Министерство науки и высшего образования Российской Федерации Федеральное государственное бюджетное образовательное учреждение высшего образования

**ИРКУТСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

|  |
| --- |
| Институт информационных технологий  и анализа данных |
| наименование института |

|  |
| --- |
| **Отчет** |
| к лабораторной работе №6 по дисциплине «ООП»  «Организация работы с базой данных» |
| наименование темы  Вариант №12 |

|  |  |  |  |
| --- | --- | --- | --- |
| Выполнил студент | ИСТБ-19-2  шифр | подпись | Ю.С. Кузнецова  И. О. Фамилия |
| Проверил | должность | подпись | Т. В. Маланова  И. О. Фамилия |

Работа защищена с оценкой

Иркутск 2021 г.

Содержание

[1 Постановка задачи 3](#_Toc70067318)

[2 Описание БД 4](#_Toc70067319)

[SQL-запросы к БД 5](#_Toc70067320)

[3 Проектирование графического интерфейса 6](#_Toc70067321)

[4 Описание структуры классов 9](#_Toc70067322)

[5 Проектирование тестов 10](#_Toc70067323)

[6 Результаты тестирования 11](#_Toc70067324)

[7 Листинг программы 19](#_Toc70067325)

[7.1 Главный класс 19](#_Toc70067326)

[7.2 Класс MainWindow 19](#_Toc70067327)

[7.3 Класс DBWorker 29](#_Toc70067328)

[7.4 Класс Orders 34](#_Toc70067329)

[7.5 Класс Clients 35](#_Toc70067330)

[7.6 Класс Employees 36](#_Toc70067331)

[7.7 Класс Works 37](#_Toc70067332)

[7.8 Класс OrdersTable 37](#_Toc70067333)

[7.9 Класс ClientTable 38](#_Toc70067334)

[7.10 Класс WorksTable 40](#_Toc70067335)

[7.11 Класс EmployeesTable 41](#_Toc70067336)

[Список литературы 43](#_Toc70067337)

# 1 Постановка задачи

Цель работы: Применение объектно-ориентированного подхода к

организации доступа к базе данных (БД).

Задачи:

1. Получение общего представления об объектном подходе к организации доступа к БД, выполнению запросов и обработке результатов.

2. Получение практического опыта организации работы с БД средствами платформы java.

Содержание задания

Для выполнения задания необходимо разработать java-приложение с

графическим пользовательским интерфейсом позволяющее просматривать и редактировать информацию из базы данных. Работу предваряет организация и наполнение базы данных (БД) в соответствии с предметной областью из индивидуального задания.

Графический пользовательский интерфейс должен предоставлять

возможности работы с записями основных таблиц БД. Состав используемых

компонентов определяется самостоятельно, но для их размещения должны

быть предусмотрены соответствующие планировщики раскладки

компонентов в основных контейнерах.

Классы, реализующие графический пользовательский интерфейс, должны содержать только обращения к методам классов для выполнения заданий.

Вариант №12

Предметная область: автомастерская. Таблицы: клиенты, работники, работы.

# 2 Описание БД

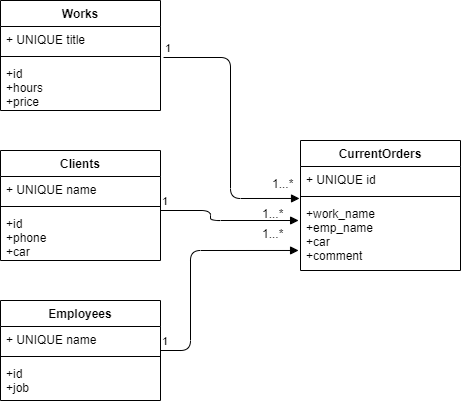


Рисунок 1 – Диаграмма классов БД

Таблица 1 – Описание таблиц БД

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Имя таблицы | Имя атрибута | Тип | Размер | Описание |
| works | id | INTEGER | 8 байт | Идентификатор услуги |
| Unique title | TEXT | 256 байт | Имя пользователя  Ключевое поле |
| price | FLOAT | 4 байта | Стоимость услуги |
| hours | FLOAT | 4 байта | Время выполнения |
| employees | id | INTEGER | 8 байт | Идентификатор исполнителя |
| Unique name | TEXT | 256 байт | Имя исполнителя |
| job | TEXT | 256 байт | Должность |
| clients | id | INTEGER | 8 байт | Идентификатор клиента |
| name | TEXT | 256 байт | Имя клиента |
| phone | TEXT | 256 байт | Номер телефона |
| car | TEXT | 256 байт | Номер машины |
| orders | id | INTEGER | 8 байт | Идентификатор заказа |
| work\_name | TEXT | 256 байт | Услуга |
| emp\_name | TEXT | 256 байт | Имя исполнителя |
| car | TEXT | 256 байт | Номер машины |
| comment | TEXT | 256 байт | Комментарий |
|  | | | | |

# SQL-запросы к БД

CREATE TABLE if not exists 'works' ('id' integer, 'title' TEXT PRIMARY KEY, 'hours' float, 'price' float);

CREATE TABLE if not exists 'employees' ('id' integer, 'name' TEXT PRIMARY KEY, 'job' text);

CREATE TABLE if not exists 'clients' ('id' integer, 'name' TEXT PRIMARY KEY, 'phone' text, 'car' text);

CREATE TABLE if not exists 'CurrentOrders' ('id' INTEGER PRIMARY KEY,'work\_name' TEXT NOT NULL, 'emp\_name' TEXT NOT NULL, 'car' TEXT NOT NULL, 'comment' text, FOREIGN KEY (work\_name) REFERENCES works (title)ON DELETE CASCADE FOREIGN KEY (emp\_name) REFERENCES employees (name)ON DELETE CASCADE FOREIGN KEY (car) REFERENCES clients (car)ON DELETE CASCADE);

INSERT INTO clients('id','name','phone', 'car') +VALUES(?,?,?,?);

SELECT \* FROM clients

DELETE FROM clients WHERE clients.id =

UPDATE clients set id=id-1 WHERE id>

DELETE FROM clients

UPDATE clients SET name=?, phone=?, car=? WHERE id=?

INSERT INTO employees('id','name','job')VALUES(?,?,?);

SELECT \* FROM employees

DELETE FROM employees WHERE employees.id =

UPDATE employees set id=id-1 WHERE id>

DELETE FROM employees

UPDATE employees SET name=?, job=? WHERE id=?

INSERT INTO works('id','title','hours', 'price')VALUES(?,?,?,?);

SELECT \* FROM works

DELETE FROM works WHERE works.id =

UPDATE works set id=id-1 WHERE id>

DELETE FROM works

UPDATE works SET title=?, hours=?, price=? WHERE id=?

SELECT \* FROM CurrentOrders

INSERT INTO CurrentOrders('id','work\_name','emp\_name', 'car','comment')VALUES(?,?,?,?,?);

DELETE FROM CurrentOrders WHERE CurrentOrders.id =

UPDATE CurrentOrders set id=id-1 WHERE id>

DELETE FROM CurrentOrders

SELECT name FROM employees

SELECT title FROM works

SELECT car FROM clients

UPDATE CurrentOrders SET work\_name=?, emp\_name=?, car=?, comment=? WHERE id=?

# 3 Проектирование графического интерфейса

На главном окне с помощью компоновщика BorderLayout размещены вкладки с таблицами (Center). На каждой панели с помощью компоновщика BorderLayout расположена таблица (Center) и панель с кнопками (East). На панелях с кнопками с помощью планировщика GridLayout расположены кнопки и поле ввода для поиска.

