** Connection connection;  
 **private** ArrayList<PreparedStatement> statements;  
 **private** ArrayList<ResultSet> rss;  
  
 **public** DatabaseController(String url, String login, String password) {  
 **try** {  
 **this**.connection = DriverManager.getConnection(url, login, password);  
 **this**.statements = **new** ArrayList<PreparedStatement>();  
 **this**.rss = **new** ArrayList<ResultSet>();  
 } **catch** (SQLException se) {  
 se.printStackTrace();  
 }  
 }  
  
 */\*  
 \* Запрос для кастомного SELECT  
 \* \*/* **protected** ResultSet query (String query, String[] params) {  
 PreparedStatement stm = **null**;  
 ResultSet rs = **null**;  
  
 statements.add(stm);  
 rss.add(rs);  
  
 **try**{  
 stm = **this**.connection.prepareStatement(query);  
  
 **int** paramKey = 1;  
 **for**(String p : params) {  
 stm.setString(paramKey++,p);  
 }  
  
 rs = stm.executeQuery();  
 } **catch** (SQLException se) {  
 se.printStackTrace();  
 } **finally** {  
 **return** rs;  
 }  
  
 }  
*/\*  
\* Select запрос к базе данных. Без экранирования, использовать осторожно  
\*/* **protected** ResultSet select (String fields, String tableName, String where, String orderBy, String limitOffset) {  
 PreparedStatement stm = **null**;  
 ResultSet rs = **null**;  
  
 statements.add(stm);  
 rss.add(rs);  
  
 **try** {  
 stm = **this**.connection.prepareStatement(**"SELECT "** + fields + **" FROM "** + tableName +  
 (where.length() > 0 ? **" WHERE "** + where : **""**) +  
 (orderBy.length() > 0 ? **" ORDER BY "** + orderBy : **""**) +  
 (limitOffset.length() > 0 ? **" LIMIT "** + limitOffset : **""**));  
  
  
 rs = stm.executeQuery();  
 } **catch** (SQLException se) {  
 se.printStackTrace();  
 } **finally** {  
 **return** rs;  
 }  
 }  
  
*/\*  
\* Select запрос к базе данных. С экранированием условия  
\*/* **protected** ResultSet select (String fields, String tableName, String where, String[] whereValues, String orderBy, String limitOffset) {  
 PreparedStatement stm = **null**;  
 ResultSet rs = **null**;  
  
 statements.add(stm);  
 rss.add(rs);  
  
 **try** {  
 stm = **this**.connection.prepareStatement(**"SELECT "** + fields + **" FROM "** + tableName +  
 (where.length() > 0 ? **" WHERE "** + where : **""**) +  
 (orderBy.length() > 0 ? **" ORDER BY "** + orderBy : **""**) +  
 (limitOffset.length() > 0 ? **" LIMIT "** + limitOffset : **""**));  
  
 **for** (**int** i = 1; i <= whereValues.length; i++)  
 stm.setString(i,whereValues[i-1]);  
  
 rs = stm.executeQuery();  
 } **catch** (SQLException se) {  
 se.printStackTrace();  
 } **finally** {  
 **return** rs;  
 }  
 }  
  
*/\*  
\* Insert запрос к базе данных. Вставляет одну строку, использует экранирование  
\* \*/* **protected int** insert (String tableName, String fields, String[] values) {  
 **int** insertedId = 0;  
 **if** (values.**length** == 0)  
 **return** insertedId;  
  
 PreparedStatement stm = **null**;  
  
 **try** {  
 String preparedValues = **""**;  
 **for** (String value : values) {  
 preparedValues += **"?,"**;  
 }  
 preparedValues = preparedValues.substring(0,preparedValues.length()-1);  
  
  
 stm = **this**.connection.prepareStatement(**"INSERT INTO "** + tableName + **" ("** + fields + **") VALUES ("** + preparedValues + **")"**,Statement.RETURN\_GENERATED\_KEYS);  
 **for** (**int** i = 1; i <= values.length; i++) {  
 stm.setString(i,values[i-1]);  
 }  
  
 insertedId = stm.executeUpdate();  
 **if** (insertedId > 0) {  
 **try** (ResultSet generatedKeys = stm.getGeneratedKeys()) {  
 **if** (generatedKeys.next()) {  
 insertedId = generatedKeys.getInt(1);  
 }  
 }  
 }  
 } **catch** (SQLException se) {  
 se.printStackTrace();  
 } **finally** {  
 **try** {stm.close();} **catch** (SQLException se) {se.printStackTrace();}  
 **return** insertedId;  
 }  
 }  
  
*/\*  
\* Insert запрос к базе данных. Вставляет множество строк, использует экранирование  
\* \*/* **protected int** insert (String tableName, String fields, ArrayList<String[]> values) {  
 **int** lastInsertedId = 0;  
 **if** (values.size() == 0)  
 **return** lastInsertedId;  
  
 PreparedStatement stm = **null**;  
  
 **try** {  
 String preparedValues = **""**;  
 **for** (String[] valueArray : values) {  
 preparedValues += **"("**;  
 **for** (String value : valueArray) {  
 preparedValues += **"?,"**;  
 }  
 preparedValues = preparedValues.substring(0,preparedValues.length()-1);  
 preparedValues += **"),"**;  
 }  
 preparedValues = preparedValues.substring(0,preparedValues.length()-1);  
  
  
 stm = **this**.connection.prepareStatement(**"INSERT INTO "** + tableName + **" ("** + fields + **") VALUES "** + preparedValues);  
 **int** index = 1;  
 **for** (String[] valueArray : values) {  
 **for** (String value : valueArray) {  
 stm.setString(index++,value);  
 }  
 }  
  
 lastInsertedId = stm.executeUpdate();  
 } **catch** (SQLException se) {  
 se.printStackTrace();  
 } **finally** {  
 **try** {stm.close();} **catch** (SQLException se) {se.printStackTrace();}  
 **return** lastInsertedId;  
 }  
 }  
  
 */\*  
 \* UPDATE запрос к базе. С экранизацией \*/* **protected int** update (String tableName, String[] fields, String[] values, String where, String[] whereValues) {  
 **int** updateId = 0;  
  
 **if** ((fields.length != values.length) || fields.length == 0)  
 **return** updateId;  
  
 PreparedStatement stm = **null**;  
  
 **try** {  
 String setStatement = **""**;  
 **for** (String field : fields) {  
 setStatement += field + **" = ?,"**;  
 }  
 setStatement = setStatement.substring(0,setStatement.length()-1);  
  
 stm = **this**.connection.prepareStatement(**"UPDATE "** + tableName + **" SET "** + setStatement +  
 (where.length() > 0 ? **" WHERE "** + where : **""**));  
  
 **int** escapeIndex = 1;  
 **for** (String value : values) {  
 stm.setString(escapeIndex++,value);  
 }  
  
 **if** (whereValues.length > 0) {  
 **for** (String value : whereValues) {  
 stm.setString(escapeIndex++,value);  
 }  
 }  
  
 updateId = stm.executeUpdate();  
 } **catch** (SQLException se) {  
 se.printStackTrace();  
 } **finally** {  
 **try** {stm.close();} **catch** (SQLException se) {}  
 **return** updateId;  
 }  
 }  
  
 **protected int** delete (String tableName, String where, String[] whereValues) {  
 **int** deletedId = 0;  
  
 PreparedStatement stm = **null**;  
  
 **try** {  
  
 stm = **this**.connection.prepareStatement(**"DELETE FROM "** + tableName +  
 (where.length() > 0 ? **" WHERE "** + where : **""**));  
  
 **int** escapeIndex = 1;  
  
 **if** (whereValues.length > 0) {  
 **for** (String value : whereValues) {  
 stm.setString(escapeIndex++,value);  
 }  
 }  
  
 deletedId = stm.executeUpdate();  
 } **catch** (SQLException se) {  
 se.printStackTrace();  
 } **finally** {  
 **try** {stm.close();} **catch** (SQLException se) {}  
 **return** deletedId;  
 }  
 }  
  
 **protected void** closeStatementSet() {  
 **try**{  
 ArrayList<PreparedStatement> delStm = **new** ArrayList<PreparedStatement>();  
 **for** (PreparedStatement stm : statements) {  
 **if** (stm != **null**) {  
 **if** (!stm.isClosed()) {  
 **try** {stm.close();} **catch** (SQLException se) {se.printStackTrace();}  
 }  
 }  
 delStm.add(stm);  
 }  
 **for** (PreparedStatement dStm : delStm)  
 statements.remove(dStm);  
  
 ArrayList<ResultSet> delRs = **new** ArrayList<ResultSet>();  
 **for** (ResultSet rs : rss) {  
 **if** (rs != **null**) {  
 **if** (!rs.isClosed()) {  
 **try** {rs.close();} **catch** (SQLException se) {se.printStackTrace();}  
 }  
 }  
 delRs.add(rs);  
 }  
 **for** (ResultSet dRs : delRs)  
 rss.remove(dRs);  
  
 } **catch** (SQLException se) {  
 se.printStackTrace();  
 }  
 }  
  
 **protected void** closeStatementSet(ResultSet outRs) {  
 **try**{  
 **for** (**int** i = 0; i < rss.size(); i ++) {  
 **if** (rss.get(i) == outRs) {  
 **if** (rss.get(i) != **null**) {  
 **if** (!rss.get(i).isClosed()) {  
 **try** {rss.get(i).close();} **catch** (SQLException se) {se.printStackTrace();}  
 }  
 }  
 rss.remove(i);  
  
 **if** (statements.get(i) != **null**) {  
 **if** (!statements.get(i).isClosed()) {  
 **try** {statements.get(i).close();} **catch** (SQLException se) {se.printStackTrace();}  
 }  
 }  
 statements.remove(i);  
 }  
 }  
 } **catch** (SQLException se) {  
 se.printStackTrace();  
 }  
 }  
  
 @Override  
 **public void** close() **throws** Exception {  
 **try** {**this**.connection.close();} **catch** (SQLException se) {se.printStackTrace();}  
 }  
}

**import** javax.swing.\*;  
**import** javax.swing.event.DocumentEvent;  
**import** javax.swing.event.DocumentListener;  
**import** javax.swing.text.BadLocationException;  
**import** java.util.regex.Matcher;  
**import** java.util.regex.Pattern;  
  
**class** PriceDocumentListener **implements** DocumentListener {  
 JTextField **currentField**;  
 Pattern **pricePattern** = Pattern.*compile*(**"^\\d+?(\\.\\d{0,2}$|$)"**);  
 Matcher **priceMatcher**;  
  
 **public** PriceDocumentListener(JTextField field) {  
 **currentField** = field;  
 }  
  
 @Override  
 **public void** changedUpdate(DocumentEvent e) {  
 System.***out***.println(**"change"**);  
 }  
  
 @Override  
 **public void** removeUpdate(DocumentEvent e) {  
 System.***out***.println(**"remove"**);  
 }  
  
 @Override  
 **public void** insertUpdate(DocumentEvent e) {  
 String tempText = **null**;  
 **try** {  
 tempText = e.getDocument().getText(0,e.getDocument().getLength());  
 **if** (tempText.length() > 0) {  
 **priceMatcher** = **pricePattern**.matcher(tempText);  
 **if** (!**priceMatcher**.matches()) {  
 **try** {  
 String originalString = **currentField**.getText(0,e.getOffset()) + **currentField**.getText(e.getOffset() + e.getLength(),**currentField**.getText().length() - (e.getOffset() + e.getLength()));  
 SwingUtilities.*invokeLater*(**new** Runnable() {  
 @Override  
 **public void** run() {  
 **currentField**.setText(originalString);  
 }  
 });  
  
 } **catch** (IllegalStateException ise) {  
 ise.printStackTrace();  
 }  
 }  
 }  
 } **catch** (BadLocationException e1) {  
 e1.printStackTrace();  
 }  
 }  
  
}

**import** javax.imageio.ImageIO;  
**import** javax.swing.\*;  
**import** javax.swing.event.CellEditorListener;  
**import** javax.swing.event.TableModelEvent;  
**import** javax.swing.event.TableModelListener;  
**import** javax.swing.table.\*;  
**import** javax.swing.text.NumberFormatter;  
**import** java.awt.\*;  
**import** java.awt.event.\*;  
**import** java.io.File;  
**import** java.io.IOException;  
**import** java.sql.DriverManager;  
**import** java.sql.ResultSet;  
**import** java.sql.ResultSetMetaData;  
**import** java.sql.SQLException;  
**import** java.text.DecimalFormat;  
**import** java.text.DecimalFormatSymbols;  
**import** java.text.NumberFormat;  
**import** java.util.\*;  
**import** java.util.List;  
  