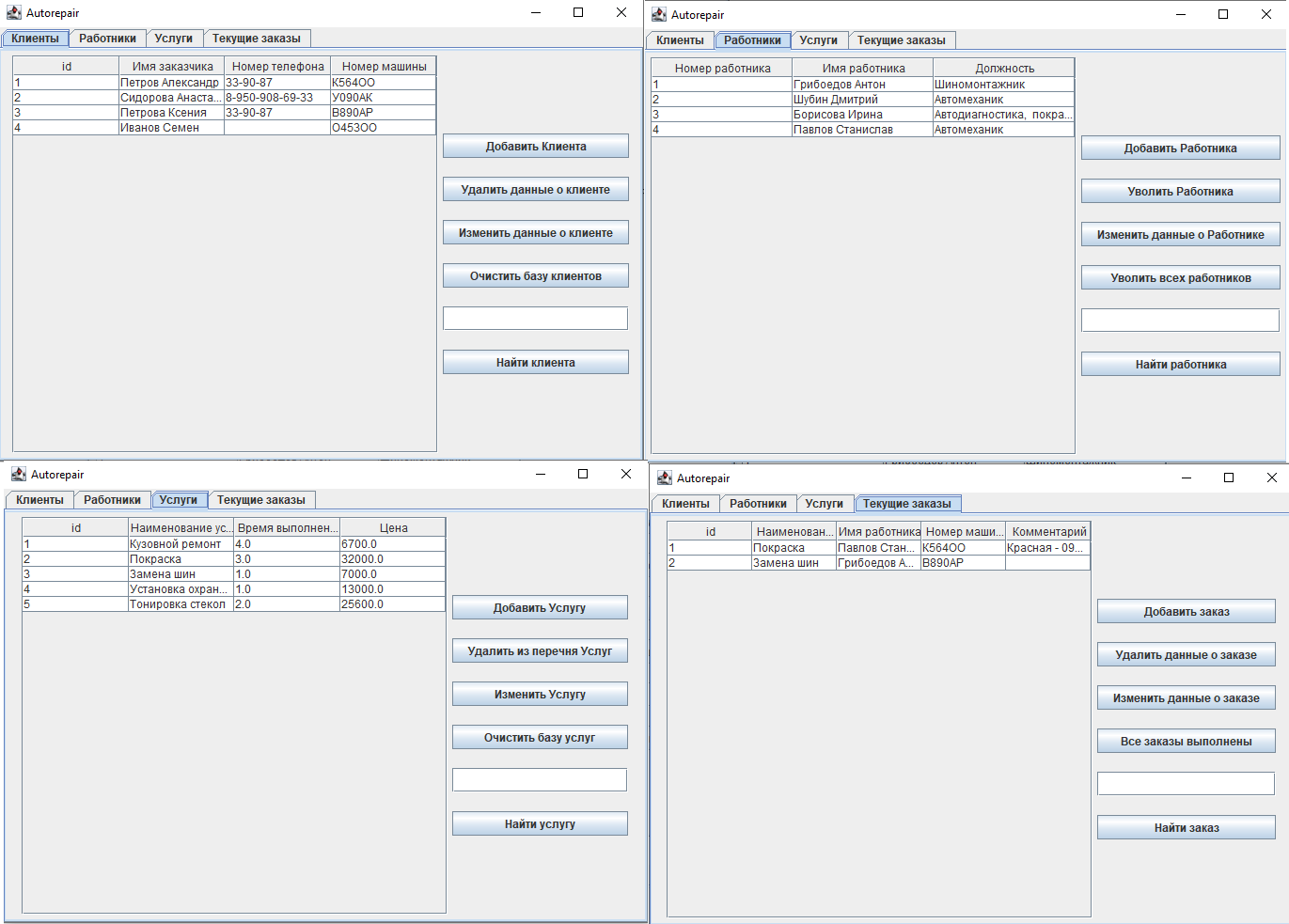


Рисунок 2 – главное окно программы

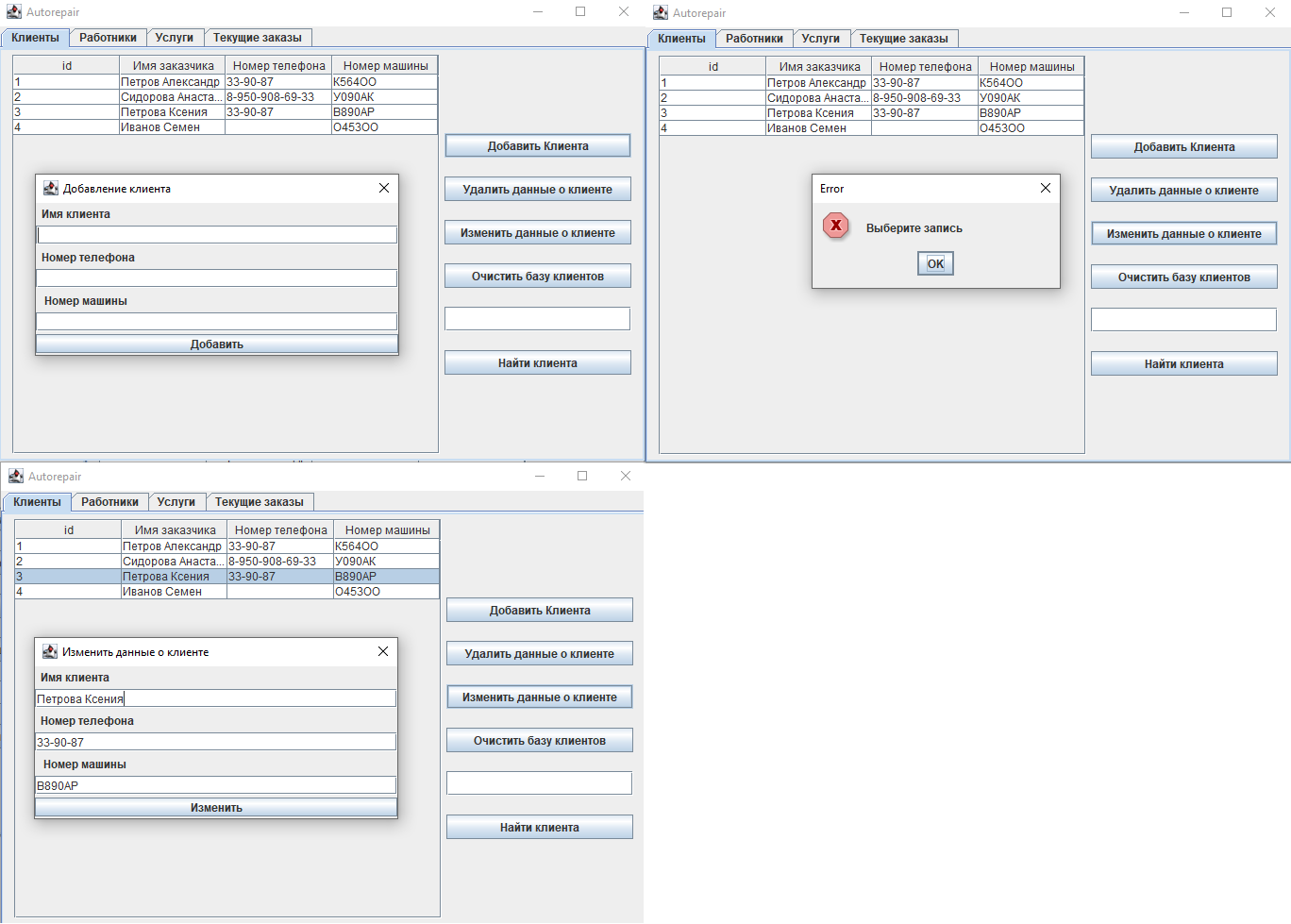


Рисунок 3 – дополнительные окна для добавления/изменения записи о клиенте

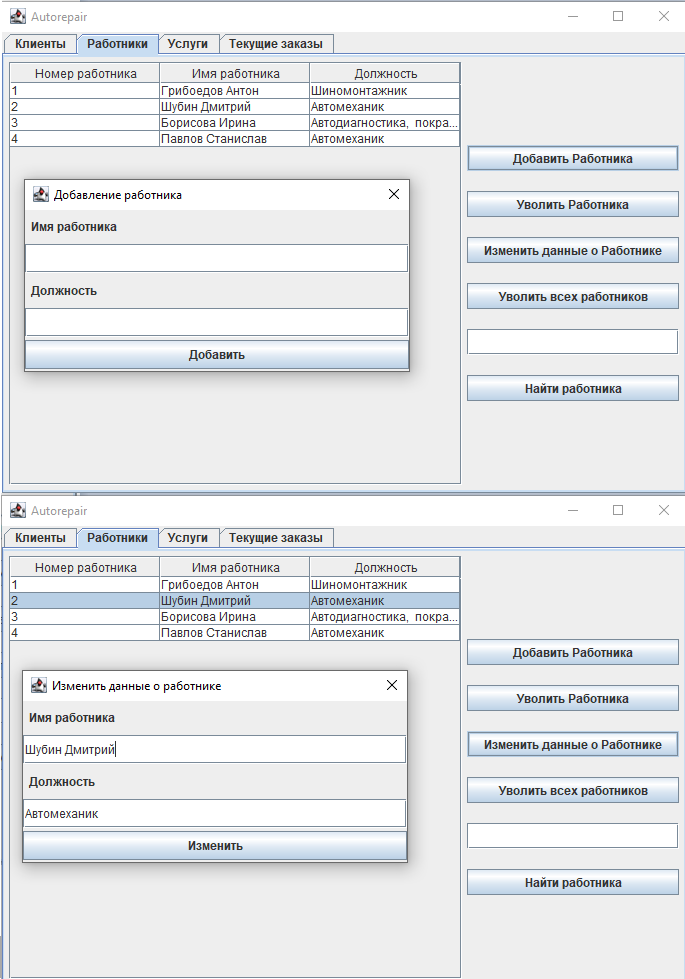


Рисунок 4 - дополнительные окна для добавления/изменения записи о работнике

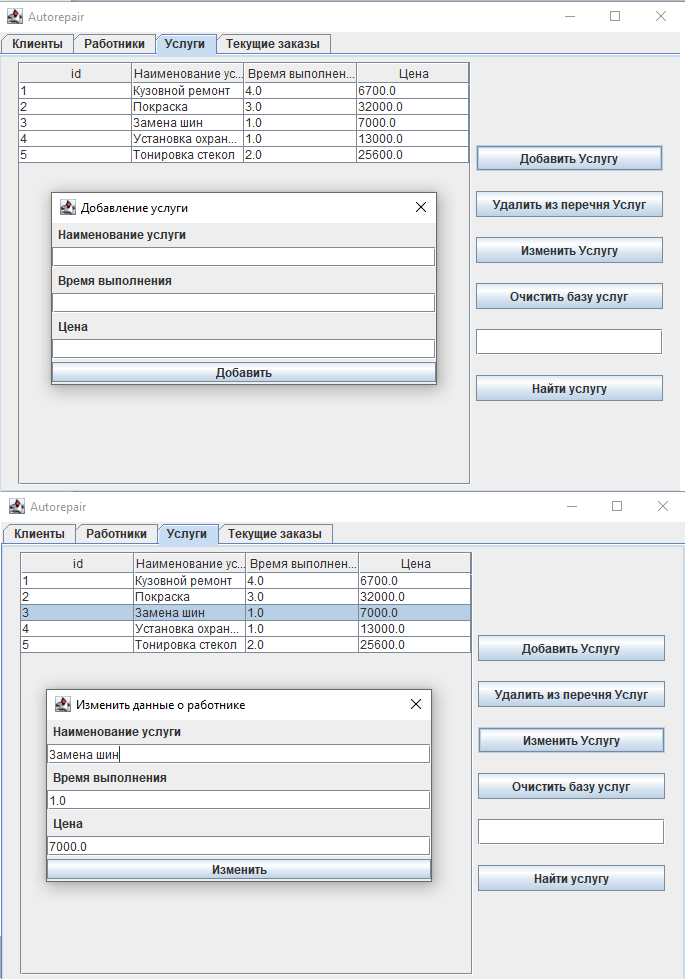


Рисунок 5 - дополнительные окна для добавления/изменения записи об услуге

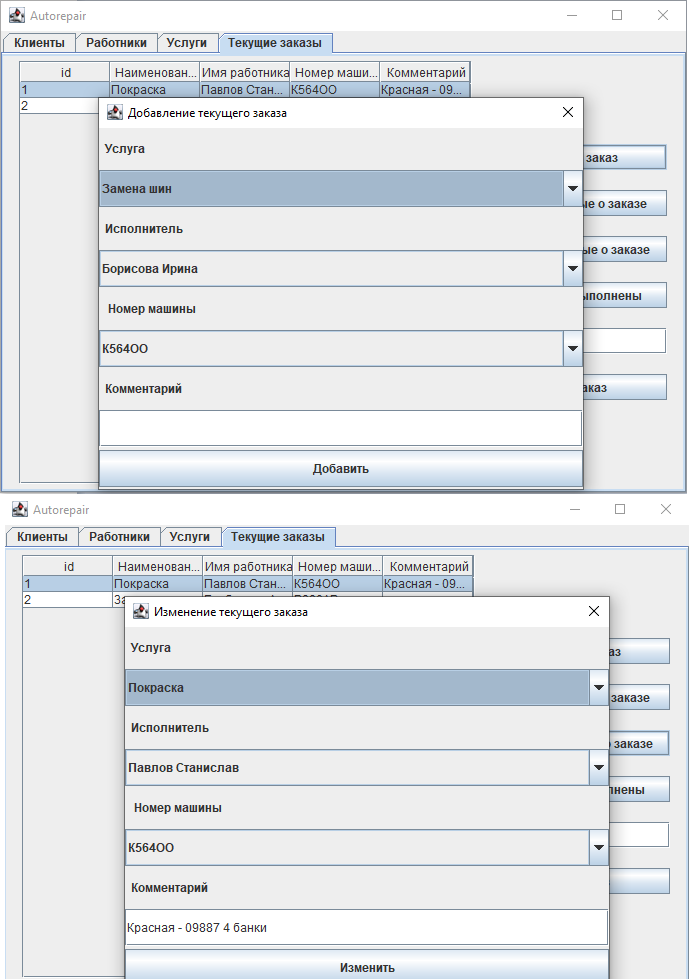


Рисунок 6 - дополнительные окна для добавления/изменения записи о заказе

# 4 Описание структуры классов

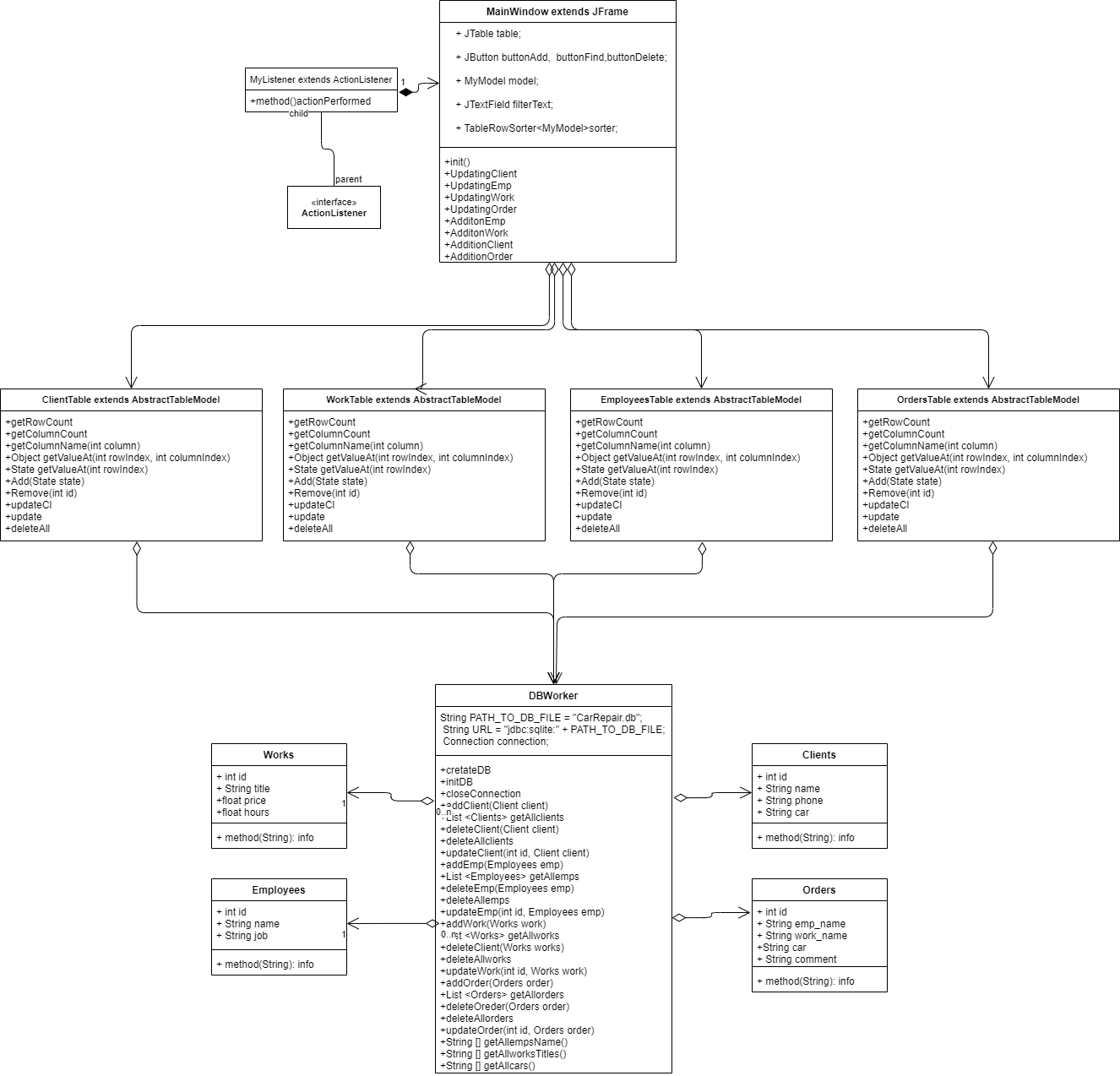


Рисунок 7 – UML диаграмма классов

# Проектирование тестов

В ходе разработки было проведено тестирование программы, результаты которого представлены в табл. 1.