*/\*\*  
 \* Created by User on 16.04.2017.  
 \*/***public class** WorkersForm {  
  
 **private** JFrame **startFrame**;  
 **private** JTable **listTable**;  
 **private** WorkerTableModel **model**;  
  
 **private** JTextField **searchField**;  
 **private** JTextField **phoneSearchField**;  
  
 **public** WorkersForm() {  
  
 UIManager.*put*(**"OptionPane.yesButtonText"**, **"Да"**);  
 UIManager.*put*(**"OptionPane.noButtonText"**, **"Нет"**);  
  
 **startFrame** = **new** JFrame(**"Работники | Ювелирный магазин"**);  
 **startFrame**.setDefaultCloseOperation(WindowConstants.***DISPOSE\_ON\_CLOSE***);  
  
  
 **startFrame**.setSize(600,400);  
 **startFrame**.setLocationRelativeTo(**null**);  
 **try** {  
 **startFrame**.setIconImage(ImageIO.*read*(**new** File(**"out/production/jewerly/images/diamond\_ico.jpg"**)));  
 } **catch** (IOException ie) {  
 ie.printStackTrace();  
 }  
  
  
 ResultSet modelSet = main.*db*.select(**"\*"**,**"workers"**,**""**,**""**,**""**);  
  
 **model** = **new** WorkerTableModel(modelSet);  
 **model**.addModelListener(**new** ModelUpdateListener() {  
 @Override  
 **public void** modelUpdated() {  
 updateModel();  
 }  
 });  
 main.*db*.closeStatementSet();  
  
 **listTable** = **new** JTable(**model**);  
  
 ButtonColumn deleteBtnColumn = **new** ButtonColumn(**listTable**, **new** AbstractAction() {  
 @Override  
 **public void** actionPerformed(ActionEvent e) {  
 JTable table = (JTable)e.getSource();  
 **int** modelRow = Integer.*valueOf*( e.getActionCommand() );  
 ((WorkerTableModel)table.getModel()).deleteValueAt(modelRow);  
 }  
 },3);  
  
 JScrollPane scroll = **new** JScrollPane(**listTable**);  
 **startFrame**.add(scroll,BorderLayout.***CENTER***);  
  
 JPanel controlPanel = **new** JPanel();  
 JPanel searchPanel = **new** JPanel();  
 JTextField nameField = **new** JTextField(**""**,15);  
 JTextField phoneField = **new** JTextField(**""**,15);  
  
 **searchField** = **new** JTextField(**""**,15);  
 **phoneSearchField** = **new** JTextField(**""**,15);  
 JButton addButton = **new** JButton(**"Добавить"**);  
  
  
 **searchField**.addKeyListener(**new** KeyAdapter() {  
 @Override  
 **public void** keyReleased(KeyEvent e) {  
 **super**.keyReleased(e);  
 updateModel();  
 }  
 });  
 **searchField**.addFocusListener(**new** FocusAdapter() {  
 @Override  
 **public void** focusGained(FocusEvent e) {  
 **super**.focusGained(e);  
 **if** (**searchField**.getText().length() > 0)  
 updateModel();  
 }  
 });  
 **phoneSearchField**.addKeyListener(**new** KeyAdapter() {  
 @Override  
 **public void** keyReleased(KeyEvent e) {  
 **super**.keyReleased(e);  
 updateModel();  
 }  
 });  
 **phoneSearchField**.addFocusListener(**new** FocusAdapter() {  
 @Override  
 **public void** focusGained(FocusEvent e) {  
 **super**.focusGained(e);  
 **if** (**phoneSearchField**.getText().length() > 0)  
 updateModel();  
 }  
 });  
  
 addButton.addActionListener(**new** ActionListener() {  
 @Override  
 **public void** actionPerformed(ActionEvent e) {  
 String name = nameField.getText();  
 String phone = phoneField.getText();  
 nameField.setText(**""**);  
 phoneField.setText(**""**);  
 **if** (name.length() > 0 && phone.length() > 0) {  
 Worker wrk = **new** Worker(name,phone);  
 **model**.insert(wrk);  
 } **else** JOptionPane.*showMessageDialog*(**startFrame**,**"Заполните все поля."**,**"Ошибка"**, JOptionPane.***ERROR\_MESSAGE***);  
 }  
 });  
  
 controlPanel.add(**new** JLabel(**"ФИО:"**));  
 controlPanel.add(nameField);  
 controlPanel.add(**new** JLabel(**"Телефон:"**));  
 controlPanel.add(phoneField);  
 controlPanel.add(addButton);  
  
 searchPanel.add(**new** JLabel(**"ФИО:"**));  
 searchPanel.add(**searchField**);  
 searchPanel.add(**new** JLabel(**"Телефон:"**));  
 searchPanel.add(**phoneSearchField**);  
  
 **startFrame**.add(searchPanel,BorderLayout.***NORTH***);  
 **startFrame**.add(controlPanel,BorderLayout.***SOUTH***);  
  
 **startFrame**.setVisible(**true**);  
 }  
  
 **private void** updateModel() {  
 ResultSet modelSet = **null**;  
  
 **if** (**searchField**.getText().length() > 0 || **phoneSearchField**.getText().length() > 0) {  
 ArrayList<String> whereValuesList = **new** ArrayList<String>();  
 String where = **""**;  
  
 **if** (**searchField**.getText().length() > 0) {  
 where += **"name LIKE ? "**;  
 whereValuesList.add(**"%"** + **searchField**.getText() + **"%"**);  
 }  
 **if** (**phoneSearchField**.getText().length() > 0) {  
 where += (where.length() > 0 ? **"AND "** : **""**) + **"phone LIKE ?"**;  
 whereValuesList.add(**"%"** + **phoneSearchField**.getText() + **"%"**);  
 }  
  
 String[] whereValues = **new** String[whereValuesList.size()];  
 whereValuesList.toArray(whereValues);  
  
 modelSet = main.*db*.select(**"\*"**, **"workers"**, where, whereValues, **""**, **""**);  
 } **else** modelSet = main.*db*.select(**"\*"**,**"workers"**,**""**,**""**,**""**);  
  
 **model**.update(modelSet);  
 main.*db*.closeStatementSet();  
 }  
}  
  
**class** WorkerTableModel **extends** AbstractTableModel {  
  
 **private** ArrayList<Worker> **libraries**;  
 **private** ModelUpdateListener **listener**;  
  
 **public** WorkerTableModel(ResultSet rs) {  
 **libraries** = **new** ArrayList<Worker>();  
  
 **try** {  
 **while** (rs.next()) {  
 Worker w = **new** Worker(rs);  
 w.fillProducts();  
 **this**.**libraries**.add(w);  
 }  
 } **catch** (SQLException se) {  
 se.printStackTrace();  
 } **finally** {  
 main.*db*.closeStatementSet();  
 }  
 }  
  
 **public void** addModelListener(ModelUpdateListener listener) {  
 **this**.**listener** = listener;  
 }  
  
 **public void** update(ResultSet rs) {  
 **libraries**.clear();  
  
 **try** {  
 **while** (rs.next()) {  
 Worker w = **new** Worker(rs);  
 w.fillProducts();  
 **this**.**libraries**.add(w);  
 }  
 } **catch** (SQLException se) {  
 se.printStackTrace();  
 } **finally** {  
 main.*db*.closeStatementSet();  
 }  
  
 fireTableDataChanged();  
 }  
  
 **public** Class<?> getColumnClass(**int** columnIndex) {  
 **switch** (columnIndex) {  
 **case** 0:  
 **return** String.**class**;  
 **case** 1:  
 **return** String.**class**;  
 **case** 2:  
 **return** Integer.**class**;  
 **case** 3:  
 **return** JButton.**class**;  
 }  
  
 **return** String.**class**;  
 }  
  
 **public int** getColumnCount() {  
 **return** 4;  
 }  
  
 **public** String getColumnName(**int** columnIndex) {  
 **switch** (columnIndex) {  
 **case** 0:  
 **return "ФИО"**;  
 **case** 1:  
 **return "Телефон"**;  
 **case** 2:  
 **return "Количество изделий"**;  
 **case** 3:  
 **return ""**;  
 }  
 **return ""**;  
 }  
  
 **public int** getRowCount() {  
 **if** (**libraries** == **null**)  
 **return** 0;  
 **return libraries**.size();  
 }  
  
 **public** Object getValueAt(**int** rowIndex, **int** columnIndex) {  
 Worker lib = **libraries**.get(rowIndex);  
 **switch** (columnIndex) {  
 **case** 0:  
 **return** lib.getName();  
 **case** 1:  
 **return** lib.getPhone();  
 **case** 2:  
 **return** lib.getProductsCount();  
 **case** 3:  
 **return "Удалить"**;  
 }  
 **return ""**;  
 }  
  
 **public boolean** isCellEditable(**int** rowIndex, **int** columnIndex) {  
 **if** (columnIndex != 2)  
 **return true**;  
 **return false**;  
 }  
  
 **public void** setValueAt(Object value, **int** rowIndex, **int** columnIndex) {  
 **if** (columnIndex == 3 )  
 **return**;  
 **if** (JOptionPane.*showConfirmDialog*(**null**,**"Вы уверены, что хотите изменить значение?"**,**"Вы уверены?"**,JOptionPane.***YES\_NO\_OPTION***) == JOptionPane.***YES\_OPTION***) {  
 Worker lib = **libraries**.get(rowIndex);  
 **if** (columnIndex == 0)  
 lib.setName(value.toString());  
 **else if** (columnIndex == 1)  
 lib.setPhone(value.toString());  
  
 **if** (!lib.save())  
 JOptionPane.*showMessageDialog*(**null**,**"Невозможно обновить значение."**,**"Ошибка"**, JOptionPane.***ERROR\_MESSAGE***);  
  
 **if** (**listener** != **null**)  
 **listener**.modelUpdated();  
 }  
 }  
  
 **public void** insert (Worker lib) {  
 **if** (lib.save())  
 **libraries**.add(lib);  
 **else** JOptionPane.*showMessageDialog*(**null**,**"Невозможно добавить это значение."**,**"Ошибка"**, JOptionPane.***ERROR\_MESSAGE***);  
  
 **if** (**listener** != **null**)  
 **listener**.modelUpdated();  
 }  
  
 **public void** deleteValueAt (**int** rowIndex) {  
 **if** (JOptionPane.*showConfirmDialog*(**null**,**"Вы уверены, что хотите удалить значение?"**,**"Вы уверены?"**,JOptionPane.***YES\_NO\_OPTION***) == JOptionPane.***YES\_OPTION***) {  
 Worker lib = **libraries**.get(rowIndex);  
 **if** (!lib.delete())  
 JOptionPane.*showMessageDialog*(**null**,**"Невозможно удалить значение."**,**"Ошибка"**, JOptionPane.***ERROR\_MESSAGE***);  
  
 **if** (**listener** != **null**)  
 **listener**.modelUpdated();  
 }  
 }  
}

**import** java.sql.ResultSet;  
**import** java.sql.SQLException;  
**import** java.util.ArrayList;  
  
*/\*\*  
 \* Created by User on 24.04.2017.  
 \*/***public class** Worker {  
  
 **private int id**;  
 **private** String **name**;  
 **private** String **phone**;  
 **private** ArrayList<ProductWorker> **products**;  
  
 **public** Worker() {  
 **this**.**id** = 0;  
 **this**.**name** = **""**;  
 **this**.**phone** = **""**;  
 }  
  
 **public** Worker(**int** id) {  
 ResultSet rs = **null**;  
 **try** {  
 rs = main.*db*.select(**"\*"**,**"workers"**,**"id = ?"**,**new** String[] {String.*valueOf*(id)},**""**,**""**);  
 rs.next();  
  
 **this**.**id** = rs.getInt(**"id"**);  
 **name** = rs.getString(**"name"**);  
 **phone** = rs.getString(**"phone"**);  
  
 **products** = **new** ArrayList<ProductWorker>();  
 } **catch** (SQLException se) {  
 se.printStackTrace();  
 } **finally** {  
 main.*db*.closeStatementSet(rs);  
 }  
 }  
  
 **public** Worker(String name, String phone) {  
 **this**.**id** = 0;  
 **this**.**name** = name;  
 **this**.**phone** = phone;  
  