Таблица 1 – Тестирование программы

|  |  |  |
| --- | --- | --- |
| № | Действие | Результат действия |
| 11 | Подключение БД | При старте программы появляется главная форма |
| 22 | Добавление клиента | Появляется форма для добавления  (У клиента обязательно должно быть имя) |
| 33 | Удаление клиента из базы | Удаляется выбранная запись из таблицы |
| 44 | Изменение записи о клиенте | Появляется форма с текущими данными клиента(обязательно надо выбрать запись) |
| 55 | Удаление всех клиентов | Вся таблица с клиентами очищается |
| 66 | Добавление работника | Появляется форма для добавления |
| 77 | Увольнение работника | Удаляется выбранная запись из таблицы |
| 88 | Изменение записи о работнике | Появляется форма с текущими данными работника |
| 99 | Тотальное сокращение | Вся таблица с работниками очищается |
| 110 | Добавление услуги | Появляется форма для добавления |
| 111 | Удаление услуги | Удаляется выбранная запись из таблицы |
| 112 | Изменение данных об услуге | Появляется форма с текущими данными об услуге |
| 113 | Удаление всех услуг | Вся таблица с услугами очищается |
| 114 | Добавление текущего заказа | Появляется форма для добавления |
| 115 | Отмена заказа | Удаляется выбранная запись из таблицы |
| 116 | Изменение заказа | Появляется форма с текущими данными об услуге |
| 117 | Все заказы выполнены | Вся таблица с заказами очищается |
| 118 | Поиск по клиентам | В таблице остаются результаты поиска |

# Результаты тестирования

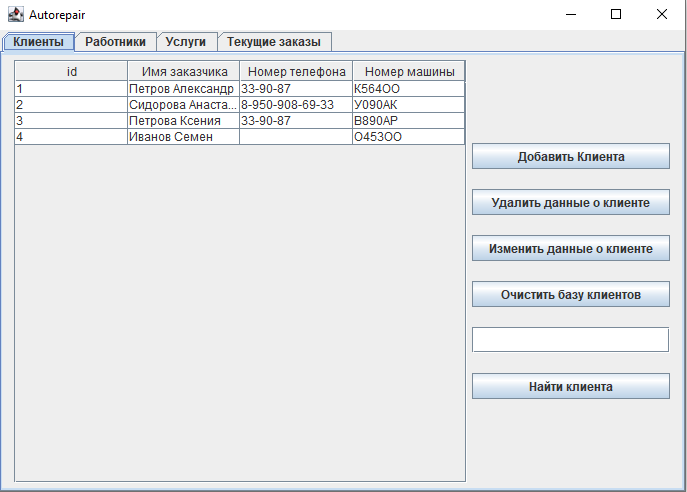


Рисунок 8 – результат теста №1

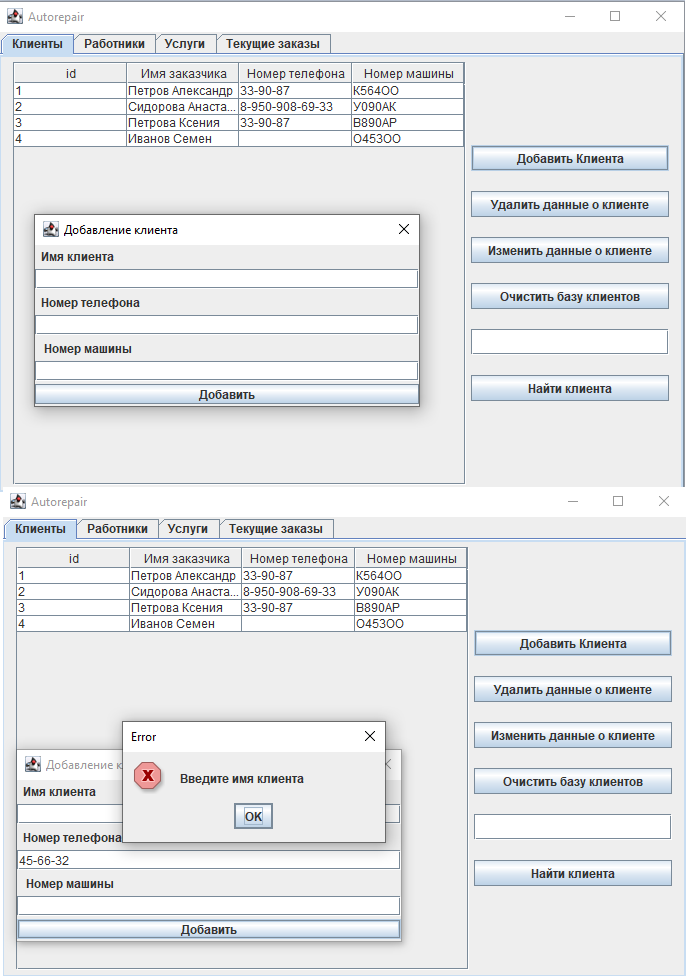


Рисунок 9 – результат теста №2

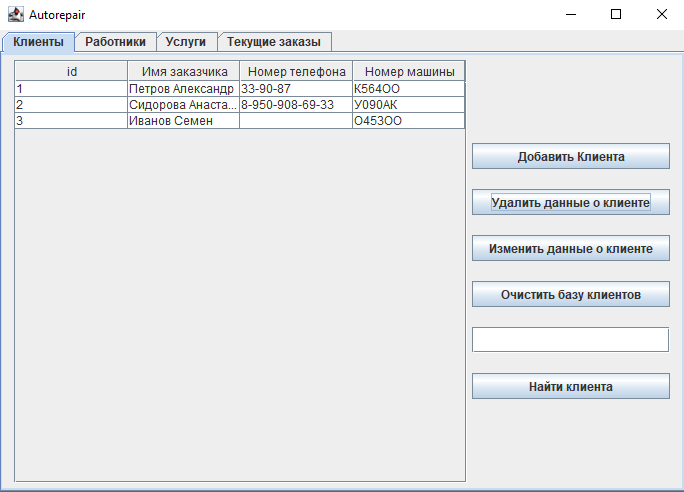


Рисунок 10 – результат теста №3(удален клиент – Петрова Ксения)

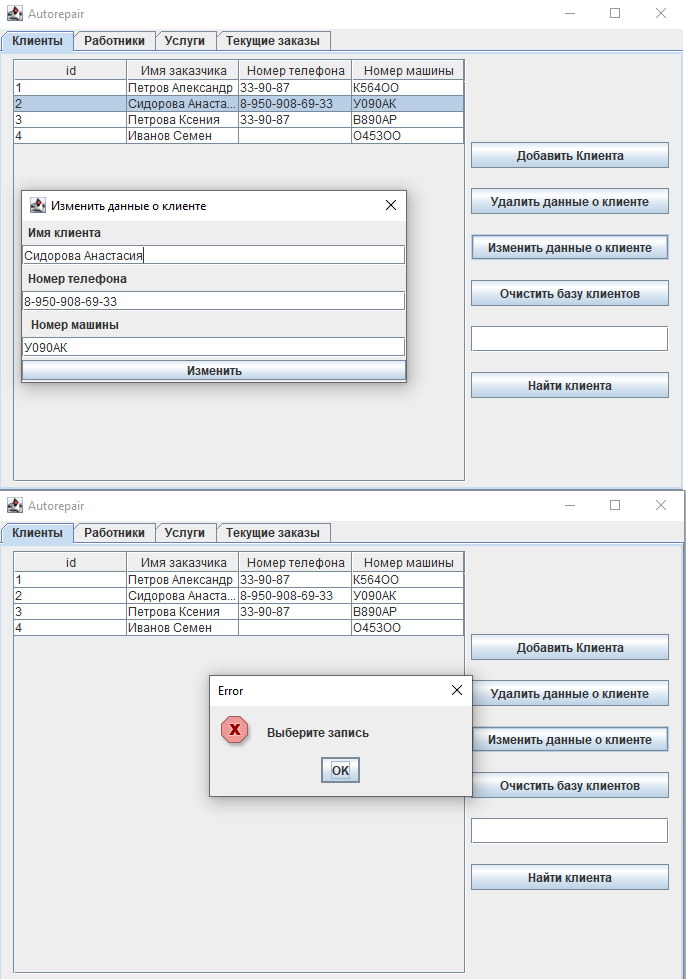


Рисунок 11 – результат теста №4

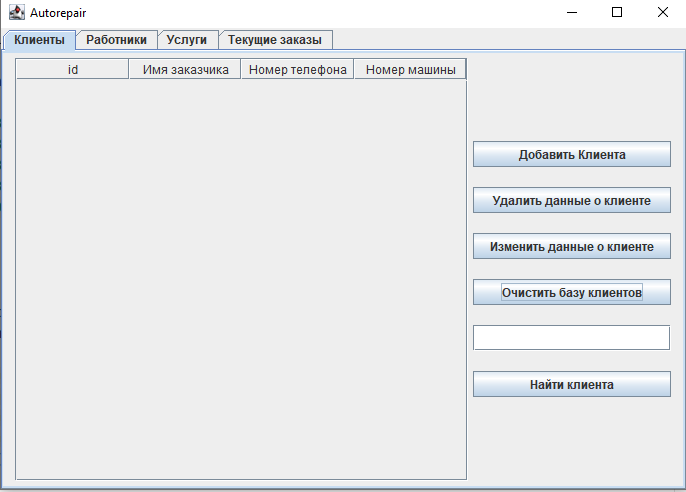


Рисунок 12 – результат теста №5

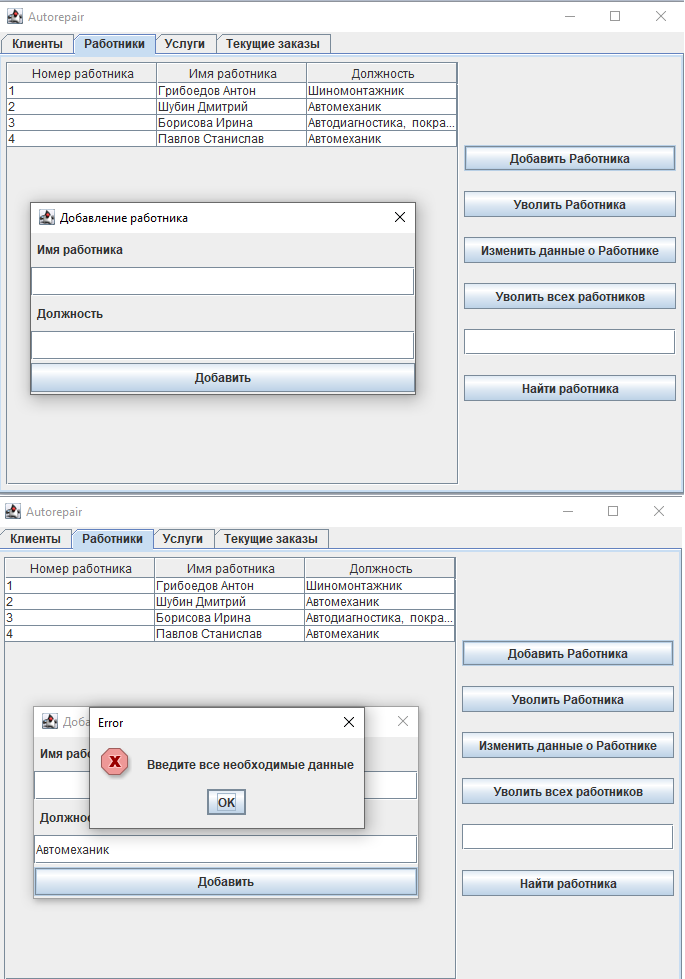


Рисунок 13 – результат теста №6

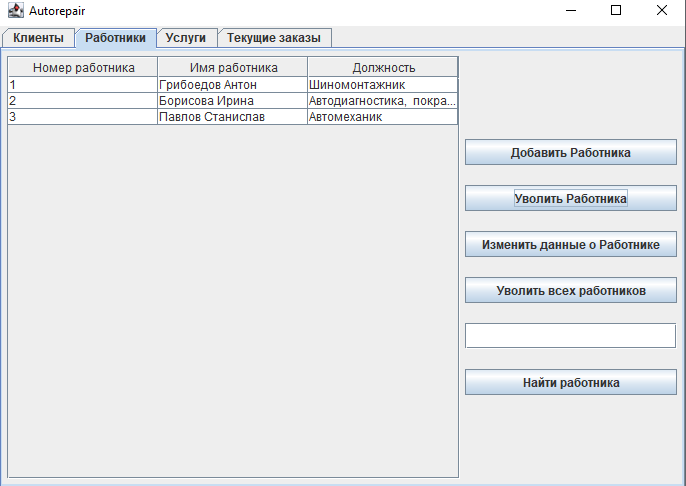


Рисунок 14 – результат теста №7(уволен Шубин Дмитрий)