 **products** = **new** ArrayList<ProductWorker>();  
 }  
  
 **public** Worker(ResultSet rs) {  
 **try** {  
 **this**.**id** = rs.getInt(**"id"**);  
 **this**.**name** = rs.getString(**"name"**);  
 **this**.**phone** = rs.getString(**"phone"**);  
  
 **products** = **new** ArrayList<ProductWorker>();  
 } **catch** (SQLException se) {  
 se.printStackTrace();  
 }  
 }  
  
 **public void** fillProducts() {  
 **products**.clear();  
  
 ResultSet productsSet = **null**;  
 **try** {  
 productsSet = main.*db*.select(**"\*"**, **"products\_workers"**, **"worker\_id = "** + **this**.**id**, **"id desc"**, **""**);  
 **while** (productsSet.next()) {  
 **this**.**products**.add(**new** ProductWorker(productsSet));  
 }  
 } **catch** (SQLException se) {  
 se.printStackTrace();  
 } **finally** {  
 main.*db*.closeStatementSet(productsSet);  
 }  
  
 }  
  
 **public int** getId(){  
 **return id**;  
 }  
 **public** String getName() {  
 **return name**;  
 }  
 **public** String getPhone() {  
 **return phone**;  
 }  
 **public int** getProductsCount() {**return products**.size();}  
  
 **public void** setName(String value) {  
 **name** = value;  
 }  
 **public void** setPhone(String value) {  
 **phone** = value;  
 }  
  
 **public boolean** save() {  
 **if** (**id** == 0) {  
 **id** = main.*db*.insert(**"workers"**, **"name,phone"**, **new** String[]{**name**,**phone**});  
 **if** (**id** > 0)  
 **return true**;  
 }  
 **else** {  
 **if** (main.*db*.update(**"workers"**, **new** String[]{**"name"**,**"phone"**}, **new** String[]{**name**,**phone**}, **"id = ?"**, **new** String[]{String.*valueOf*(**id**)}) > 0)  
 **return true**;  
 }  
  
 **return false**;  
 }  
  
 **public boolean** delete() {  
 **if** (**id** == 0)  
 **return false**;  
  
 **if** (main.*db*.delete(**"workers"**,**"id = ?"**,**new** String[] {String.*valueOf*(**id**)}) > 0)  
 **return true**;  
  
 **return false**;  
 }  
}

**import** java.sql.ResultSet;  
**import** java.sql.SQLException;  
  
*/\*\*  
 \* Created by User on 01.05.2017.  
 \*/***public class** ProductWorker {  
  
 **private int** id;  
 **private** Product product;  
 **private** Worker worker;  
  
 **public** ProductWorker(Product p, Worker w) {  
 product = p;  
 worker = w;  
 }  
  
 **public** ProductWorker(ResultSet rs) {  
 **try** {  
 id = rs.getInt(**"id"**);  
 product = **new** Product(rs.getInt(**"product\_id"**));  
 worker = **new** Worker(rs.getInt(**"worker\_id"**));  
 } **catch** (SQLException se) {  
 se.printStackTrace();  
 }  
 }  
  
 **public int** getId() {  
 **return** id;  
 }  
 **public** Product getProduct() {  
 **return** product;  
 }  
 **public** Worker getWorker() {  
 **return** worker;  
 }  
  
 **public boolean** save() {  
 **if** (id == 0) {  
 id = main.db.insert(**"products\_workers"**, **"product\_id,worker\_id"**,  
 **new** String[]{String.valueOf(product.getId()),String.valueOf(worker.getId())});  
 **if** (id > 0)  
 **return true**;  
 }  
  
 **return false**;  
 }  
  
 **public boolean** delete() {  
 **if** (id == 0)  
 **return false**;  
  
 **if** (main.db.delete(**"products\_workers"**,**"id = ?"**,**new** String[] {String.valueOf(id)}) > 0)  
 **return true**;  
  
 **return false**;  
 }  
}

**import** com.sun.org.apache.xpath.internal.operations.Bool;  
**import** javafx.scene.control.DatePicker;  
  
**import** javax.imageio.ImageIO;  
**import** javax.swing.\*;  
**import** javax.swing.table.AbstractTableModel;  
**import** javax.swing.table.DefaultTableModel;  
**import** java.awt.\*;  
**import** java.awt.event.\*;  
**import** java.io.File;  
**import** java.io.IOException;  
**import** java.sql.ResultSet;  
**import** java.sql.SQLException;  
**import** java.text.DateFormat;  
**import** java.text.ParseException;  
**import** java.text.SimpleDateFormat;  
**import** java.util.ArrayList;  
**import** java.util.Date;  
**import** java.util.regex.Matcher;  
**import** java.util.regex.Pattern;  
  
*/\*\*  
 \* Created by User on 30.04.2017.  
 \*/***public class** ProductsForm {  
  
 **private** JFrame **startFrame**;  
 **private** JTable **listTable**;  
 **private** ProductTableModel **model**;  
  
 **private** JTextField **idSearchField**;  
 **private** JTextField **descriptionSearchField**;  
 **private** JComboBox **statusSearchField**;  
 **private** JComboBox **typeSearchField**;  
  
 **public** ProductsForm() {  
  
 UIManager.*put*(**"OptionPane.yesButtonText"**, **"Да"**);  
 UIManager.*put*(**"OptionPane.noButtonText"**, **"Нет"**);  
  
 **startFrame** = **new** JFrame(**"Изделия | Ювелирный магазин"**);  
 **startFrame**.setDefaultCloseOperation(WindowConstants.***DISPOSE\_ON\_CLOSE***);  
  
 **startFrame**.setSize(1200,400);  
 **startFrame**.setLocationRelativeTo(**null**);  
  
 **try** {  
 **startFrame**.setIconImage(ImageIO.*read*(**new** File(**"out/production/jewerly/images/diamond\_ico.jpg"**)));  
 } **catch** (IOException ie) {  
 ie.printStackTrace();  
 }  
  
 ResultSet modelSet = **null**;  
 modelSet = main.*db*.select(**"\*"**,**"products"**,**""**,**"id desc"**,**""**);  
  
 **model** = **new** ProductTableModel(modelSet);  
 **model**.addModelListener(**new** ModelUpdateListener() {  
 @Override  
 **public void** modelUpdated() {  
 updateModel();  
 }  
 });  
  
 **listTable** = **new** JTable(**model**);  
 main.*db*.closeStatementSet();  
  
 ButtonColumn editBtnColumn = **new** ButtonColumn(**listTable**, **new** AbstractAction() {  
 @Override  
 **public void** actionPerformed(ActionEvent e) {  
 **int** modelRow = Integer.*valueOf*( e.getActionCommand() );  
  
 Product product = **model**.getProduct(modelRow);  
 JDialog productDialog = **new** JDialog(**startFrame**,**"Редактирование изделия"**, Dialog.ModalityType.***APPLICATION\_MODAL***);  
 ProductPanel productPanel = **new** ProductPanel(product);  
  
 productDialog.setSize(600,400);  
 productDialog.getContentPane().add(productPanel);  
 productDialog.pack();  
 productDialog.setLocationRelativeTo(**null**);  
  
  
 productDialog.setVisible(**true**);  
  
 **if** (productPanel.isSaved()) {  
 product = productPanel.getProduct();  
 **if** (!product.save())  
 JOptionPane.*showMessageDialog*(**startFrame**,**"Невозможно изменить данные изделия."**,**"Ошибка"**,JOptionPane.***ERROR\_MESSAGE***);  
  
 updateModel();  
 }  
 }  
 },5);  
  
 **if** (main.*activeManager*.getSudo()) {  
 ButtonColumn deleteBtnColumn = **new** ButtonColumn(**listTable**, **new** AbstractAction() {  
 @Override  
 **public void** actionPerformed(ActionEvent e) {  
 JTable table = (JTable) e.getSource();  
 **int** modelRow = Integer.*valueOf*(e.getActionCommand());  
 ((ProductTableModel) table.getModel()).deleteValueAt(modelRow);  
 }  
 }, 6);  
 }  
  
 JScrollPane scroll = **new** JScrollPane(**listTable**);  
  
 JPanel searchPanel = **new** JPanel();  
  
 **idSearchField** = **new** JTextField(**""**,5);  
 **idSearchField**.addKeyListener(**new** KeyAdapter() {  
 @Override  
 **public void** keyReleased(KeyEvent e) {  
 **super**.keyReleased(e);  
 updateModel();  
 }  
 });  
 **idSearchField**.addFocusListener(**new** FocusAdapter() {  
 @Override  
 **public void** focusGained(FocusEvent e) {  
 **super**.focusGained(e);  
 **if** (**idSearchField**.getText().length() > 0)  
 updateModel();  
 }  
 });  
  
 **descriptionSearchField** = **new** JTextField(**""**,15);  
 **descriptionSearchField**.addKeyListener(**new** KeyAdapter() {  
 @Override  
 **public void** keyReleased(KeyEvent e) {  
 **super**.keyReleased(e);  
 updateModel();  
 }  
 });  
 **descriptionSearchField**.addFocusListener(**new** FocusAdapter() {  
 @Override  
 **public void** focusGained(FocusEvent e) {  
 **super**.focusGained(e);  
 **if** (**descriptionSearchField**.getText().length() > 0)  
 updateModel();  
 }  
 });  
  
 **statusSearchField** = **new** JComboBox();  
 **statusSearchField**.addItem(**new** ComboItem());  
 ResultSet statuses = **null**;  
 statuses = main.*db*.select(**"\*"**,**"products\_statuses"**,**""**,**""**,**""**);  
 **try** {  
 **while** (statuses.next()) {  
 **statusSearchField**.addItem(**new** ComboItem(statuses,**"name"**,**"id"**));  
 }  
 } **catch** (SQLException se) {  
 se.printStackTrace();  
 } **finally** {  
 main.*db*.closeStatementSet(statuses);  
 }  
 **statusSearchField**.addActionListener(**new** ActionListener() {  
 @Override  
 **public void** actionPerformed(ActionEvent e) {  
 updateModel();  
 }  
 });  
  
 **typeSearchField** = **new** JComboBox();  
 **typeSearchField**.addItem(**new** ComboItem());  
 statuses = **null**;  
 statuses = main.*db*.select(**"\*"**,**"products\_types"**,**""**,**""**,**""**);  
 **try** {  
 **while** (statuses.next()) {  
 **typeSearchField**.addItem(**new** ComboItem(statuses,**"name"**,**"id"**));  
 }  
 } **catch** (SQLException se) {  
 se.printStackTrace();  
 } **finally** {  
 main.*db*.closeStatementSet(statuses);  
  
 }  
 **typeSearchField**.addActionListener(**new** ActionListener() {  
 @Override  
 **public void** actionPerformed(ActionEvent e) {  
 updateModel();  
 }  
 });  
  
 searchPanel.add(**new** JLabel(**"№ заказа:"**));  
 searchPanel.add(**idSearchField**);  
  
 searchPanel.add(**new** JLabel(**"Описание:"**));  
 searchPanel.add(**descriptionSearchField**);  
  
 searchPanel.add(**new** JLabel(**"Тип изделия:"**));  
 searchPanel.add(**typeSearchField**);  
  
 searchPanel.add(**new** JLabel(**"Статус изделия:"**));  
 searchPanel.add(**statusSearchField**);  
  
 **startFrame**.add(searchPanel,BorderLayout.***NORTH***);  
 **startFrame**.add(scroll, BorderLayout.***CENTER***);  
  
 **startFrame**.setVisible(**true**);  
 }  
  
 **private void** updateModel() {  
 ResultSet modelSet = **null**;  
  
 **if** (**idSearchField**.getText().length() > 0 ||  
 **descriptionSearchField**.getText().length() > 0 ||  
 **statusSearchField**.getSelectedIndex() > 0 ||  
 **typeSearchField**.getSelectedIndex() > 0) {  
 ArrayList<String> whereValuesList = **new** ArrayList<String>();  
 String where = **""**;  
 Boolean orderJoin = **false**;  
  
 **if** (**idSearchField**.getText().length() > 0) {  
 where += **"o.id = ? "**;  
 whereValuesList.add(**idSearchField**.getText());  
 orderJoin = **true**;  
 }  
 **if** (**descriptionSearchField**.getText().length() > 0) {  
 where += (where.length() > 0 ? **"AND "** : **""**) + **"p.description LIKE ?"**;  
 whereValuesList.add(**"%"** + **descriptionSearchField**.getText() + **"%"**);  
 }  
  