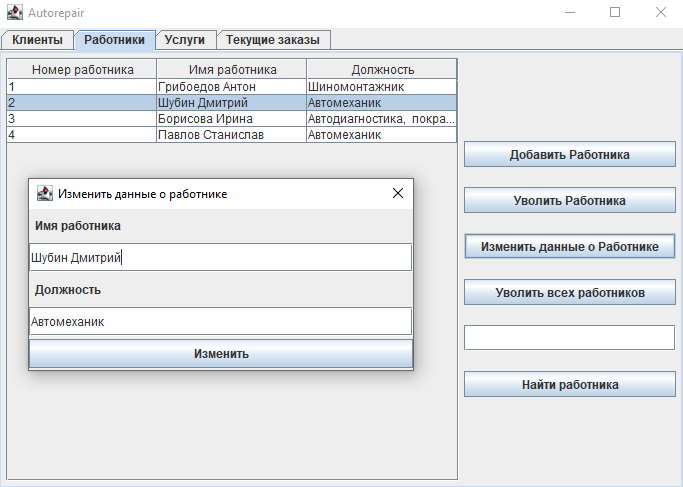


Рисунок 15 – результат теста №8

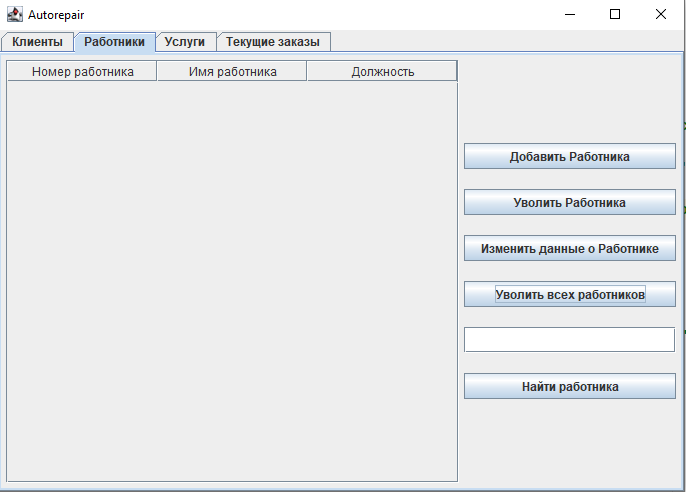


Рисунок 16 – результат теста №9

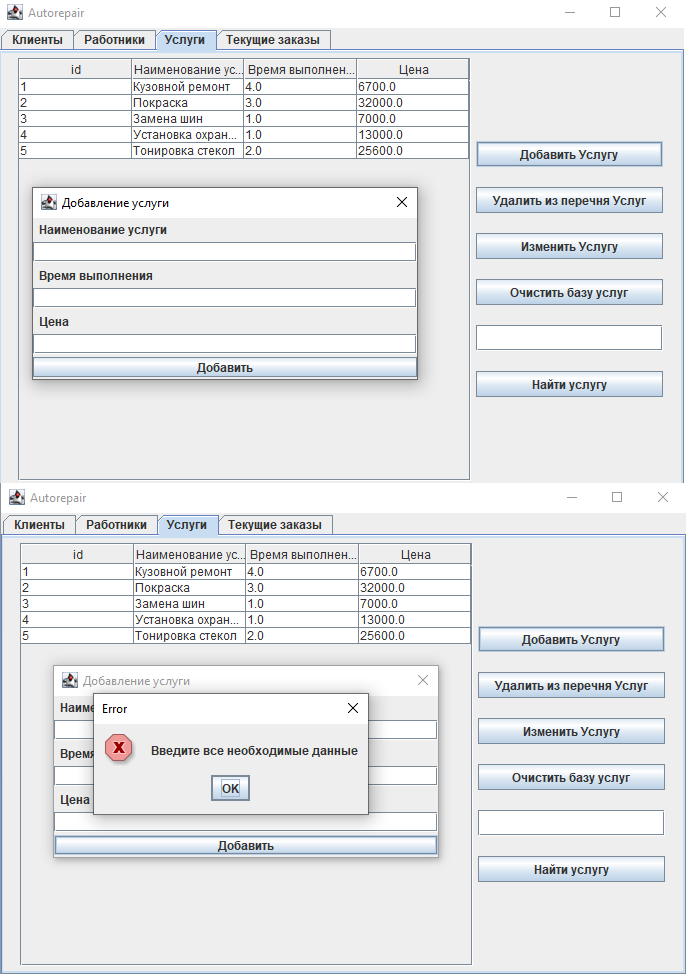


Рисунок 17 – результат теста №10

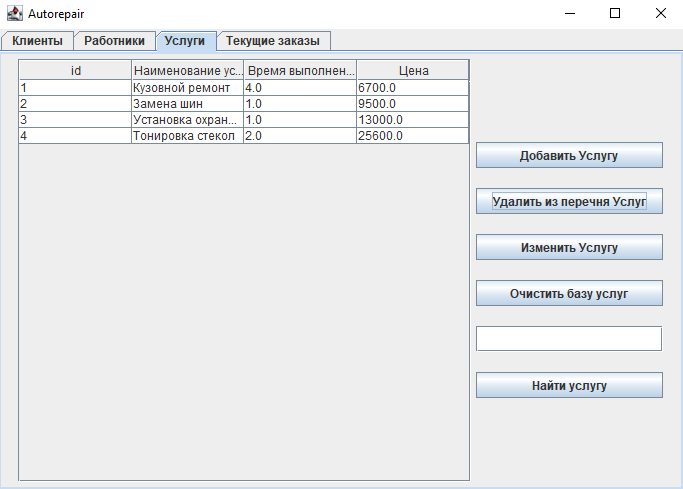


Рисунок 18 – результат теста №11(из перечня удалена Покраска)

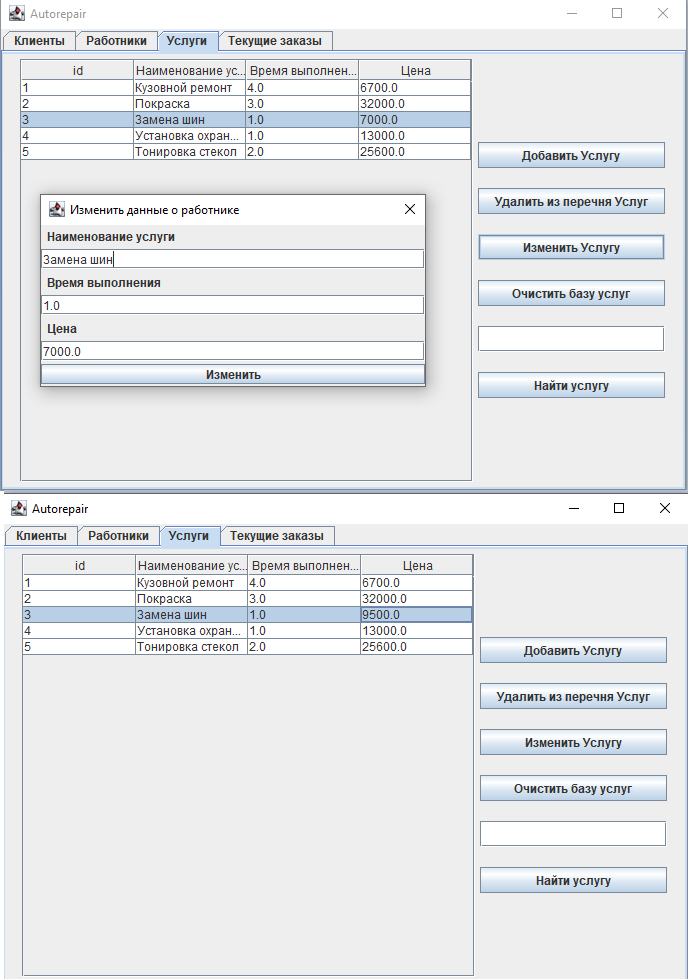


Рисунок 19 – результат теста №12

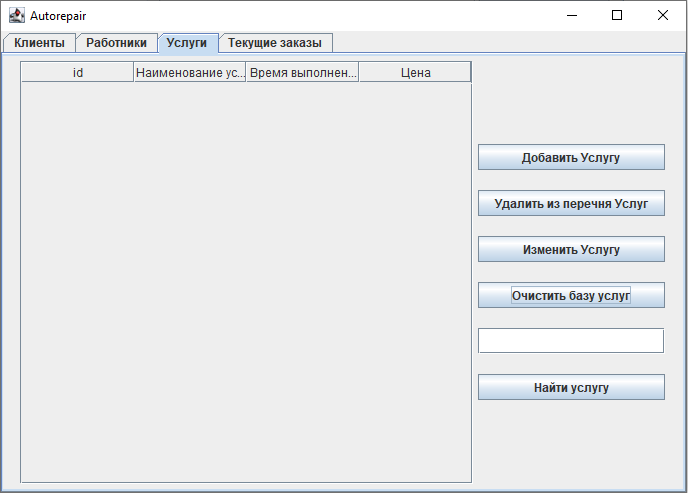


Рисунок 20 – результат теста №13

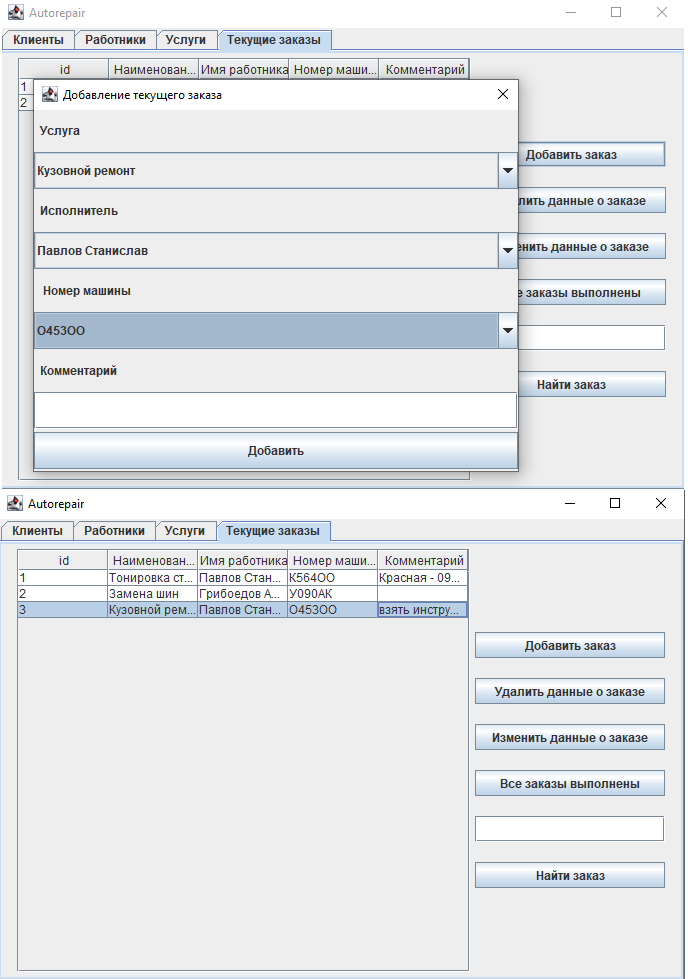


Рисунок 21 – результат теста №14

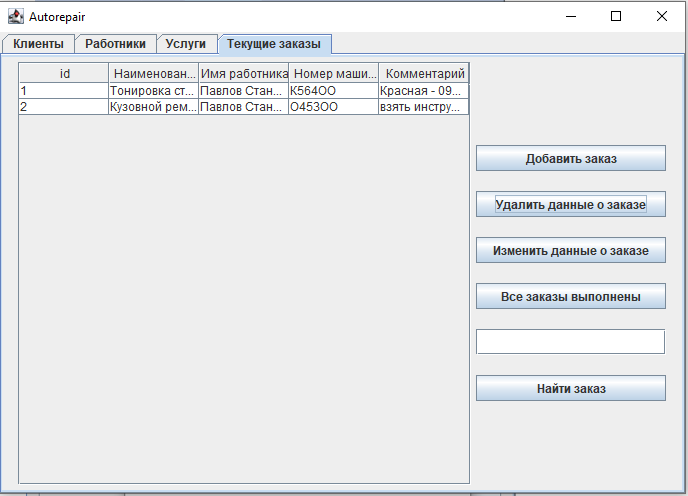


Рисунок 22 – результат теста №15(удален заказ на замену шин)

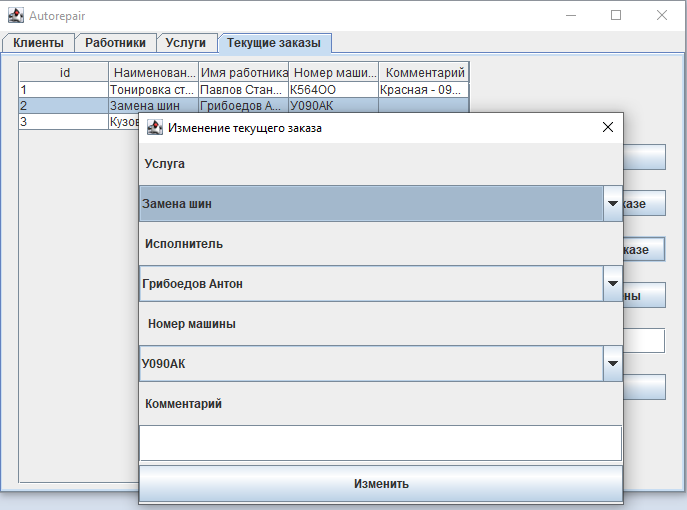


Рисунок 23 – результат теста №16

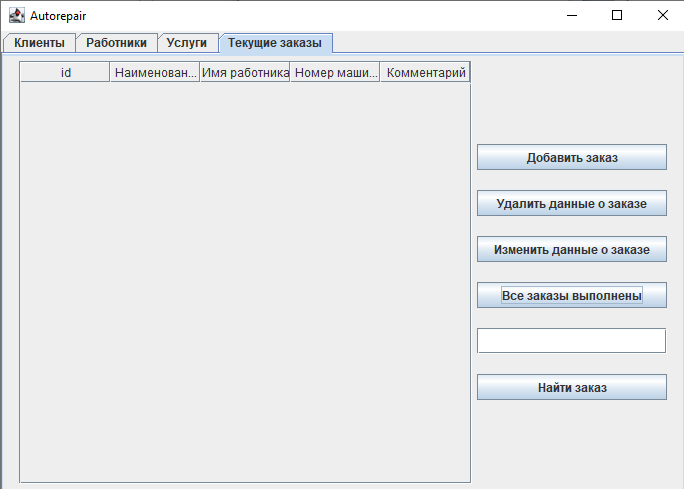


Рисунок 24 – результат теста №17

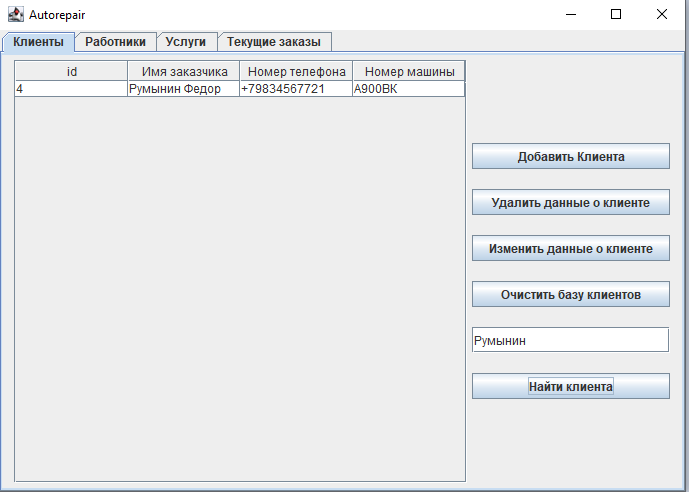


Рисунок 25 – результат теста №18

# 7 Листинг программы

# 7.1 Главный класс

**public class** Main {  
  
 **public static void** main(String[] args) {  
 **new** MainWindow();  
 }  
}

# 7.2 Класс MainWindow

**import** com.company.basicClasses.Clients;  
**import** com.company.basicClasses.Employees;  
**import** com.company.basicClasses.Orders;  
**import** com.company.basicClasses.Works;  
**import** com.company.tables.ClientTable;  
**import** com.company.tables.EmployeesTable;  
**import** com.company.tables.OrdersTable;  
**import** com.company.tables.WorksTable;  
  
**import** javax.swing.\*;  
**import** javax.swing.table.TableRowSorter;  
**import** java.awt.\*;  
**import** java.awt.event.ActionEvent;  
**import** java.awt.event.ActionListener;  
**import** java.sql.SQLException;  
  
**public class** MainWindow **extends** JFrame {  
 **private** JPanel **clients**, **employees**, **works**, **orders**;  
 **private** JPanel **butpanelE**, **butpanelW**,**butpanelC**, **butpanelO**;  
 **private** JTable **tbclients**, **tbemployees**, **tbworks**, **tborders**;  
 **private** JButton **addWork**, **deleteWork**, **changeWork**, **deleteAllWorks**, **findWork**;  
 **private** JButton **addEmp**, **deleteEmp**, **changeEmp**, **deleteAllEmps**, **findEmp**;  
 **private** JButton **addClient**, **deleteClient**, **changeClient**, **deleteAllClients**, **findClient**;  
 **private** JButton **addOrder**, **deleteOrder**, **changeOrder**, **deleteAllOrders**, **findOrder**;  
 **private** WorksTable **wmodel**;  
 **private** EmployeesTable **emodel**;  
 **private** ClientTable **cmodel**;  
 **private** OrdersTable **omodel**;  
 **private** JTextField **filterText1**, **filterText2**, **filterText3**, **filterText4**;  
 **private** TableRowSorter<ClientTable> **sorter1**;  
 **private** TableRowSorter<WorksTable> **sorter2**;  
 **private** TableRowSorter<EmployeesTable> **sorter3**;  
 **private** TableRowSorter<OrdersTable> **sorter4**;  
 **public** MainWindow(){  
 **super**(**"Autorepair"**);  
 setSize(400, 600);  
 **this**.setDefaultCloseOperation(WindowConstants.***EXIT\_ON\_CLOSE***);  
 DBWorker.*initDB*();  
 init();  
 JTabbedPane tabs = **new** JTabbedPane();  
 **clients** = **new** JPanel();  
 **employees** = **new** JPanel();  
 **works**=**new** JPanel();  
 **orders** = **new** JPanel();  
 **try** {  
 **cmodel**=**new** ClientTable();  
 **emodel**=**new** EmployeesTable();  
 **wmodel**=**new** WorksTable();  
 **omodel**=**new** OrdersTable();  
 } **catch** (SQLException throwables) {  
 throwables.printStackTrace();  
 }  
 **sorter1** = **new** TableRowSorter<>(**cmodel**);  
 **sorter2** = **new** TableRowSorter<>(**wmodel**);  
 **sorter3** = **new** TableRowSorter<>(**emodel**);  
 **sorter4** = **new** TableRowSorter<>(**omodel**);  
 **tbclients** = **new** JTable(**cmodel**);  
 **tbemployees**=**new** JTable(**emodel**);  
 **tbworks** =**new** JTable(**wmodel**);  
 **tborders**=**new** JTable(**omodel**);  
 **tbclients**.setRowSorter(**sorter1**);  
 **tbemployees**.setRowSorter(**sorter3**);  
 **tbworks**.setRowSorter(**sorter2**);  
 **tborders**.setRowSorter(**sorter4**);  
 JScrollPane jScrollPaneClients = **new** JScrollPane(**tbclients**);  
 JScrollPane jScrollPaneEmps=**new** JScrollPane(**tbemployees**);  
 JScrollPane jScrollPaneWork=**new** JScrollPane(**tbworks**);  
 JScrollPane jScrollPaneOrder=**new** JScrollPane(**tborders**);  
 **employees**.add(jScrollPaneEmps);  
 **clients**.add(jScrollPaneClients);  
 **works**.add(jScrollPaneWork);  
 **orders**.add(jScrollPaneOrder);  
 **clients**.add(**butpanelC**, BorderLayout.***WEST***);  
 **employees**.add(**butpanelE**,BorderLayout.***WEST***);  
 **works**.add(**butpanelW**, BorderLayout.***WEST***);  
 **orders**.add(**butpanelO**, BorderLayout.***WEST***);  
 tabs.addTab(**"Клиенты"**, **clients**);  
 tabs.addTab(**"Работники"**, **employees**);  
 tabs.addTab(**"Услуги"**, **works**);  
 tabs.addTab(**"Текущие заказы"**, **orders**);  
 **this**.add(tabs);  
  
  
 **this**.pack();  
 **this**.setLocationRelativeTo(**null**);  
 **this**.setVisible(**true**);  
  
 }  
 **private void** init(){  
 **addWork** = **new** JButton(**"Добавить Услугу"**);  
 **addWork**.addActionListener(**new** MyListener());  
 **deleteWork**=**new** JButton(**"Удалить из перечня Услуг"**);  
 **deleteWork**.addActionListener(**new** MyListener());  
 **changeWork**=**new** JButton(**"Изменить Услугу"**);  
 **changeWork**.addActionListener(**new** MyListener());  
 **deleteAllWorks** = **new** JButton(**"Очистить базу услуг"**);  
 **deleteAllWorks**.addActionListener(**new** MyListener());  
 **findWork**= **new** JButton(**"Найти услугу"**);  
 **findWork**.addActionListener(**new** MyListener());  
 **filterText1** = **new** JTextField(10);  
 **addEmp** = **new** JButton(**"Добавить Работника"**);  
 **addEmp**.addActionListener(**new** MyListener());  
 **deleteEmp**=**new** JButton(**"Уволить Работника"**);  
 **deleteEmp**.addActionListener(**new** MyListener());  
 **changeEmp**=**new** JButton(**"Изменить данные о Работнике"**);  
 **changeEmp**.addActionListener(**new** MyListener());  
 **deleteAllEmps** = **new** JButton(**"Уволить всех работников"**);  
 **deleteAllEmps**.addActionListener(**new** MyListener());  
 **findEmp**= **new** JButton(**"Найти работника"**);  
 **findEmp**.addActionListener(**new** MyListener());  
 **filterText2** = **new** JTextField(10);  
 **addClient** = **new** JButton(**"Добавить Клиента"**);  
 **addClient**.addActionListener(**new** MyListener());  
 **deleteClient**=**new** JButton(**"Удалить данные о клиенте"**);  
 **deleteClient**.addActionListener(**new** MyListener());  
 **changeClient**=**new** JButton(**"Изменить данные о клиенте"**);  
 **changeClient**.addActionListener(**new** MyListener());  
 **deleteAllClients**=**new** JButton(**"Очистить базу клиентов"**);  
 **deleteAllClients**.addActionListener(**new** MyListener());  
 **findClient**= **new** JButton(**"Найти клиента"**);  
 **findClient**.addActionListener(**new** MyListener());  
 **filterText3** = **new** JTextField(10);  
 **addOrder** = **new** JButton(**"Добавить заказ"**);  
 **addOrder**.addActionListener(**new** MyListener());  
 **deleteOrder**=**new** JButton(**"Удалить данные о заказе"**);  
 **deleteOrder**.addActionListener(**new** MyListener());  
 **changeOrder**=**new** JButton(**"Изменить данные о заказе"**);  
 **changeOrder**.addActionListener(**new** MyListener());  
 **deleteAllOrders**=**new** JButton(**"Все заказы выполнены"**);  
 **deleteAllOrders**.addActionListener(**new** MyListener());  
 **findOrder**= **new** JButton(**"Найти заказ"**);  
 **findOrder**.addActionListener(**new** MyListener());  
 **filterText4** = **new** JTextField(10);  
 GridLayout gl=**new** GridLayout(6,1,3,20);  
 **butpanelW** = **new** JPanel(gl);  
 **butpanelW**.add(**addWork**);  
 **butpanelW**.add(**deleteWork**);  
 **butpanelW**.add(**changeWork**);  
 **butpanelW**.add(**deleteAllWorks**);  
 **butpanelW**.add(**filterText1**);  
 **butpanelW**.add(**findWork**);  
 **butpanelE** = **new** JPanel(gl);  
 **butpanelE**.add(**addEmp**);  
 **butpanelE**.add(**deleteEmp**);  
 **butpanelE**.add(**changeEmp**);  
 **butpanelE**.add(**deleteAllEmps**);  
 **butpanelE**.add(**filterText2**);  
 **butpanelE**.add(**findEmp**);  
 **butpanelC** = **new** JPanel(gl);  
 **butpanelC**.add(**addClient**);  
 **butpanelC**.add(**deleteClient**);  
 **butpanelC**.add(**changeClient**);  
 **butpanelC**.add(**deleteAllClients**);  
 **butpanelC**.add(**filterText3**);  
 **butpanelC**.add(**findClient**);  
 **butpanelO**=**new** JPanel(gl);  
 **butpanelO**.add(**addOrder**);  
 **butpanelO**.add(**deleteOrder**);  
 **butpanelO**.add(**changeOrder**);  
 **butpanelO**.add(**deleteAllOrders**);  
 **butpanelO**.add(**filterText4**);  
 **butpanelO**.add(**findOrder**);  
 }  
 **class** MyListener **implements** ActionListener{  
  