 **if** (**statusSearchField**.getSelectedIndex() > 0) {  
 where += (where.length() > 0 ? **"AND "** : **""**) + **"p.status = ? "**;  
 whereValuesList.add(String.*valueOf*(((ComboItem)**statusSearchField**.getSelectedItem()).getValue()));  
 }  
  
 **if** (**typeSearchField**.getSelectedIndex() > 0) {  
 where += (where.length() > 0 ? **"AND "** : **""**) + **"p.type = ? "**;  
 whereValuesList.add(String.*valueOf*(((ComboItem)**typeSearchField**.getSelectedItem()).getValue()));  
 }  
  
 String[] whereValues = **new** String[whereValuesList.size()];  
 whereValuesList.toArray(whereValues);  
  
 modelSet = main.*db*.query(**"SELECT p.\* FROM products as p "** +  
 (orderJoin ? **"JOIN orders as o "** :**""**) +  
 **"WHERE "** + where + **"ORDER BY p.id desc LIMIT 50"**,  
 whereValues);  
 } **else** modelSet = main.*db*.select(**"\*"**,**"products"**,**""**,**"id desc"**,**""**);  
  
 **model**.update(modelSet);  
 main.*db*.closeStatementSet();  
 }  
}  
  
  
**class** ProductTableModel **extends** AbstractTableModel {  
  
 **private** ArrayList<Product> **products**;  
 **private** ModelUpdateListener **listener**;  
  
 **public** ProductTableModel(ArrayList<Product> p) {  
 **products** = p;  
 }  
  
 **public** ProductTableModel(ResultSet rs) {  
 **products** = **new** ArrayList<Product>();  
  
 **try** {  
 **while** (rs.next()) {  
 **this**.**products**.add(**new** Product(rs));  
 }  
 } **catch** (SQLException se) {  
 se.printStackTrace();  
 } **finally** {  
 main.*db*.closeStatementSet();  
 }  
 }  
  
 **public void** addModelListener(ModelUpdateListener listener) {  
 **this**.**listener** = listener;  
 }  
  
 **public void** update(ArrayList<Product> p) {  
 **products** = p;  
  
 fireTableDataChanged();  
 }  
  
 **public void** update(ResultSet rs) {  
 **products**.clear();  
  
 **try** {  
 **while** (rs.next()) {  
 **this**.**products**.add(**new** Product(rs));  
 }  
 } **catch** (SQLException se) {  
 se.printStackTrace();  
 } **finally** {  
 main.*db*.closeStatementSet();  
 }  
  
 fireTableDataChanged();  
 }  
  
 **public** Class<?> getColumnClass(**int** columnIndex) {  
 **switch** (columnIndex) {  
 **case** 0:  
 **return int**.**class**;  
 **case** 1:  
 **case** 2:  
 **case** 3:  
 **return** String.**class**;  
 **case** 4:  
 **return** Double.**class**;  
 **case** 5:  
 **case** 6:  
 **return** JButton.**class**;  
 }  
  
 **return** String.**class**;  
 }  
  
 **public int** getColumnCount() {  
 **return** main.*activeManager*.getSudo() ? 7 : 6;  
 }  
  
 **public** String getColumnName(**int** columnIndex) {  
 **switch** (columnIndex) {  
 **case** 0:  
 **return "№ заказа"**;  
 **case** 1:  
 **return "Описание"**;  
 **case** 2:  
 **return "Тип изделия"**;  
 **case** 3:  
 **return "Статус изделия"**;  
 **case** 4:  
 **return "Цена"**;  
 }  
 **return ""**;  
 }  
  
 **public int** getRowCount() {  
 **if** (**products** == **null**)  
 **return** 0;  
 **return products**.size();  
 }  
  
 **public** Object getValueAt(**int** rowIndex, **int** columnIndex) {  
 Product product = **products**.get(rowIndex);  
 **switch** (columnIndex) {  
 **case** 0:  
 **return** product.getOrder().getId();  
 **case** 1:  
 **return** product.getDescription();  
 **case** 2:  
 **return** product.getType().getName();  
 **case** 3:  
 **return** product.getStatus().getName();  
 **case** 4:  
 **return** product.getPrice();  
 **case** 5:  
 **return "Редактировать"**;  
 **case** 6:  
 **return "Удалить"**;  
 }  
 **return ""**;  
 }  
  
 **public boolean** isCellEditable(**int** rowIndex, **int** columnIndex) {  
 **if** (columnIndex == 5 || columnIndex == 6)  
 **return true**;  
 **return false**;  
 }  
  
 **public void** setValueAt(Object value, **int** rowIndex, **int** columnIndex) {  
 }  
  
 **public** Product getProduct(**int** rowIndex) {  
 **return products**.get(rowIndex);  
 }  
  
 **public void** deleteValueAt (**int** rowIndex) {  
 **if** (JOptionPane.*showConfirmDialog*(**null**,**"Вы уверены, что хотите удалить изделие?"**,**"Вы уверены?"**,JOptionPane.***YES\_NO\_OPTION***) == JOptionPane.***YES\_OPTION***) {  
 Product product = **products**.get(rowIndex);  
 **if** (!product.delete())  
 JOptionPane.*showMessageDialog*(**null**,**"Невозможно удалить изделие."**,**"Ошибка"**, JOptionPane.***ERROR\_MESSAGE***);  
  
 **if** (**listener** != **null**)  
 **listener**.modelUpdated();  
 }  
 }  
}

**import** javax.swing.\*;  
**import** javax.swing.border.Border;  
**import** javax.swing.table.AbstractTableModel;  
**import** java.awt.\*;  
**import** java.awt.event.ActionEvent;  
**import** java.awt.event.ActionListener;  
**import** java.sql.ResultSet;  
**import** java.sql.SQLException;  
**import** java.util.ArrayList;  
  
*/\*\*  
 \* Created by User on 30.04.2017.  
 \*/***public class** ProductPanel **extends** JPanel {  
  
 **private** Product product;  
  
 **private** JComboBox typeField = **new** JComboBox();  
 **private** JComboBox statusField = **new** JComboBox();  
 **private** JTextArea descriptionField = **new** JTextArea(5,15);  
 **private** JTextField priceField = **new** JTextField(**""**,15);  
 **private** JTextField materialPriceField = **new** JTextField(**""**,15);  
  
 **private** JTable materialsTable;  
 **private** ProductMaterialTableModel materialsModel;  
 **private** JComboBox materialField = **new** JComboBox();  
 **private** JTextField materialCountField = **new** JTextField(**""**,7);  
  
 **private** JTable workersTable;  
 **private** ProductWorkerTableModel workersModel;  
 **private** JComboBox workerField = **new** JComboBox();  
  
 **private** JButton saveButton = **new** JButton(**"Сохранить"**);  
 **private** JButton cancelButton = **new** JButton(**"Отмена"**);  
  
 **private boolean** saved = **false**;  
  
 **public** ProductPanel(Product prd) {  
 **this**.product = prd;  
  
  
 JPanel fieldsPanel = **new** JPanel(**new** GridBagLayout());  
 GridBagConstraints c = **new** GridBagConstraints();  
  
 JPanel tablesPanel = **new** JPanel(**new** BorderLayout());  
 JPanel tablesMaterialPanel = **new** JPanel(**new** BorderLayout());  
 JPanel tablesMaterialControlPanel = **new** JPanel();  
 JPanel tablesWorkerPanel = **new** JPanel(**new** BorderLayout());  
 JPanel tablesWorkerControlPanel = **new** JPanel();  
  
 JPanel controlPanel = **new** JPanel();  
  
 c.insets = **new** Insets(5,5,5,5);  
  
 priceField.getDocument().addDocumentListener(**new** PriceDocumentListener(priceField));  
 priceField.setText(String.valueOf(product.getPrice()));  
  
 descriptionField.setText(product.getDescription());  
  
 ResultSet fillSet = **null**;  
 **try** {  
 fillSet = main.db.select(**"\*"**,**"products\_types"**,**""**,**""**,**""**);  
 **while** (fillSet.next()) {  
 ComboItem ci = **new** ComboItem(fillSet,**"name"**,**"id"**);  
 typeField.addItem(ci);  
  
 **if** (product.getType().getId() == ci.getValue())  
 typeField.setSelectedItem(ci);  
 }  
 } **catch** (SQLException se) {  
 se.printStackTrace();  
 } **finally** {  
 main.db.closeStatementSet(fillSet);  
 }  
  
 fillSet = **null**;  
 **try** {  
 fillSet = main.db.select(**"\*"**,**"products\_statuses"**,**""**,**"name ASC"**,**""**);  
 **while** (fillSet.next()) {  
 ComboItem ci = **new** ComboItem(fillSet,**"name"**,**"id"**);  
 statusField.addItem(ci);  
  
 **if** (product.getStatus().getId() == ci.getValue())  
 statusField.setSelectedItem(ci);  
 }  
 } **catch** (SQLException se) {  
 se.printStackTrace();  
 } **finally** {  
 main.db.closeStatementSet(fillSet);  
 }  
  
 product.fillMaterials();  
  
 materialsModel = **new** ProductMaterialTableModel(product.getMaterials());  
 materialsModel.addModelListener(**new** ModelUpdateListener() {  
 @Override  
 **public void** modelUpdated() {  
 updateMaterialModel();  
 }  
 });  
  
 materialsTable = **new** JTable(materialsModel);  
 JScrollPane materialsScroll = **new** JScrollPane(materialsTable);  
 **new** ButtonColumn(materialsTable, **new** AbstractAction() {  
 @Override  
 **public void** actionPerformed(ActionEvent e) {  
 JTable table = (JTable) e.getSource();  
 **int** modelRow = Integer.valueOf(e.getActionCommand());  
 ((ProductMaterialTableModel) table.getModel()).deleteValueAt(modelRow);  
 }  
 }, 3);  
  
 tablesMaterialControlPanel.add(**new** JLabel(**"Материал:"**));  
 tablesMaterialControlPanel.add(materialField);  
 tablesMaterialControlPanel.add(**new** JLabel(**"Количество:"**));  
 tablesMaterialControlPanel.add(materialCountField);  
  
 JButton tableMaterialAddBtn = **new** JButton(**"Добавить"**);  
 tableMaterialAddBtn.addActionListener(**new** ActionListener() {  
 @Override  
 **public void** actionPerformed(ActionEvent e) {  
 **if** (materialField.getSelectedIndex() > 0 && materialCountField.getText().length() > 0) {  
 Material m = **new** Material(((ComboItem)materialField.getSelectedItem()).getValue());  
 ProductMaterial pm = **new** ProductMaterial(product,m,Integer.valueOf(materialCountField.getText()));  
 materialCountField.setText(**""**);  
 **if** (!pm.save())  
 JOptionPane.showMessageDialog(**null**,**"Невозможно добавить этот материал. Возможно он уже добавлен."**,**"Ошибка"**,JOptionPane.ERROR\_MESSAGE);  
 updateMaterialModel();  
 } **else** JOptionPane.showMessageDialog(**null**,**"Заполните все поля"**,**"Ошибка"**,JOptionPane.ERROR\_MESSAGE);  
 }  
 });  
 tablesMaterialControlPanel.add(tableMaterialAddBtn);  
  
 materialCountField.getDocument().addDocumentListener(**new** IntDocumentListener(materialCountField));  
 ResultSet materialsSet = **null**;  
 materialField.addItem(**new** ComboItem());  
 **try** {  
 String mIds = product.getMaterialsSplitIds();  
 materialsSet = main.db.select(**"\*"**,**"materials"**,(mIds.length() > 0 ? **"id NOT IN ("** + mIds + **")"** : **""**),**""**,**""**);  
 **while** (materialsSet.next())  
 materialField.addItem(**new** ComboItem(materialsSet,**"name"**,**"id"**));  
 } **catch** (SQLException se) {  
 se.printStackTrace();  
 } **finally** {  
 main.db.closeStatementSet(materialsSet);  
 }  
  
 product.fillWorkers();  
  
 workersModel = **new** ProductWorkerTableModel(product.getWorkers());  
 workersModel.addModelListener(**new** ModelUpdateListener() {  
 @Override  
 **public void** modelUpdated() {  
 updateWorkerModel();  
 }  
 });  
  
 workersTable = **new** JTable(workersModel);  
 JScrollPane workersScroll = **new** JScrollPane(workersTable);  
 **new** ButtonColumn(workersTable, **new** AbstractAction() {  
 @Override  
 **public void** actionPerformed(ActionEvent e) {  
 JTable table = (JTable) e.getSource();  
 **int** modelRow = Integer.valueOf(e.getActionCommand());  
 ((ProductWorkerTableModel) table.getModel()).deleteValueAt(modelRow);  
 }  
 }, 1);  
  
 tablesWorkerControlPanel.add(**new** JLabel(**"Рабочий:"**));  
 tablesWorkerControlPanel.add(workerField);  
  