 @Override  
 **public void** actionPerformed(ActionEvent e) {  
 **if** (e.getActionCommand().equals(**"Добавить Клиента"**)) {  
 AdditionClient();  
 }  
 **else if** (e.getActionCommand().equals(**"Удалить данные о клиенте"**)) {  
 **try** { **int** id=**tbclients**.getSelectedRow();  
 **cmodel**.remove(id);  
 } **catch** (IndexOutOfBoundsException ex) {  
 JOptionPane.*showMessageDialog*(MainWindow.**this**, **"Выберите запись"**, **"Error"**, JOptionPane.***ERROR\_MESSAGE***);  
 } **catch** (SQLException throwables) {  
 throwables.printStackTrace();  
 }  
 }  
 **else if** (e.getActionCommand().equals(**"Очистить базу клиентов"**)) {  
 **try** {  
 **cmodel**.deleteAll();  
 } **catch** (IndexOutOfBoundsException ex) {  
 JOptionPane.*showMessageDialog*(MainWindow.**this**, **"Выберите запись"**, **"Error"**, JOptionPane.***ERROR\_MESSAGE***);  
 } **catch** (SQLException throwables) {  
 throwables.printStackTrace();  
 }  
 }  
 **else if** (e.getActionCommand().equals(**"Изменить данные о клиенте"**)) {  
 **try** { UpdatingClient();  
 } **catch** (IndexOutOfBoundsException ex) {  
 JOptionPane.*showMessageDialog*(MainWindow.**this**, **"Выберите запись"**, **"Error"**, JOptionPane.***ERROR\_MESSAGE***);  
 }  
 }  
 **else if** (e.getActionCommand().equals(**"Добавить Работника"**)) {  
 AdditionEmp();  
 }  
 **else if** (e.getActionCommand().equals(**"Уволить Работника"**)) {  
 **try** { **int** id=**tbemployees**.getSelectedRow();  
 **emodel**.remove(id);  
 } **catch** (IndexOutOfBoundsException ex) {  
 JOptionPane.*showMessageDialog*(MainWindow.**this**, **"Выберите запись"**, **"Error"**, JOptionPane.***ERROR\_MESSAGE***);  
 } **catch** (SQLException throwables) {  
 throwables.printStackTrace();  
 }  
 }  
 **else if** (e.getActionCommand().equals(**"Изменить данные о Работнике"**)) {  
 **try** { UpdatingEmps();  
 } **catch** (IndexOutOfBoundsException ex) {  
 JOptionPane.*showMessageDialog*(MainWindow.**this**, **"Выберите запись"**, **"Error"**, JOptionPane.***ERROR\_MESSAGE***);  
 }  
 }  
 **else if** (e.getActionCommand().equals(**"Уволить всех работников"**)) {  
 **try** { **emodel**.deleteAll();  
 } **catch** (IndexOutOfBoundsException ex) {  
 JOptionPane.*showMessageDialog*(MainWindow.**this**, **"Выберите запись"**, **"Error"**, JOptionPane.***ERROR\_MESSAGE***);  
 } **catch** (SQLException throwables) {  
 throwables.printStackTrace();  
 }  
 }  
 **else if** (e.getActionCommand().equals(**"Добавить Услугу"**)) {  
 AdditionWork();  
 }  
 **else if** (e.getActionCommand().equals(**"Удалить из перечня Услуг"**)) {  
 **try** { **int** id=**tbworks**.getSelectedRow();  
 **wmodel**.remove(id);  
 } **catch** (IndexOutOfBoundsException ex) {  
 JOptionPane.*showMessageDialog*(MainWindow.**this**, **"Выберите запись"**, **"Error"**, JOptionPane.***ERROR\_MESSAGE***);  
 } **catch** (SQLException throwables) {  
 throwables.printStackTrace();  
 }  
 }  
 **else if** (e.getActionCommand().equals(**"Изменить Услугу"**)) {  
 **try** { UpdatingWork();  
 } **catch** (IndexOutOfBoundsException ex) {  
 JOptionPane.*showMessageDialog*(MainWindow.**this**, **"Выберите запись"**, **"Error"**, JOptionPane.***ERROR\_MESSAGE***);  
 }  
 }  
 **else if** (e.getActionCommand().equals(**"Очистить базу услуг"**)) {  
 **try** {  
 **wmodel**.deleteAll();  
 } **catch** (IndexOutOfBoundsException ex) {  
 JOptionPane.*showMessageDialog*(MainWindow.**this**, **"Выберите запись"**, **"Error"**, JOptionPane.***ERROR\_MESSAGE***);  
 } **catch** (SQLException throwables) {  
 throwables.printStackTrace();  
 }  
 }  
 **else if** (e.getActionCommand().equals(**"Добавить заказ"**)) {  
 **try** {  
 AdditionOrder();  
 } **catch** (SQLException throwables) {  
 throwables.printStackTrace();  
 }  
 }  
 **else if** (e.getActionCommand().equals(**"Удалить данные о заказе"**)) {  
 **try** { **int** id=**tborders**.getSelectedRow();  
 **omodel**.remove(id);  
 } **catch** (IndexOutOfBoundsException ex) {  
 JOptionPane.*showMessageDialog*(MainWindow.**this**, **"Выберите запись"**, **"Error"**, JOptionPane.***ERROR\_MESSAGE***);  
 } **catch** (SQLException throwables) {  
 throwables.printStackTrace();  
 }  
 }  
 **else if** (e.getActionCommand().equals(**"Все заказы выполнены"**)) {  
 **try** {  
 **omodel**.deleteAll();  
 } **catch** (SQLException throwables) {  
 throwables.printStackTrace();  
 }  
 }  
 **else if** (e.getActionCommand().equals(**"Изменить данные о заказе"**)) {  
 **try** { UpdatingOrder();  
 } **catch** (IndexOutOfBoundsException | SQLException ex) {  
 JOptionPane.*showMessageDialog*(MainWindow.**this**, **"Выберите запись"**, **"Error"**, JOptionPane.***ERROR\_MESSAGE***);  
 }  
 }  
 **else if** (e.getActionCommand().equals(**"Найти клиента"**)) {  
 String text = **filterText3**.getText();  
 **if** (text.length() == 0) {  
 **sorter1**.setRowFilter(**null**);  
 } **else** {  
 **sorter1**.setRowFilter(RowFilter.*regexFilter*(text));  
 }  
 }  
 **else if** (e.getActionCommand().equals(**"Найти услугу"**)) {  
 String text = **filterText1**.getText();  
 **if** (text.length() == 0) {  
 **sorter2**.setRowFilter(**null**);  
 } **else** {  
 **sorter2**.setRowFilter(RowFilter.*regexFilter*(text));  
 }  
 }  
 **else if** (e.getActionCommand().equals(**"Найти работника"**)) {  
 String text = **filterText2**.getText();  
 **if** (text.length() == 0) {  
 **sorter3**.setRowFilter(**null**);  
 } **else** {  
 **sorter3**.setRowFilter(RowFilter.*regexFilter*(text));  
 }  
 }  
 **else if** (e.getActionCommand().equals(**"Найти заказ"**)) {  
 String text = **filterText4**.getText();  
 **if** (text.length() == 0) {  
 **sorter4**.setRowFilter(**null**);  
 } **else** {  
 **sorter4**.setRowFilter(RowFilter.*regexFilter*(text));  
 }  
 }  
 }  
 **public void** AdditionClient(){  
 JDialog dialog = **new** JDialog();  
 dialog.setModal(**true**);  
 dialog.setLocation(600,500);  
 dialog.setTitle(**"Добавление клиента"**);  
 dialog.setSize(400,200);  
 GridLayout glA = **new** GridLayout(7,2,3,3 );  
 JPanel add=**new** JPanel(glA);  
 JLabel name=**new** JLabel(**" Имя клиента"**);  
 JLabel phone=**new** JLabel(**" Номер телефона"**);  
 JLabel car= **new** JLabel(**" Номер машины"**);  
 JTextField Name = **new** JTextField(50);  
 JTextField Phone = **new** JTextField(20);  
 JTextField Car = **new** JTextField(10);  
 JButton Add = **new** JButton(**"Добавить"**);  
 add.add(name); add.add(Name); add.add(phone);add.add(Phone); add.add(car); add.add(Car);  
 add.add(Add);  
 Add.addActionListener(**new** ActionListener() {  
 @Override  
 **public void** actionPerformed(ActionEvent e) {  
  
 **try** { **int** id=**tbclients**.getRowCount()+1;  
 **if** (Name.getText().equals(**""**)){  
 JOptionPane.*showMessageDialog*(**null**, **"Введите имя клиента"**, **"Error"**, JOptionPane.***ERROR\_MESSAGE***);  
 }  
 **else** {  
 **cmodel**.add(**new** Clients(id, Name.getText(), Phone.getText(), Car.getText()));  
 Name.setText(**null**);  
 Phone.setText(**null**);  
 Car.setText(**null**);  
 }  
  
 } **catch** (SQLException throwables) {  
 throwables.printStackTrace();  
 }  
 }  
 });  
 dialog.add(add);  
 dialog.setVisible(**true**);  
 }  
 **public void** UpdatingClient(){  
 JDialog dialog = **new** JDialog();  
 dialog.setModal(**true**);  
 dialog.setLocation(500,400);  
 dialog.setTitle(**"Изменить данные о клиенте"**);  
 dialog.setSize(400,200);  
 GridLayout glA = **new** GridLayout(7,2,3,3 );  
 JPanel add=**new** JPanel(glA);  
 Clients clients = **cmodel**.getClient(**tbclients**.getSelectedRow());  
 JLabel name=**new** JLabel(**" Имя клиента"**);  
 JLabel phone=**new** JLabel(**" Номер телефона"**);  
 JLabel car= **new** JLabel(**" Номер машины"**);  
 JTextField Name = **new** JTextField(50); Name.setText(clients.getName());  
 JTextField Phone = **new** JTextField(20); Phone.setText(clients.getPhone());  
 JTextField Car = **new** JTextField(10); Car.setText(clients.getCar());  
 JButton Upd = **new** JButton(**"Изменить"**);  
 add.add(name); add.add(Name); add.add(phone);add.add(Phone); add.add(car); add.add(Car);  
 add.add(Upd);  
 Upd.addActionListener(**new** ActionListener() {  
 @Override  
 **public void** actionPerformed(ActionEvent e) {  
  
 **try** { **int** id = **tbclients**.getSelectedRow();  
 **if** (Name.getText().equals(**""**)){  
 JOptionPane.*showMessageDialog*(MainWindow.**this**, **"Введите имя клиента"**, **"Error"**, JOptionPane.***ERROR\_MESSAGE***);  
 }  
 **else** { **cmodel**.updateCl(id+1, **new** Clients(id,Name.getText(), Phone.getText(), Car.getText()));  
 }  
 } **catch** (SQLException throwables) {  
 throwables.printStackTrace();  
 }  
 }  
 });  
 dialog.add(add);  
 dialog.setVisible(**true**);  
 }  
 **public void** AdditionEmp(){  
 JDialog dialog = **new** JDialog();  
 dialog.setModal(**true**);  
 dialog.setLocation(500,400);  
 dialog.setTitle(**"Добавление работника"**);  
 dialog.setSize(400,200);  
 GridLayout glA = **new** GridLayout(5,2,3,3 );  
 JPanel add=**new** JPanel(glA);  
 JLabel name=**new** JLabel(**" Имя работника"**);  
 JLabel job=**new** JLabel(**" Должность"**);  
 JTextField Name = **new** JTextField(50);  
 JTextField Job = **new** JTextField(70);  
  
 JButton Add = **new** JButton(**"Добавить"**);  
 add.add(name); add.add(Name); add.add(job);add.add(Job);  
 add.add(Add);  
 Add.addActionListener(**new** ActionListener() {  
 @Override  
 **public void** actionPerformed(ActionEvent e) {  
  
 **try** { **int** id=**tbemployees**.getRowCount()+1;  
 **if** (Name.getText().equals(**""**)){  
 JOptionPane.*showMessageDialog*(**null**, **"Введите все необходимые данные"**, **"Error"**, JOptionPane.***ERROR\_MESSAGE***);  
 }  
 **else if**(Job.getText().equals(**""**)) {  
 JOptionPane.*showMessageDialog*(**null**, **"Введите все необходимые данные"**, **"Error"**, JOptionPane.***ERROR\_MESSAGE***);  
 }  
 **else**{**emodel**.add(**new** Employees(id, Name.getText(), Job.getText()));  
 Name.setText(**null**);  
 Job.setText(**null**);}  
 } **catch** (SQLException throwables) {  
 throwables.printStackTrace();  
 }  
 }  
 });  
 dialog.add(add);  
 dialog.setVisible(**true**);  
 }  
 **public void** UpdatingEmps(){  
 JDialog dialog = **new** JDialog();  
 dialog.setModal(**true**);  
 dialog.setLocation(500,400);  
 dialog.setTitle(**"Изменить данные о работнике"**);  
 dialog.setSize(400,200);  
 GridLayout glA = **new** GridLayout(5,2,3,3 );  
 JPanel add=**new** JPanel(glA);  
 Employees employees=**emodel**.getEmp(**tbemployees**.getSelectedRow());  
 JLabel name=**new** JLabel(**" Имя работника"**);  
 JLabel job=**new** JLabel(**" Должность"**);  
 JTextField Name = **new** JTextField(50); Name.setText(employees.getName());  
 JTextField Job = **new** JTextField(20); Job.setText(employees.getJob());  
  
 JButton Upd = **new** JButton(**"Изменить"**);  
 add.add(name); add.add(Name); add.add(job);add.add(Job);  
 add.add(Upd);  
 Upd.addActionListener(**new** ActionListener() {  
 @Override  
 **public void** actionPerformed(ActionEvent e) {  
  
 **try** { **int** id = **tbemployees**.getSelectedRow();  
 **if** (Name.getText().equals(**""**)){  
 JOptionPane.*showMessageDialog*(**null**, **"Введите все необходимые данные"**, **"Error"**, JOptionPane.***ERROR\_MESSAGE***);  
 }  
 **else if**(Job.getText().equals(**""**)) {  
 JOptionPane.*showMessageDialog*(**null**, **"Введите все необходимые данные"**, **"Error"**, JOptionPane.***ERROR\_MESSAGE***);  
 }  
  
 **else** {  
 **emodel**.updateEm(id+1,**new** Employees(id,Name.getText(), Job.getText()));  
 }  
 } **catch** (SQLException throwables) {  
 throwables.printStackTrace();  
 }  
 }  
 });  
 dialog.add(add);  
 dialog.setVisible(**true**);  
 }  
 **public void** AdditionWork(){  
 JDialog dialog = **new** JDialog();  
 dialog.setModal(**true**);  
 dialog.setLocation(700,400);  
 dialog.setTitle(**"Добавление услуги"**);  
 dialog.setSize(400,200);  
 GridLayout glA = **new** GridLayout(7,2,3,3 );  
 JPanel add=**new** JPanel(glA);  
 JLabel title=**new** JLabel(**" Наименование услуги"**);  
 JLabel hours=**new** JLabel(**" Время выполнения"**);  
 JLabel price=**new** JLabel(**" Цена"**);  
 JTextField Title = **new** JTextField(50);  
 JTextField Hours = **new** JTextField(70);  
 JTextField Price = **new** JTextField(70);  
 JButton Add = **new** JButton(**"Добавить"**);  
 add.add(title); add.add(Title); add.add(hours);add.add(Hours);add.add(price);add.add(Price);  
 add.add(Add);  
 Add.addActionListener(**new** ActionListener() {  
 @Override  
 **public void** actionPerformed(ActionEvent e) {  
  
 **try** { **int** id=**tbworks**.getRowCount()+1;  
 **if**((Title.getText().equals(**""**))||(Hours.getText().equals(**""**))||(Price.getText().equals(**""**))){  
 JOptionPane.*showMessageDialog*(**null**, **"Введите все необходимые данные"**, **"Error"**, JOptionPane.***ERROR\_MESSAGE***);  
 }  
 **else**{  
 **try**{  
 Float.*parseFloat*(Hours.getText());  
 Float.*parseFloat*(Price.getText());  
 **wmodel**.add(**new** Works(id,Title.getText(),Float.*parseFloat*(Hours.getText()), Float.*parseFloat*(Price.getText())));  
 Title.setText(**null**);  
 Hours.setText(**null**);  
 Price.setText(**null**);  
 }  
 **catch** (NumberFormatException ex){JOptionPane.*showMessageDialog*(**null**, **"Введите числа"**, **"Error"**, JOptionPane.***ERROR\_MESSAGE***);}  
 }  
  
 } **catch** (SQLException throwables) {  
 throwables.printStackTrace();  
 }  
 }  
 });  
 dialog.add(add);  
 dialog.setVisible(**true**);  
 }  
 **public void** UpdatingWork(){  
 JDialog dialog = **new** JDialog();  
 dialog.setModal(**true**);  
 dialog.setLocation(500,400);  
 dialog.setTitle(**"Изменить данные о работнике"**);  
 dialog.setSize(400,200);  
 GridLayout glA = **new** GridLayout(7,2,3,3 );  
 JPanel add=**new** JPanel(glA);  
 Works works=**wmodel**.getWork(**tbworks**.getSelectedRow());  
 JLabel title=**new** JLabel(**" Наименование услуги"**);  
 JLabel hours=**new** JLabel(**" Время выполнения"**);  
 JLabel price=**new** JLabel(**" Цена"**);  
 JTextField Title = **new** JTextField(50); Title.setText(works.getTitle());  
 JTextField Hours = **new** JTextField(70); Hours.setText(String.*valueOf*(works.getHours()));  
 JTextField Price = **new** JTextField(70); Price.setText(String.*valueOf*(works.getPrice()));  
 JButton Upd = **new** JButton(**"Изменить"**);  
 add.add(title); add.add(Title); add.add(hours);add.add(Hours);add.add(price);add.add(Price);  
 add.add(Upd);  
 Upd.addActionListener(**new** ActionListener() {  
 @Override  
 **public void** actionPerformed(ActionEvent e) {  
  
 **try** { **int** id = **tbworks**.getSelectedRow();  
 **if**((Title.getText().equals(**""**))||(Hours.getText().equals(**""**))||(Price.getText().equals(**""**))){  
 JOptionPane.*showMessageDialog*(**null**, **"Введите все необходимые данные"**, **"Error"**, JOptionPane.***ERROR\_MESSAGE***);  
 }  
 **else**{  
 **try**{  
 Float.*parseFloat*(Hours.getText());  
 Float.*parseFloat*(Price.getText());  
 **wmodel**.updateW(id+1, **new** Works(id,Title.getText(), Float.*parseFloat*(Hours.getText()), Float.*parseFloat*(Price.getText())));  
 }  
 **catch** (NumberFormatException ex){JOptionPane.*showMessageDialog*(**null**, **"Введите числа"**, **"Error"**, JOptionPane.***ERROR\_MESSAGE***);}  
 }  
 } **catch** (SQLException throwables) {  
 throwables.printStackTrace();  
 }  
 }  
 });  
 dialog.add(add);  
 dialog.setVisible(**true**);  
 }  
 **public void** AdditionOrder() **throws** SQLException {  
  
 JDialog dialog = **new** JDialog();  
 dialog.setModal(**true**);  
 dialog.setLocation(600,600);  
 dialog.setTitle(**"Добавление текущего заказа"**);  
 dialog.setSize(500,400);  
 GridLayout glA = **new** GridLayout(9,2,30,3 );  
 JPanel add=**new** JPanel(glA);  
 JLabel title=**new** JLabel(**" Услуга"**);  
 JLabel emp=**new** JLabel(**" Исполнитель"**);  
 JLabel comment=**new** JLabel(**" Комментарий"**);  
 JLabel car= **new** JLabel(**" Номер машины"**);  
 JTextField Comment = **new** JTextField(100);  
 JComboBox works = **new** JComboBox(DBWorker.*getAllworksTitles*());  
 JComboBox emps = **new** JComboBox(DBWorker.*getAllempsName*());  
 JComboBox cars = **new** JComboBox(DBWorker.*getAllcars*());  
 JButton Add = **new** JButton(**"Добавить"**);  
 add.add(title); add.add(works); add.add(emp); add.add(emps); add.add(car); add.add(cars); add.add(comment); add.add(Comment);  
 add.add(Add);  
 Add.addActionListener(**new** ActionListener() {  
 @Override  
 **public void** actionPerformed(ActionEvent e) {  
 **int** id=**tborders**.getRowCount()+1;  
  
 **try** { **omodel**.add(**new** Orders(id, (String)works.getSelectedItem(), (String)emps.getSelectedItem(), (String)cars.getSelectedItem(), Comment.getText()));  
 Comment.setText(**null**);  
 } **catch** (SQLException throwables) {  
 throwables.printStackTrace();  
 }  
 }  
 });  
 dialog.add(add);  
 dialog.setVisible(**true**);  
 }  
 **public void** UpdatingOrder() **throws** SQLException {  
  
 JDialog dialog = **new** JDialog();  
 dialog.setModal(**true**);  
 dialog.setLocation(600,600);  
 dialog.setTitle(**"Изменение текущего заказа"**);  
 dialog.setSize(500,400);  
 GridLayout glA = **new** GridLayout(9,2,30,3 );  
 JPanel add=**new** JPanel(glA);  
 JLabel title=**new** JLabel(**" Услуга"**);  
 JLabel emp=**new** JLabel(**" Исполнитель"**);  
 JLabel comment=**new** JLabel(**" Комментарий"**);  
 JLabel car= **new** JLabel(**" Номер машины"**);  
 JTextField Comment = **new** JTextField(100);  
 Orders orders = **omodel**.getOrder(**tborders**.getSelectedRow());  
 JComboBox works = **new** JComboBox(DBWorker.*getAllworksTitles*()); works.setSelectedItem(orders.getWork());  
 JComboBox emps = **new** JComboBox(DBWorker.*getAllempsName*()); emps.setSelectedItem(orders.getEmp());  
 JComboBox cars = **new** JComboBox(DBWorker.*getAllcars*()); cars.setSelectedItem(orders.getCar()); Comment.setText(orders.getComment());  
 JButton Add = **new** JButton(**"Изменить"**);  
 add.add(title); add.add(works); add.add(emp); add.add(emps); add.add(car); add.add(cars);add.add(comment); add.add(Comment);  
 add.add(Add);  
 Add.addActionListener(**new** ActionListener() {  
 @Override  
 **public void** actionPerformed(ActionEvent e) {  
 **int** id=**tborders**.getSelectedRow();  
 **try** { **omodel**.updateO(id+1, **new** Orders(id, (String)works.getSelectedItem(), (String)emps.getSelectedItem(), (String)cars.getSelectedItem(), Comment.getText()));  
 } **catch** (SQLException throwables) {  
 throwables.printStackTrace();  
 }  
 }  
 });  
 dialog.add(add);  
 dialog.setVisible(**true**);  
 }  
}  
}

# 7.3 Класс DBWorker

**import** com.company.basicClasses.Clients;  
**import** com.company.basicClasses.Employees;  
**import** com.company.basicClasses.Orders;  
**import** com.company.basicClasses.Works;  
  
**import** java.sql.\*;  
**import** java.util.ArrayList;  
**import** java.util.List;  
  
**public class** DBWorker {  
 **public static final** String ***PATH\_TO\_DB\_FILE*** = **"CarRepair.db"**;  
 **public static final** String ***URL*** = **"jdbc:sqlite:"** + ***PATH\_TO\_DB\_FILE***;  
 **public static** Connection *connection*;  
  