 JButton tableWorkerAddBtn = **new** JButton(**"Добавить"**);  
 tableWorkerAddBtn.addActionListener(**new** ActionListener() {  
 @Override  
 **public void** actionPerformed(ActionEvent e) {  
 **if** (workerField.getSelectedIndex() > 0) {  
 Worker w = **new** Worker(((ComboItem)workerField.getSelectedItem()).getValue());  
 ProductWorker pw = **new** ProductWorker(product,w);  
 **if** (!pw.save())  
 JOptionPane.showMessageDialog(**null**,**"Невозможно добавить работника. Возможно он уже добавлен."**,**"Ошибка"**,JOptionPane.ERROR\_MESSAGE);  
 updateWorkerModel();  
 } **else** JOptionPane.showMessageDialog(**null**,**"Заполните все поля"**,**"Ошибка"**,JOptionPane.ERROR\_MESSAGE);  
 }  
 });  
 tablesWorkerControlPanel.add(tableWorkerAddBtn);  
  
 ResultSet workersSet = **null**;  
 workerField.addItem(**new** ComboItem());  
 **try** {  
 String mIds = product.getWorkersSplitIds();  
 workersSet = main.db.select(**"\*"**,**"workers"**,(mIds.length() > 0 ? **"id NOT IN ("** + mIds + **")"** : **""**),**""**,**""**);  
 **while** (workersSet.next())  
 workerField.addItem(**new** ComboItem(workersSet,**"name"**,**"id"**));  
 } **catch** (SQLException se) {  
 se.printStackTrace();  
 } **finally** {  
 main.db.closeStatementSet(workersSet);  
 }  
  
  
 c.gridx = 0;  
 c.gridy = 0;  
 fieldsPanel.add(**new** JLabel(**"Тип изделия:"**),c);  
 c.gridx = 1;  
 fieldsPanel.add(typeField,c);  
  
 c.gridx = 0;  
 c.gridy = 1;  
 fieldsPanel.add(**new** JLabel(**"Статус:"**),c);  
 c.gridx = 1;  
 fieldsPanel.add(statusField,c);  
  
 c.gridx = 0;  
 c.gridy = 2;  
 fieldsPanel.add(**new** JLabel(**"Описание:"**),c);  
 c.gridx = 1;  
 fieldsPanel.add(descriptionField,c);  
  
 c.gridx = 0;  
 c.gridy = 3;  
 fieldsPanel.add(**new** JLabel(**"Цена за материалы:"**),c);  
 c.gridx = 1;  
 fieldsPanel.add(**materialPriceField**,c);  
 **materialPriceField**.setEditable(**false**);  
  
 Double materialTotalPrice = 0d;  
 **for** (ProductMaterial pm : **product**.getMaterials())  
 materialTotalPrice += pm.getPrice();  
 **materialPriceField**.setText(String.*valueOf*(materialTotalPrice));  
  
 c.**gridx** = 0;  
 c.**gridy** = 4;  
 fieldsPanel.add(**new** JLabel(**"Цена:"**),c);  
 c.**gridx** = 1;  
 fieldsPanel.add(**priceField**,c);  
  
 tablesMaterialPanel.add(materialsScroll,BorderLayout.***NORTH***);  
 tablesMaterialPanel.add(tablesMaterialControlPanel,BorderLayout.***SOUTH***);  
 tablesWorkerPanel.add(workersScroll,BorderLayout.***NORTH***);  
 tablesWorkerPanel.add(tablesWorkerControlPanel,BorderLayout.***SOUTH***);  
 tablesPanel.add(tablesMaterialPanel,BorderLayout.***NORTH***);  
 tablesPanel.add(tablesWorkerPanel,BorderLayout.***SOUTH***);  
  
  
 controlPanel.add(**saveButton**);  
 controlPanel.add(**cancelButton**);  
  
 add(fieldsPanel,BorderLayout.***NORTH***);  
 add(tablesPanel,BorderLayout.CENTER);  
 add(controlPanel,BorderLayout.SOUTH);  
  
  
 saveButton.addActionListener(**new** ActionListener() {  
 @Override  
 **public void** actionPerformed(ActionEvent e) {  
 saveProduct();  
 }  
 });  
 cancelButton.addActionListener(**new** ActionListener() {  
 @Override  
 **public void** actionPerformed(ActionEvent e) {  
 cancelDialog();  
 }  
 });  
 }  
  
 **protected void** updateMaterialModel() {  
 product.fillMaterials();  
 materialsModel.update(product.getMaterials());  
  
 ResultSet materialsSet = **null**;  
 materialField.removeAllItems();  
 materialField.addItem(**new** ComboItem());  
 **try** {  
 String mIds = product.getMaterialsSplitIds();  
 materialsSet = main.db.select(**"\*"**,**"materials"**,(mIds.length() > 0 ? **"id NOT IN ("** + mIds + **")"** : **""**),**""**,**""**);  
 **while** (materialsSet.next())  
 materialField.addItem(**new** ComboItem(materialsSet,**"name"**,**"id"**));  
 } **catch** (SQLException se) {  
 se.printStackTrace();  
 } **finally** {  
 main.db.closeStatementSet(materialsSet);  
 }  
  
 Double materialTotalPrice = 0d;  
 **for** (ProductMaterial pm : product.getMaterials())  
 materialTotalPrice += pm.getPrice();  
 materialPriceField.setText(String.valueOf(materialTotalPrice));  
 }  
  
 **protected void** updateWorkerModel() {  
 product.fillWorkers();  
 workersModel.update(product.getWorkers());  
  
 ResultSet workersSet = **null**;  
 workerField.removeAllItems();  
 workerField.addItem(**new** ComboItem());  
 **try** {  
 String mIds = product.getWorkersSplitIds();  
 workersSet = main.db.select(**"\*"**,**"workers"**,(mIds.length() > 0 ? **"id NOT IN ("** + mIds + **")"** : **""**),**""**,**""**);  
 **while** (workersSet.next())  
 workerField.addItem(**new** ComboItem(workersSet,**"name"**,**"id"**));  
 } **catch** (SQLException se) {  
 se.printStackTrace();  
 } **finally** {  
 main.db.closeStatementSet(workersSet);  
 }  
 }  
  
 **protected boolean** isSaved() {  
 **return** saved;  
 }  
  
 **protected** Product getProduct() {  
 **return this**.product;  
 }  
  
 **private void** saveProduct() {  
 Window win = SwingUtilities.getWindowAncestor(**this**);  
 ArrayList<String> errors = **new** ArrayList<String>();  
  
 **if** (errors.size() > 0) {  
 String errorMessage = **""**;  
 **for** (String error : errors) {  
 errorMessage += error+**"\n"**;  
 }  
  
 JOptionPane.showMessageDialog(win,errorMessage,**"Ошибка"**,JOptionPane.ERROR\_MESSAGE);  
 } **else** {  
 product.setType(**new** Library(((ComboItem)typeField.getSelectedItem()).getValue(),**"products\_types"**));  
 product.setStatus(**new** Library(((ComboItem)statusField.getSelectedItem()).getValue(),**"products\_statuses"**));  
 product.setDescription(descriptionField.getText());  
 product.setPrice(priceField.getText().length() > 0 ? Double.valueOf(priceField.getText()) : 0d);  
  
 saved = **true**;  
 win.dispose();  
 }  
 }  
  
 **private void** cancelDialog() {  
 Window win = SwingUtilities.getWindowAncestor(**this**);  
 **if** (win != **null**)  
 win.dispose();  
 }  
}  
  
**class** ProductMaterialTableModel **extends** AbstractTableModel {  
 **private** ArrayList<ProductMaterial> products\_materials;  
 **private** ModelUpdateListener listener;  
  
 **public** ProductMaterialTableModel(ArrayList<ProductMaterial> pm) {  
 products\_materials = pm;  
 }  
  
 **public** ProductMaterialTableModel(ResultSet rs) {  
 products\_materials = **new** ArrayList<ProductMaterial>();  
  
 **try** {  
 **while** (rs.next()) {  
 **this**.products\_materials.add(**new** ProductMaterial(rs));  
 }  
 } **catch** (SQLException se) {  
 se.printStackTrace();  
 } **finally** {  
 main.db.closeStatementSet();  
 }  
 }  
  
 **public void** addModelListener(ModelUpdateListener listener) {  
 **this**.listener = listener;  
 }  
  
 **public void** update(ResultSet rs) {  
 products\_materials.clear();  
  
 **try** {  
 **while** (rs.next()) {  
 **this**.products\_materials.add(**new** ProductMaterial(rs));  
 }  
 } **catch** (SQLException se) {  
 se.printStackTrace();  
 } **finally** {  
 main.db.closeStatementSet();  
 }  
  
 fireTableDataChanged();  
 }  
  
 **public void** update(ArrayList<ProductMaterial> pm) {  
 products\_materials = pm;  
  
 fireTableDataChanged();  
 }  
  
 **public** Class<?> getColumnClass(**int** columnIndex) {  
 **switch** (columnIndex) {  
 **case** 0:  
 **return** String.**class**;  
 **case** 1:  
 **return** Integer.**class**;  
 **case** 2:  
 **return** Double.**class**;  
 **case** 3:  
 **return** JButton.**class**;  
 }  
  
 **return** String.**class**;  
 }  
  
 **public int** getColumnCount() {  
 **return** 4;  
 }  
  
 **public** String getColumnName(**int** columnIndex) {  
 **switch** (columnIndex) {  
 **case** 0:  
 **return "Материал"**;  
 **case** 1:  
 **return "Количество, грамм"**;  
 **case** 2:  
 **return "Цена"**;  
 }  
 **return ""**;  
 }  
  
 **public int** getRowCount() {  
 **if** (products\_materials == **null**)  
 **return** 0;  
 **return** products\_materials.size();  
 }  
  
 **public** Object getValueAt(**int** rowIndex, **int** columnIndex) {  
 ProductMaterial product\_material = products\_materials.get(rowIndex);  
 **switch** (columnIndex) {  
 **case** 0:  
 **return** product\_material.getMaterial().getName();  
 **case** 1:  
 **return** product\_material.getCount();  
 **case** 2:  
 **return** product\_material.getPrice();  
 **case** 3:  
 **return "Удалить"**;  
 }  
 **return ""**;  
 }  
  
 **public boolean** isCellEditable(**int** rowIndex, **int** columnIndex) {  
 **if** (columnIndex == 1 || columnIndex == 3)  
 **return true**;  
 **return false**;  
 }  
  
 **public void** setValueAt(Object value, **int** rowIndex, **int** columnIndex) {  
 **if** (columnIndex == 1) {  
 ProductMaterial pm = products\_materials.get(rowIndex);  
 pm.setCount((**int**)value);  
  
 **if** (!pm.save())  
 JOptionPane.showMessageDialog(**null**,**"Невозможно изменить значение."**,**"Ошибка"**, JOptionPane.ERROR\_MESSAGE);  
  
 **if** (listener != **null**)  
 listener.modelUpdated();  
  
 }  
  
 }  
  
 **public** ProductMaterial getProductMaterial(**int** rowIndex) {  
 **return** products\_materials.get(rowIndex);  
 }  
  
 **public void** deleteValueAt (**int** rowIndex) {  
 **if** (JOptionPane.showConfirmDialog(**null**,**"Вы уверены, что хотите удалить материал?"**,**"Вы уверены?"**,JOptionPane.YES\_NO\_OPTION) == JOptionPane.YES\_OPTION) {  
 ProductMaterial product = products\_materials.get(rowIndex);  
 **if** (!product.delete())  
 JOptionPane.showMessageDialog(**null**,**"Невозможно удалить материал."**,**"Ошибка"**, JOptionPane.ERROR\_MESSAGE);  
  
 **if** (listener != **null**)  
 listener.modelUpdated();  
 }  
 }  
}  
  
**class** ProductWorkerTableModel **extends** AbstractTableModel {  
 **private** ArrayList<ProductWorker> products\_workers;  
 **private** ModelUpdateListener listener;  
  
 **public** ProductWorkerTableModel(ArrayList<ProductWorker> pm) {  
 products\_workers = pm;  
 }  
  
 **public** ProductWorkerTableModel(ResultSet rs) {  
 products\_workers = **new** ArrayList<ProductWorker>();  
  
 **try** {  
 **while** (rs.next()) {  
 **this**.products\_workers.add(**new** ProductWorker(rs));  
 }  
 } **catch** (SQLException se) {  
 se.printStackTrace();  
 } **finally** {  
 main.db.closeStatementSet();  
 }  
 }  
  
 **public void** addModelListener(ModelUpdateListener listener) {  
 **this**.listener = listener;  
 }  
  
 **public void** update(ResultSet rs) {  
 products\_workers.clear();  
  
 **try** {  
 **while** (rs.next()) {  
 **this**.products\_workers.add(**new** ProductWorker(rs));  
 }  
 } **catch** (SQLException se) {  
 se.printStackTrace();  
 } **finally** {  
 main.db.closeStatementSet();  
 }  
  
 fireTableDataChanged();  
 }  
  
 **public void** update(ArrayList<ProductWorker> pm) {  
 products\_workers = pm;  
  
 fireTableDataChanged();  
 }  
  
 **public** Class<?> getColumnClass(**int** columnIndex) {  
 **switch** (columnIndex) {  
 **case** 0:  
 **return** String.**class**;  
 **case** 1:  
 **return** JButton.**class**;  
 }  
  
 **return** String.**class**;  
 }  
  
 **public int** getColumnCount() {  
 **return** 2;  
 }  
  
 **public** String getColumnName(**int** columnIndex) {  
 **switch** (columnIndex) {  
 **case** 0:  
 **return "ФИО работника"**;  
 }  
 **return ""**;  
 }  
  
 **public int** getRowCount() {  
 **if** (products\_workers == **null**)  
 **return** 0;  
 **return** products\_workers.size();  
 }  
  
 **public** Object getValueAt(**int** rowIndex, **int** columnIndex) {  
 ProductWorker product\_worker = products\_workers.get(rowIndex);  
 **switch** (columnIndex) {  
 **case** 0:  
 **return** product\_worker.getWorker().getName();  
 **case** 1:  
 **return "Удалить"**;  
 }  
 **return ""**;  
 }  
  
 **public boolean** isCellEditable(**int** rowIndex, **int** columnIndex) {  
 **if** (columnIndex == 1)  
 **return true**;  
 **return false**;  
 }  
  
 **public void** setValueAt(Object value, **int** rowIndex, **int** columnIndex) {  
 }  
  
 **public** ProductWorker getProductWorker(**int** rowIndex) {  
 **return** products\_workers.get(rowIndex);  
 }  
  
 **public void** deleteValueAt (**int** rowIndex) {  
 **if** (JOptionPane.showConfirmDialog(**null**,**"Вы уверены, что хотите снять работника?"**,**"Вы уверены?"**,JOptionPane.YES\_NO\_OPTION) == JOptionPane.YES\_OPTION) {  
 ProductWorker product = products\_workers.get(rowIndex);  
 **if** (!product.delete())  
 JOptionPane.showMessageDialog(**null**,**"Невозможно снять работника."**,**"Ошибка"**, JOptionPane.ERROR\_MESSAGE);  
  
 **if** (listener != **null**)  
 listener.modelUpdated();  
 }  
 }  
}

**import** com.sun.org.apache.xpath.internal.operations.Bool;  
**import** javafx.scene.control.DatePicker;  
  
**import** javax.swing.\*;  
**import** javax.swing.table.AbstractTableModel;  
**import** javax.swing.table.DefaultTableModel;  
**import** java.awt.\*;  
**import** java.awt.event.\*;  
**import** java.sql.ResultSet;  
**import** java.sql.SQLException;  
**import** java.text.DateFormat;  
**import** java.text.ParseException;  
**import** java.text.SimpleDateFormat;  
**import** java.util.ArrayList;  
**import** java.util.Date;  
**import** java.util.regex.Matcher;  
**import** java.util.regex.Pattern;  
  
*/\*\*  
 \* Created by User on 25.04.2017.  
 \*/***public class** OrdersForm {  
  
 **private** JFrame **startFrame**;  
 **private** JTable **listTable**;  
 **private** OrderTableModel **model**;  
  
 **private** JTextField **idSearchField**;  
 **private** JTextField **fioSearchField**;  
 **private** JTextField **dateSearchField**;  
 **private** JComboBox **statusSearchField**;  
 **private** JComboBox **paymentStatusSearchField**;  
  
 **public** OrdersForm() {  
  
 UIManager.*put*(**"OptionPane.yesButtonText"**, **"Да"**);  
 UIManager.*put*(**"OptionPane.noButtonText"**, **"Нет"**);  
  
 **startFrame** = **new** JFrame(**"Заказы | Ювелирный магазин"**);  
 **startFrame**.setDefaultCloseOperation(WindowConstants.***DISPOSE\_ON\_CLOSE***);  
  
 **startFrame**.setSize(1200,400);  
 **startFrame**.setLocationRelativeTo(**null**);  
   
   
  
 ResultSet modelSet = **null**;  
 modelSet = main.*db*.select(**"\*"**,**"orders"**,**""**,**"date desc"**,**""**);  
  
 **model** = **new** OrderTableModel(modelSet);  
 **model**.addModelListener(**new** ModelUpdateListener() {  
 @Override  
 **public void** modelUpdated() {  
 updateModel();  
 }  
 });  
  
 **listTable** = **new** JTable(**model**);  
 main.*db*.closeStatementSet();  
  
 ButtonColumn editBtnColumn = **new** ButtonColumn(**listTable**, **new** AbstractAction() {  
 @Override  
 **public void** actionPerformed(ActionEvent e) {  
 **int** modelRow = Integer.*valueOf*( e.getActionCommand() );  
  
 Order order = **model**.getOrder(modelRow);  
 editOrder(order);  
  
 }  
 },6);  
  
 **if** (main.*activeManager*.getSudo()) {  
 ButtonColumn deleteBtnColumn = **new** ButtonColumn(**listTable**, **new** AbstractAction() {  
 @Override  
 **public void** actionPerformed(ActionEvent e) {  
 JTable table = (JTable) e.getSource();  
 **int** modelRow = Integer.*valueOf*(e.getActionCommand());  
 ((OrderTableModel) table.getModel()).deleteValueAt(modelRow);  
 }  
 }, 7);  
 }  
  
 JScrollPane scroll = **new** JScrollPane(**listTable**);  
  
 JPanel controlPanel = **new** JPanel();  
 JPanel searchPanel = **new** JPanel();  
  
 **idSearchField** = **new** JTextField(**""**,5);  
 **idSearchField**.addKeyListener(**new** KeyAdapter() {  
 @Override  
 **public void** keyReleased(KeyEvent e) {  
 **super**.keyReleased(e);  
 updateModel();  
 }  
 });  
 **idSearchField**.addFocusListener(**new** FocusAdapter() {  
 @Override  
 **public void** focusGained(FocusEvent e) {  
 **super**.focusGained(e);  
 **if** (**idSearchField**.getText().length() > 0)  
 updateModel();  
 }  
 });  
  
 **fioSearchField** = **new** JTextField(**""**,15);  
 **fioSearchField**.addKeyListener(**new** KeyAdapter() {  
 @Override  
 **public void** keyReleased(KeyEvent e) {  
 **super**.keyReleased(e);  
 updateModel();  
 }  
 });  
 **fioSearchField**.addFocusListener(**new** FocusAdapter() {  
 @Override  
 **public void** focusGained(FocusEvent e) {  
 **super**.focusGained(e);  
 **if** (**fioSearchField**.getText().length() > 0)  
 updateModel();  
 }  
 });  
  
 **dateSearchField** = **new** JTextField(**""**,10);  
 **dateSearchField**.addKeyListener(**new** KeyAdapter() {  
 @Override  
 **public void** keyReleased(KeyEvent e) {  
 **super**.keyReleased(e);  
 **if** (**dateSearchField**.getText().length() == 10) {  
 Pattern datePattern = Pattern.*compile*(**"\\d\\d\\.\\d\\d\\.\\d\\d\\d\\d"**);  
 Matcher dateMatcher = datePattern.matcher(**dateSearchField**.getText());  
 **if** (dateMatcher.matches())  
 updateModel();  
 }  
 }  
 });  
 **dateSearchField**.addFocusListener(**new** FocusAdapter() {  
 @Override  
 **public void** focusGained(FocusEvent e) {  
 **super**.focusGained(e);  
 **if** (**dateSearchField**.getText().length() == 10) {  
 Pattern datePattern = Pattern.*compile*(**"\\d\\d\\.\\d\\d\\.\\d\\d\\d\\d"**);  
 Matcher dateMatcher = datePattern.matcher(**dateSearchField**.getText());  
 **if** (dateMatcher.matches())  
 updateModel();  
 }  
 }  
 });  
  
 **statusSearchField** = **new** JComboBox();  
 **statusSearchField**.addItem(**new** ComboItem());  
 ResultSet statuses = **null**;  
 statuses = main.*db*.select(**"\*"**,**"orders\_statuses"**,**""**,**""**,**""**);  
 **try** {  
 **while** (statuses.next()) {  
 **statusSearchField**.addItem(**new** ComboItem(statuses,**"name"**,**"id"**));  
 }  
 } **catch** (SQLException se) {  
 se.printStackTrace();  
 } **finally** {  
 main.*db*.closeStatementSet(statuses);  
 }  
 **statusSearchField**.addActionListener(**new** ActionListener() {  
 @Override  
 **public void** actionPerformed(ActionEvent e) {  
 updateModel();  
 }  
 });  
  
 **paymentStatusSearchField** = **new** JComboBox();  
 **paymentStatusSearchField**.addItem(**new** ComboItem());  
 statuses = **null**;  
 statuses = main.*db*.select(**"\*"**,**"payment\_statuses"**,**""**,**""**,**""**);  
 **try** {  
 **while** (statuses.next()) {  
 **paymentStatusSearchField**.addItem(**new** ComboItem(statuses,**"name"**,**"id"**));  
 }  
 } **catch** (SQLException se) {  
 se.printStackTrace();  
 } **finally** {  
 main.*db*.closeStatementSet(statuses);  
  
 }  
 **paymentStatusSearchField**.addActionListener(**new** ActionListener() {  
 @Override  
 **public void** actionPerformed(ActionEvent e) {  
 updateModel();  
 }  
 });  
  
 JButton addButton = **new** JButton(**"Добавить"**);  
  
 addButton.addActionListener(**new** ActionListener() {  
 @Override  
 **public void** actionPerformed(ActionEvent e) {  
 JDialog orderDialog = **new** JDialog(**startFrame**,**"Добавление заказа"**, Dialog.ModalityType.***APPLICATION\_MODAL***);  
 OrderPanel orderPanel = **new** OrderPanel();  
  
 orderDialog.setSize(600,400);  
 orderDialog.getContentPane().add(orderPanel);  
 orderDialog.setLocationRelativeTo(**null**);  
  
 orderDialog.setVisible(**true**);  
  
 **if** (orderPanel.isSaved()) {  
 Order newOrder = orderPanel.getOrder();  
 **if** (!newOrder.save())  
 JOptionPane.*showMessageDialog*(**startFrame**,**"Невозможно добавить заказ."**,**"Ошибка"**,JOptionPane.***ERROR\_MESSAGE***);  
 **else** {  
 updateModel();  
 editOrder(newOrder);  
 }  
 }  
 }  
 });  
  
 controlPanel.add(addButton);  
  
 searchPanel.add(**new** JLabel(**"№ заказа:"**));  
 searchPanel.add(**idSearchField**);  
  
 searchPanel.add(**new** JLabel(**"ФИО клиента:"**));  
 searchPanel.add(**fioSearchField**);  
  
 searchPanel.add(**new** JLabel(**"Дата оформления:"**));  
 searchPanel.add(**dateSearchField**);  
  
 searchPanel.add(**new** JLabel(**"Статус заказа:"**));  
 searchPanel.add(**statusSearchField**);  
  
 searchPanel.add(**new** JLabel(**"Статус оплаты заказа:"**));  
 searchPanel.add(**paymentStatusSearchField**);  
  
  
  
 **startFrame**.add(searchPanel,BorderLayout.***NORTH***);  
 **startFrame**.add(scroll, BorderLayout.***CENTER***);  
 **startFrame**.add(controlPanel,BorderLayout.***SOUTH***);  
  
 **startFrame**.setVisible(**true**);  
 }  
  
 **private boolean** editOrder(Order o) {  
 JDialog orderDialog = **new** JDialog(**startFrame**,**"Редактирование заказа"**, Dialog.ModalityType.***APPLICATION\_MODAL***);  
 OrderPanel orderPanel = **new** OrderPanel(o);  
  
 orderDialog.setSize(600,400);  
 orderDialog.getContentPane().add(orderPanel);  
 orderDialog.pack();  
 orderDialog.setLocationRelativeTo(**null**);  
  
  
 orderDialog.setVisible(**true**);  
  
 **if** (orderPanel.isSaved()) {  
 o = orderPanel.getOrder();  
 Boolean res = o.save();  
 **if** (!res)  
 JOptionPane.*showMessageDialog*(**startFrame**,**"Невозможно изменить данные заказа."**,**"Ошибка"**,JOptionPane.***ERROR\_MESSAGE***);  
  
 updateModel();  
  
 **return** res;  
 }  
  
 **return false**;  
 }  
  
 **private void** updateModel() {  
 ResultSet modelSet = **null**;  
  
 **if** (**idSearchField**.getText().length() > 0 ||  
 **fioSearchField**.getText().length() > 0 ||  
 **dateSearchField**.getText().length() > 0 ||  
 **statusSearchField**.getSelectedIndex() > 0 ||  
 **paymentStatusSearchField**.getSelectedIndex() > 0) {  
 ArrayList<String> whereValuesList = **new** ArrayList<String>();  
 String where = **""**;  
 Boolean clientJoin = **false**;  
  
 **if** (**idSearchField**.getText().length() > 0) {  
 where += **"o.id = ? "**;  
 whereValuesList.add(**idSearchField**.getText());  
 }  
 **if** (**fioSearchField**.getText().length() > 0) {  
 where += (where.length() > 0 ? **"AND "** : **""**) + **"c.name LIKE ?"**;  
 whereValuesList.add(**"%"** + **fioSearchField**.getText() + **"%"**);  
 clientJoin = **true**;  
 }  
 **if** (**dateSearchField**.getText().length() == 10) {  
 Pattern datePattern = Pattern.*compile*(**"\\d\\d\\.\\d\\d\\.\\d\\d\\d\\d"**);  
 Matcher dateMatcher = datePattern.matcher(**dateSearchField**.getText());  
  
 **if** (dateMatcher.matches()) {  
 **long** unixStartTime = 0;  
 **long** unixEndTime = 0;  
 **try** {  
 DateFormat dateFormat = **new** SimpleDateFormat(**"dd.MM.yyyy"**);  
 Date date = dateFormat.parse(**dateSearchField**.getText());  
 unixStartTime = (**long**) date.getTime() / 1000;  
 unixEndTime = unixStartTime + 86400;  
 } **catch** (ParseException pe) {  
 pe.printStackTrace();  
 }  
  
 **if** (unixEndTime > 0 && unixStartTime > 0) {  
 where += (where.length() > 0 ? **"AND "** : **""**) + **"(o.date BETWEEN ? AND ?) "**;  
 whereValuesList.add(String.*valueOf*(unixStartTime));  
 whereValuesList.add(String.*valueOf*(unixEndTime));  
 }  
 }  
 }  
  
 **if** (**statusSearchField**.getSelectedIndex() > 0) {  
 where += (where.length() > 0 ? **"AND "** : **""**) + **"o.status = ? "**;  
 whereValuesList.add(String.*valueOf*(((ComboItem)**statusSearchField**.getSelectedItem()).getValue()));  
 }  
  
 **if** (**paymentStatusSearchField**.getSelectedIndex() > 0) {  
 where += (where.length() > 0 ? **"AND "** : **""**) + **"o.payment\_status = ? "**;  
 whereValuesList.add(String.*valueOf*(((ComboItem)**paymentStatusSearchField**.getSelectedItem()).getValue()));  
 }  
  
 String[] whereValues = **new** String[whereValuesList.size()];  
 whereValuesList.toArray(whereValues);  
  
 modelSet = main.*db*.query(**"SELECT o.\* FROM orders as o "** +  
 (clientJoin ? **"JOIN clients as c "** :**""**) +  
 **"WHERE "** + where + **"ORDER BY o.date desc"**,  
 whereValues);  
 } **else** modelSet = main.*db*.select(**"\*"**,**"orders"**,**""**,**"date desc"**,**""**);  
  
 **model**.update(modelSet);  
 main.*db*.closeStatementSet();  
 }  
}  
  
  
**class** OrderTableModel **extends** AbstractTableModel {  
  
 **private** ArrayList<Order> **orders**;  
 **private** ModelUpdateListener **listener**;  
  
  
 **public** OrderTableModel(ArrayList<Order> o) {  
 **orders** = o;  
 }  
  
 **public** OrderTableModel(ResultSet rs) {  
 **orders** = **new** ArrayList<Order>();  
  
 **try** {  
 **while** (rs.next()) {  
 **this**.**orders**.add(**new** Order(rs));  
 }  
 } **catch** (SQLException se) {  
 se.printStackTrace();  
 } **finally** {  
 main.*db*.closeStatementSet();  
 }  
 }  
  
 **public void** addModelListener(ModelUpdateListener listener) {  
 **this**.**listener** = listener;  
 }  
  
 **public void** update(ArrayList<Order> o) {  
 **orders** = o;  
  
 fireTableDataChanged();  
 }  
  
 **public void** update(ResultSet rs) {  
 **orders**.clear();  
  
 **try** {  
 **while** (rs.next()) {  
 **this**.**orders**.add(**new** Order(rs));  
 }  
 } **catch** (SQLException se) {  
 se.printStackTrace();  
 } **finally** {  
 main.*db*.closeStatementSet();  
 }  
  
 fireTableDataChanged();  
 }  
  
 **public** Class<?> getColumnClass(**int** columnIndex) {  
 **switch** (columnIndex) {  
 **case** 0:  
 **return int**.**class**;  
 **case** 1:  
 **case** 3:  
 **case** 4:  
 **return** String.**class**;  
 **case** 2:  
 **return** Date.**class**;  
 **case** 5:  
 **return** Double.**class**;  
 **case** 6:  
 **case** 7:  
 **return** JButton.**class**;  
 }  
  
 **return** String.**class**;  
 }  
  
 **public int** getColumnCount() {  
 **return** main.*activeManager*.getSudo() ? 8 : 7;  
 }  
  
 **public** String getColumnName(**int** columnIndex) {  
 **switch** (columnIndex) {  
 **case** 0:  
 **return "№ заказа"**;  
 **case** 1:  
 **return "ФИО клиента"**;  
 **case** 2:  
 **return "Дата оформления"**;  
 **case** 3:  
 **return "Статус заказа"**;  
 **case** 4:  
 **return "Статус оплаты"**;  
 **case** 5:  
 **return "Цена"**;  
 }  
 **return ""**;  
 }  
  
 **public int** getRowCount() {  
 **if** (**orders** == **null**)  
 **return** 0;  
 **return orders**.size();  
 }  
  
 **public** Object getValueAt(**int** rowIndex, **int** columnIndex) {  
 Order order = **orders**.get(rowIndex);  
 **switch** (columnIndex) {  
 **case** 0:  
 **return** order.getId();  
 **case** 1:  
 **return** order.getClient().getName();  
 **case** 2:  
 **return** order.getDate();  
 **case** 3:  
 **return** order.getStatus().getName();  
 **case** 4:  
 **return** order.getPaymentStatus().getName();  
 **case** 5:  
 **return** order.getPrice();  
 **case** 6:  
 **return "Редактировать"**;  
 **case** 7:  
 **return "Удалить"**;  
 }  
 **return ""**;  
 }  
  
 **public boolean** isCellEditable(**int** rowIndex, **int** columnIndex) {  
 **if** (columnIndex == 6 || columnIndex == 7)  
 **return true**;  
 **return false**;  
 }  
  
 **public void** setValueAt(Object value, **int** rowIndex, **int** columnIndex) {  
 }  
  
 **public** Order getOrder(**int** rowIndex) {  
 **return orders**.get(rowIndex);  
 }  
  
 **public void** deleteValueAt (**int** rowIndex) {  
 **if** (JOptionPane.*showConfirmDialog*(**null**,**"Вы уверены, что хотите удалить заказ? Удалятся также и все изделия."**,**"Вы уверены?"**,JOptionPane.***YES\_NO\_OPTION***) == JOptionPane.***YES\_OPTION***) {  
 Order order = **orders**.get(rowIndex);  
 **if** (!order.delete())  
 JOptionPane.*showMessageDialog*(**null**,**"Невозможно удалить заказ."**,**"Ошибка"**, JOptionPane.***ERROR\_MESSAGE***);  
  
 **if** (**listener** != **null**)  
 **listener**.modelUpdated();  
 }  
 }  
  
}

**import** javax.swing.\*;  
**import** java.awt.\*;  
**import** java.awt.event.ActionEvent;  
**import** java.awt.event.ActionListener;  
**import** java.sql.ResultSet;  
**import** java.sql.SQLException;  
**import** java.util.ArrayList;  
  
*/\*\*  
 \* Created by User on 30.04.2017.  
 \*/***public class** OrderPanel **extends** JPanel {  
  
 **private** Order **order**;  
  
 **private** JComboBox **clientField** = **new** JComboBox();  
 **private** JComboBox **statusField** = **new** JComboBox();  
 **private** JComboBox **paymentStatusField** = **new** JComboBox();  
  
 **private** JTextField **priceField** = **new** JTextField(**""**,15);  
 **private** JTextField **productPriceField** = **new** JTextField(**""**,15);  
  
 **private** JTable **productsTable**;  
 **private** ProductTableModel **productsModel**;  
  
 **private** JButton **saveButton** = **new** JButton(**"Сохранить"**);  
 **private** JButton **cancelButton** = **new** JButton(**"Отмена"**);  
  
 **private boolean saved** = **false**;  
  
 **public** OrderPanel() {  
 **this**.**order** = **new** Order();  
  
 setLayout(**new** GridBagLayout());  
 GridBagConstraints c = **new** GridBagConstraints();  
  
 c.**insets** = **new** Insets(5,5,5,5);  
  
 **clientField**.addItem(**new** ComboItem());  
 ResultSet fillSet = **null**;  
 **try** {  
 fillSet = main.*db*.select(**"\*"**,**"clients"**,**""**,**"name ASC"**,**""**);  
 **while** (fillSet.next()) {  
 **clientField**.addItem(**new** ComboItem(fillSet,**"name"**,**"id"**));  
 }  
 } **catch** (SQLException se) {  
 se.printStackTrace();  
 } **finally** {  
 main.*db*.closeStatementSet(fillSet);  
 }  
  
 fillSet = **null**;  
 **try** {  
 fillSet = main.*db*.select(**"\*"**,**"orders\_statuses"**,**""**,**"name ASC"**,**""**);  
 **while** (fillSet.next()) {  
 **statusField**.addItem(**new** ComboItem(fillSet,**"name"**,**"id"**));  
 }  
 } **catch** (SQLException se) {  
 se.printStackTrace();  
 } **finally** {  
 main.*db*.closeStatementSet(fillSet);  
 }  
  
 fillSet = **null**;  
 **try** {  
 fillSet = main.*db*.select(**"\*"**,**"payment\_statuses"**,**""**,**"name ASC"**,**""**);  
 **while** (fillSet.next()) {  
 **paymentStatusField**.addItem(**new** ComboItem(fillSet,**"name"**,**"id"**));  
 }  
 } **catch** (SQLException se) {  
 se.printStackTrace();  
 } **finally** {  
 main.*db*.closeStatementSet(fillSet);  
 }  
  
 c.**gridx** = 0;  
 c.**gridy** = 0;  
 add(**new** JLabel(**"Клиент\*:"**),c);  
 c.**gridx** = 1;  
 add(**clientField**,c);  
  
 c.**gridx** = 0;  
 c.**gridy** = 1;  
 add(**new** JLabel(**"Статус:"**),c);  
 c.**gridx** = 1;  
 add(**statusField**,c);  
  
 c.**gridx** = 0;  
 c.**gridy** = 2;  
 add(**new** JLabel(**"Статус оплаты:"**),c);  
 c.**gridx** = 1;  
 add(**paymentStatusField**,c);  
  
 c.**gridx** = 0;  
 c.**gridy** = 3;  
 add(**saveButton**,c);  
 c.**gridy** = 4;  
 add(**cancelButton**,c);  
  
  
 **saveButton**.addActionListener(**new** ActionListener() {  
 @Override  
 **public void** actionPerformed(ActionEvent e) {  
 saveOrder();  
 }  
 });  
 **cancelButton**.addActionListener(**new** ActionListener() {  
 @Override  
 **public void** actionPerformed(ActionEvent e) {  
 cancelDialog();  
 }  
 });  
 }  
  
 **public** OrderPanel(Client cln) {  
 **this**.**order** = **new** Order();  
  
 setLayout(**new** GridBagLayout());  
 GridBagConstraints c = **new** GridBagConstraints();  
  
 c.**insets** = **new** Insets(5,5,5,5);  
  
 **clientField**.addItem(**new** ComboItem());  
 ResultSet fillSet = **null**;  
 **try** {  
 fillSet = main.*db*.select(**"\*"**,**"clients"**,**""**,**"name ASC"**,**""**);  
 **while** (fillSet.next()) {  
 ComboItem ci = **new** ComboItem(fillSet,**"name"**,**"id"**);  
 **clientField**.addItem(ci);  
  
 **if** (cln.getId() == ci.getValue())  
 **clientField**.setSelectedItem(ci);  
 }  
 } **catch** (SQLException se) {  
 se.printStackTrace();  
 } **finally** {  
 main.*db*.closeStatementSet(fillSet);  
 }  
  
 fillSet = **null**;  
 **try** {  
 fillSet = main.*db*.select(**"\*"**,**"orders\_statuses"**,**""**,**"name ASC"**,**""**);  
 **while** (fillSet.next()) {  
 **statusField**.addItem(**new** ComboItem(fillSet,**"name"**,**"id"**));  
 }  
 } **catch** (SQLException se) {  
 se.printStackTrace();  
 } **finally** {  
 main.*db*.closeStatementSet(fillSet);  
 }  
  
 fillSet = **null**;  
 **try** {  
 fillSet = main.*db*.select(**"\*"**,**"payment\_statuses"**,**""**,**"name ASC"**,**""**);  
 **while** (fillSet.next()) {  
 **paymentStatusField**.addItem(**new** ComboItem(fillSet,**"name"**,**"id"**));  
 }  
 } **catch** (SQLException se) {  
 se.printStackTrace();  
 } **finally** {  
 main.*db*.closeStatementSet(fillSet);  
 }  
  
 c.**gridx** = 0;  
 c.**gridy** = 0;  
 add(**new** JLabel(**"Клиент\*:"**),c);  
 c.**gridx** = 1;  
 add(**clientField**,c);  
  
 c.**gridx** = 0;  
 c.**gridy** = 1;  
 add(**new** JLabel(**"Статус:"**),c);  
 c.**gridx** = 1;  
 add(**statusField**,c);  
  
 c.**gridx** = 0;  
 c.**gridy** = 2;  
 add(**new** JLabel(**"Статус оплаты:"**),c);  
 c.**gridx** = 1;  
 add(**paymentStatusField**,c);  
  
 c.**gridx** = 0;  
 c.**gridy** = 3;  
 add(**saveButton**,c);  
 c.**gridy** = 4;  
 add(**cancelButton**,c);  
  
  
 **saveButton**.addActionListener(**new** ActionListener() {  
 @Override  
 **public void** actionPerformed(ActionEvent e) {  
 saveOrder();  
 }  
 });  
 **cancelButton**.addActionListener(**new** ActionListener() {  
 @Override  
 **public void** actionPerformed(ActionEvent e) {  
 cancelDialog();  
 }  
 });  
 }  
  
 **public** OrderPanel (Order ord) {  
 **order** = ord;  
  
 JPanel fieldsPanel = **new** JPanel(**new** GridBagLayout());  
 GridBagConstraints c = **new** GridBagConstraints();  
  
 JPanel tablePanel = **new** JPanel(**new** BorderLayout());  
  
  
 c.**insets** = **new** Insets(5,5,5,5);  
  
 **clientField**.addItem(**order**.getClient().getName());  
 **clientField**.setEditable(**false**);  
  
 ResultSet fillSet = **null**;  
 **try** {  
 fillSet = main.*db*.select(**"\*"**,**"orders\_statuses"**,**""**,**"name ASC"**,**""**);  
 **while** (fillSet.next()) {  
 ComboItem ci = **new** ComboItem(fillSet,**"name"**,**"id"**);  
 **statusField**.addItem(ci);  
  
 **if** (**order**.getStatus().getId() == ci.getValue())  
 **statusField**.setSelectedItem(ci);  
 }  
 } **catch** (SQLException se) {  
 se.printStackTrace();  
 } **finally** {  
 main.*db*.closeStatementSet(fillSet);  
 }  
  
 fillSet = **null**;  
 **try** {  
 fillSet = main.*db*.select(**"\*"**,**"payment\_statuses"**,**""**,**"name ASC"**,**""**);  
 **while** (fillSet.next()) {  
 ComboItem ci = **new** ComboItem(fillSet,**"name"**,**"id"**);  
 **paymentStatusField**.addItem(ci);  
  
 **if** (**order**.getPaymentStatus().getId() == ci.getValue())  
 **paymentStatusField**.setSelectedItem(ci);  
 }  
 } **catch** (SQLException se) {  
 se.printStackTrace();  
 } **finally** {  
 main.*db*.closeStatementSet(fillSet);  
 }  
  
 **priceField**.getDocument().addDocumentListener(**new** PriceDocumentListener(**priceField**));  
 **priceField**.setText(String.*valueOf*(**order**.getPrice()));  
  
 c.**gridx** = 0;  
 c.**gridy** = 0;  
 fieldsPanel.add(**new** JLabel(**"Клиент\*:"**),c);  
 c.**gridx** = 1;  
 fieldsPanel.add(**clientField**,c);  
  
 c.**gridx** = 0;  
 c.**gridy** = 1;  
 fieldsPanel.add(**new** JLabel(**"Статус:"**),c);  
 c.**gridx** = 1;  
 fieldsPanel.add(**statusField**,c);  
  
 c.**gridx** = 0;  
 c.**gridy** = 2;  
 fieldsPanel.add(**new** JLabel(**"Статус оплаты:"**),c);  
 c.**gridx** = 1;  
 fieldsPanel.add(**paymentStatusField**,c);  
  
 c.**gridx** = 0;  
 c.**gridy** = 3;  
 fieldsPanel.add(**new** JLabel(**"Цена за изделия:"**),c);  
 c.**gridx** = 1;  
 fieldsPanel.add(**productPriceField**,c);  
  
 **order**.fillProducts();  
  
 **productPriceField**.setEditable(**false**);  
 Double productTotalPrice = 0d;  
 **for** (Product p : **order**.getProducts())  
 productTotalPrice += p.getPrice();  
 **productPriceField**.setText(String.*valueOf*(productTotalPrice));  
  
 c.**gridx** = 0;  
 c.**gridy** = 4;  
 fieldsPanel.add(**new** JLabel(**"Цена:"**),c);  
 c.**gridx** = 1;  
 fieldsPanel.add(**priceField**,c);  
  
 c.**gridx** = 0;  
 c.**gridy** = 5;  
 fieldsPanel.add(**saveButton**,c);  
 c.**gridy** = 6;  
 fieldsPanel.add(**cancelButton**,c);  
  
 **productsModel** = **new** ProductTableModel(**order**.getProducts());  
 **productsModel**.addModelListener(**new** ModelUpdateListener() {  
 @Override  
 **public void** modelUpdated() {  
 updateModel();  
 }  
 });  
  
 **productsTable** = **new** JTable(**productsModel**);  
 JScrollPane productsScroll = **new** JScrollPane(**productsTable**);  
  
 JButton addProduct = **new** JButton(**"Добавить"**);  
 addProduct.addActionListener(**new** ActionListener() {  
 @Override  
 **public void** actionPerformed(ActionEvent e) {  
 Product product = **new** Product(**order**);  
 product.save();  
 **if** (!editProduct(product))  
 product.delete();  
  
 updateModel();  
 }  
 });  
  
 **new** ButtonColumn(**productsTable**, **new** AbstractAction() {  
 @Override  
 **public void** actionPerformed(ActionEvent e) {  
 **int** modelRow = Integer.*valueOf*( e.getActionCommand() );  
  
 Product product = **productsModel**.getProduct(modelRow);  
 editProduct(product);  
  
 }  
 },5);  
  
 **if** (main.*activeManager*.getSudo()) {  
 **new** ButtonColumn(**productsTable**, **new** AbstractAction() {  
 @Override  
 **public void** actionPerformed(ActionEvent e) {  
 JTable table = (JTable) e.getSource();  
 **int** modelRow = Integer.*valueOf*(e.getActionCommand());  
 ((ProductTableModel) table.getModel()).deleteValueAt(modelRow);  
 }  
 }, 6);  
 }  
  
  
 **saveButton**.addActionListener(**new** ActionListener() {  
 @Override  
 **public void** actionPerformed(ActionEvent e) {  
 saveOrder();  
 }  
 });  
 **cancelButton**.addActionListener(**new** ActionListener() {  
 @Override  
 **public void** actionPerformed(ActionEvent e) {  
 cancelDialog();  
 }  
 });  
  
 tablePanel.add(productsScroll,BorderLayout.***NORTH***);  
 tablePanel.add(addProduct,BorderLayout.***SOUTH***);  
  
 add(fieldsPanel,BorderLayout.***NORTH***);  
 add(tablePanel,BorderLayout.***CENTER***);  
 }  
  
 **protected boolean** editProduct(Product p) {  
 Window win = SwingUtilities.*getWindowAncestor*(**this**);  
 JDialog productDialog = **new** JDialog(win,**"Редактирование изделия"**, Dialog.ModalityType.***APPLICATION\_MODAL***);  
 ProductPanel productPanel = **new** ProductPanel(p);  
  
 productDialog.setSize(600,400);  
 productDialog.getContentPane().add(productPanel);  
 productDialog.pack();  
 productDialog.setLocationRelativeTo(**null**);  
  
 productDialog.setVisible(**true**);  
  
 **if** (productPanel.isSaved()) {  
 p = productPanel.getProduct();  
 Boolean res = p.save();  
 **if** (!res)  
 JOptionPane.*showMessageDialog*(win,**"Невозможно изменить данные изделия."**,**"Ошибка"**,JOptionPane.***ERROR\_MESSAGE***);  
  
 updateModel();  
  
 **return** res;  
 }  
  
 **return false**;  
 }  
  
 **protected void** updateModel() {  
 **order**.fillProducts();  
 **productsModel**.update(**order**.getProducts());  
  
 Double productTotalPrice = 0d;  
 **for** (Product p : **order**.getProducts())  
 productTotalPrice += p.getPrice();  
 **productPriceField**.setText(String.*valueOf*(productTotalPrice));  
 }  
  
 **protected boolean** isSaved() {  
 **return saved**;  
 }  
  
 **protected** Order getOrder() {  
 **return this**.**order**;  
 }  
  
 **private void** saveOrder() {  
 Window win = SwingUtilities.*getWindowAncestor*(**this**);  
 ArrayList<String> errors = **new** ArrayList<String>();  
  
 **if** (**order**.getId() == 0) {  
 **if** (**clientField**.getSelectedIndex() == 0)  
 errors.add(**"Выберите клиента"**);  
 }  
  
 **if** (errors.size() > 0) {  
 String errorMessage = **""**;  
 **for** (String error : errors) {  
 errorMessage += error+**"\n"**;  
 }  
  
 JOptionPane.*showMessageDialog*(win,errorMessage,**"Ошибка"**,JOptionPane.***ERROR\_MESSAGE***);  
 } **else** {  
 **order**.setStatus(**new** Library(((ComboItem)**statusField**.getSelectedItem()).getValue(),**"orders\_statuses"**));  
 **order**.setPaymentStatus(**new** Library(((ComboItem)**paymentStatusField**.getSelectedItem()).getValue(),**"payment\_statuses"**));  
 **if** (**order**.getId() == 0) {  
 **order**.setDate(System.*currentTimeMillis*());  
 **order**.setClient(**new** Client(((ComboItem)**clientField**.getSelectedItem()).getValue()));  
 } **else  
 order**.setPrice(**priceField**.getText().length() > 0 ? Double.*valueOf*(**priceField**.getText()) : 0);  
  
 **saved** = **true**;  
 win.dispose();  
 }  
 }  
  
 **private void** cancelDialog() {  
 Window win = SwingUtilities.*getWindowAncestor*(**this**);  
 **if** (win != **null**)  
 win.dispose();  
 }  
}