 **public static void** initDB() {  
 **try** {  
 *connection* = DriverManager.*getConnection*(***URL***);  
 **if** (*connection* != **null**) {  
 *//createDB();* }  
 } **catch** (SQLException ex) {  
 System.***out***.println(**"Ошибка подключения: "** + ex);  
 }  
 }  
  
 **public static void** closeConnection() **throws** SQLException {  
 **if** (*connection* != **null**) {  
 *connection*.close();  
 }  
 }  
  
 **public static void** createDB() **throws** SQLException {  
 Statement stat = *connection*.createStatement();  
 stat.execute(**"CREATE TABLE if not exists 'works' ('id' integer, 'title' TEXT PRIMARY KEY, 'hours' float, 'price' float);"**);  
 stat.execute(**"CREATE TABLE if not exists 'employees' ('id' integer, 'name' TEXT PRIMARY KEY, 'job' text);"**);  
 stat.execute(**"CREATE TABLE if not exists 'clients' ('id' integer, 'name' TEXT PRIMARY KEY, 'phone' text, 'car' text);"**);  
 stat.execute(**"CREATE TABLE if not exists 'CurrentOrders' ('id' INTEGER PRIMARY KEY, "** +  
 **"'work\_name' TEXT NOT NULL, 'emp\_name' TEXT NOT NULL, 'car' TEXT NOT NULL, 'comment' text, FOREIGN KEY (work\_name) REFERENCES works (title)ON DELETE CASCADE FOREIGN KEY (emp\_name) REFERENCES employees (name)ON DELETE CASCADE FOREIGN KEY (car) REFERENCES clients (car)ON DELETE CASCADE);"**);  
 System.***out***.println(**"Таблицы созданы"**);  
 }  
  
  
 **public static void** addClient(Clients client) **throws** SQLException {  
 PreparedStatement statement=*connection*.prepareStatement(**"INSERT INTO clients('id','name','phone', 'car') "** +  
 **"VALUES(?,?,?,?);"**);  
 statement.setObject(1, client.getId());  
 statement.setObject(2, client.getName());  
 statement.setObject(3, client.getPhone());  
 statement.setObject(4,client.getCar());  
 statement.executeUpdate();  
 statement.close();  
 }  
  
 **public static** List<Clients> getAllclients() **throws** SQLException {  
 Statement statement = *connection*.createStatement();  
 List<Clients> list = **new** ArrayList<>();  
 ResultSet resultSet = statement.executeQuery(**"SELECT \* FROM clients"**);  
 **while** (resultSet.next()) {  
 list.add(**new** Clients(resultSet.getInt(**"id"**),resultSet.getString(**"name"**), resultSet.getString(**"phone"**), resultSet.getString(**"car"**)));  
 }  
 resultSet.close();  
 statement.close();  
 **return** list;  
 }  
 **public static void** deleteClients(Clients clients) **throws** SQLException  
 { Statement statement= *connection*.createStatement();  
 statement.executeUpdate(**"DELETE FROM clients WHERE clients.id = "**+clients.getId());  
 statement.execute(**"UPDATE clients set id=id-1 WHERE id>"**+clients.getId());  
 System.***out***.println(**"deleted!"**);  
 statement.close();  
 }  
 **public static void** deleteAllclients() **throws** SQLException {  
 Statement statement = *connection*.createStatement();  
 statement.execute(**"DELETE FROM clients"**);  
 System.***out***.println(**"all deleted!"**);  
 statement.close();  
 }  
 **public static void** updateClient(**int** id, Clients clients) **throws** SQLException {  
 PreparedStatement statement = *connection*.prepareStatement(**"UPDATE clients SET name=?, phone=?, car=? WHERE id=?"**);  
 statement.setObject(1, clients.getName());  
 statement.setObject(2, clients.getPhone());  
 statement.setObject(3, clients.getCar());  
 statement.setObject(4, id);  
 statement.execute();  
 statement.close();  
 }  
  
 **public static void** addEmp(Employees employees) **throws** SQLException {  
 PreparedStatement statement=*connection*.prepareStatement(**"INSERT INTO employees('id','name','job') "** +  
 **"VALUES(?,?,?);"**);  
 statement.setObject(1, employees.getId());  
 statement.setObject(2, employees.getName());  
 statement.setObject(3, employees.getJob());  
 statement.execute();  
 statement.close();  
 }  
 **public static** List<Employees> getAllemps() **throws** SQLException {  
 Statement statement = *connection*.createStatement();  
 List<Employees> list = **new** ArrayList<>();  
 ResultSet resultSet = statement.executeQuery(**"SELECT \* FROM employees"**);  
 **while** (resultSet.next()) {  
 list.add(**new** Employees(resultSet.getInt(**"id"**),resultSet.getString(**"name"**), resultSet.getString(**"job"**)));  
 }  
 resultSet.close();  
 statement.close();  
 **return** list;  
 }  
 **public static void** deleteEmps(Employees employees) **throws** SQLException  
 { Statement statement= *connection*.createStatement();  
 statement.executeUpdate(**"DELETE FROM employees WHERE employees.id = "**+employees.getId());  
 statement.execute(**"UPDATE employees set id=id-1 WHERE id>"**+employees.getId());  
 System.***out***.println(**"deleted!"**);  
 statement.close();  
 }  
 **public static void** deleteAllemps() **throws** SQLException {  
 Statement statement = *connection*.createStatement();  
 statement.execute(**"DELETE FROM employees"**);  
 System.***out***.println(**"all deleted!"**);  
 statement.close();  
 }  
 **public static void** updateEmp(**int** id, Employees employees) **throws** SQLException {  
 PreparedStatement statement = *connection*.prepareStatement(**"UPDATE employees SET name=?, job=? WHERE id=?"**);  
 statement.setObject(1, employees.getName());  
 statement.setObject(2, employees.getJob());  
 statement.setObject(3, id);  
 statement.execute();  
 statement.close();  
 }  
  
 **public static void** addWork(Works works) **throws** SQLException {  
 PreparedStatement statement=*connection*.prepareStatement(**"INSERT INTO works('id','title','hours', 'price') "** +  
 **"VALUES(?,?,?,?);"**);  
 statement.setObject(1, works.getId());  
 statement.setObject(2, works.getTitle());  
 statement.setObject(3, works.getHours());  
 statement.setObject(4, works.getPrice());  
 statement.execute();  
 statement.close();  
 }  
 **public static** List<Works> getAllworks() **throws** SQLException {  
 Statement statement = *connection*.createStatement();  
 List<Works> list = **new** ArrayList<>();  
 ResultSet resultSet = statement.executeQuery(**"SELECT \* FROM works"**);  
 **while** (resultSet.next()) {  
 list.add(**new** Works(resultSet.getInt(**"id"**),resultSet.getString(**"title"**), resultSet.getFloat(**"hours"**), resultSet.getFloat(**"price"**)));  
 }  
 resultSet.close();  
 statement.close();  
 **return** list;  
 }  
 **public static void** deleteWorks(Works works) **throws** SQLException  
 { Statement statement= *connection*.createStatement();  
 statement.executeUpdate(**"DELETE FROM works WHERE works.id = "**+works.getId());  
 statement.execute(**"UPDATE works set id=id-1 WHERE id>"**+works.getId());  
 System.***out***.println(**"deleted!"**);  
 statement.close();  
 }  
 **public static void** deleteAllworks() **throws** SQLException {  
 Statement statement = *connection*.createStatement();  
 statement.execute(**"DELETE FROM works"**);  
 System.***out***.println(**"all deleted!"**);  
 statement.close();  
 }  
 **public static void** updateWork(**int** id, Works works) **throws** SQLException {  
 PreparedStatement statement = *connection*.prepareStatement(**"UPDATE works SET title=?, hours=?, price=? WHERE id=?"**);  
 statement.setObject(1, works.getTitle());  
 statement.setObject(2, works.getHours());  
 statement.setObject(3, works.getPrice());  
 statement.setObject(4, id);  
 statement.execute();  
 statement.close();  
 }  
  
 **public static** List<Orders> getAllorders() **throws** SQLException {  
 Statement statement = *connection*.createStatement();  
 List<Orders> list = **new** ArrayList<>();  
 ResultSet resultSet = statement.executeQuery(**"SELECT \* FROM CurrentOrders"**);  
 **while** (resultSet.next()) {  
 list.add(**new** Orders(resultSet.getInt(**"id"**),resultSet.getString(**"work\_name"**), resultSet.getString(**"emp\_name"**), resultSet.getString(**"car"**),resultSet.getString(**"comment"**)));  
 }  
 resultSet.close();  
 statement.close();  
 **return** list;  
 }  
 **public static void** addOrder(Orders orders) **throws** SQLException {  
 PreparedStatement statement=*connection*.prepareStatement(**"INSERT INTO CurrentOrders('id','work\_name','emp\_name', 'car','comment') "** +  
 **"VALUES(?,?,?,?,?);"**);  
 statement.setObject(1, orders.getId());  
 statement.setObject(2, orders.getWork());  
 statement.setObject(3, orders.getEmp());  
 statement.setObject(4, orders.getCar());  
 statement.setObject(5, orders.getComment());  
 statement.execute();  
 statement.close();  
 }  
  
 **public static void** deleteOrders(Orders orders) **throws** SQLException  
 { Statement statement= *connection*.createStatement();  
 statement.executeUpdate(**"DELETE FROM CurrentOrders WHERE CurrentOrders.id = "**+orders.getId());  
 statement.execute(**"UPDATE CurrentOrders set id=id-1 WHERE id>"**+orders.getId());  
 System.***out***.println(**"deleted!"**);  
 statement.close();  
 }  
 **public static void** deleteAllorders() **throws** SQLException {  
 Statement statement = *connection*.createStatement();  
 statement.execute(**"DELETE FROM CurrentOrders"**);  
 System.***out***.println(**"all deleted!"**);  
 statement.close();  
 }  
 **public static** String [] getAllempsName() **throws** SQLException {  
 *initDB*();  
 Statement statement = *connection*.createStatement();  
 List <String> Emp = **new** ArrayList<>();  
 ResultSet resultSet = statement.executeQuery(**"SELECT name FROM employees"**);  
 **while** (resultSet.next()) {  
 Emp.add(resultSet.getString(**"name"**));  
 }  
 String [] emp = **new** String[Emp.size()];  
 Emp.toArray(emp);  
 resultSet.close();  
 statement.close();  
 *closeConnection*();  
 **return** emp;  
 }  
 **public static** String [] getAllworksTitles() **throws** SQLException {  
 *initDB*();  
 Statement statement = *connection*.createStatement();  
 List <String> works = **new** ArrayList<>();  
 ResultSet resultSet = statement.executeQuery(**"SELECT title FROM works"**);  
 **while** (resultSet.next()) {  
 works.add(resultSet.getString(**"title"**));  
 }  
 String [] work = **new** String[works.size()];  
 works.toArray(work);  
 resultSet.close();  
 statement.close();  
 *closeConnection*();  
 **return** work;  
 }  
 **public static** String [] getAllcars() **throws** SQLException {  
 *initDB*();  
 Statement statement = *connection*.createStatement();  
 List <String> cars = **new** ArrayList<>();  
 ResultSet resultSet = statement.executeQuery(**"SELECT car FROM clients"**);  
 **while** (resultSet.next()) {  
 cars.add(resultSet.getString(**"car"**));  
 }  
 String [] car = **new** String[cars.size()];  
 cars.toArray(car);  
 resultSet.close();  
 statement.close();  
 *closeConnection*();  
 **return** car;  
 }  
 **public static void** updateOrder(**int** id, Orders orders) **throws** SQLException {  
 PreparedStatement statement = *connection*.prepareStatement(**"UPDATE CurrentOrders SET work\_name=?, emp\_name=?, car=?, comment=? WHERE id=?"**);  
 statement.setObject(1, orders.getWork());  
 statement.setObject(2, orders.getEmp());  
 statement.setObject(3, orders.getCar());  
 statement.setObject(4, orders.getComment());  
 statement.setObject(5, id);  
 statement.execute();  
 statement.close();  
 }  
}

# 7.4 Класс Orders

**public class** Orders {  
 **private int id**;  
 **private** String **work\_name** = **""**;  
 **private** String **emp\_name**=**""**;  
 **private** String **comment**;  
 **private** String **car**;  
 **public** Orders(**int** id, String work\_name, String emp\_name,String car, String comment){  
 **this**.**id**=id;  
 **this**.**work\_name**=work\_name;  
 **this**.**emp\_name**=emp\_name;  
 **this**.**comment**=comment;  
 **this**.**car**=car;  
 }  
  
 @Override  
 **public** String toString() {  
 **return "Orders{"** +  
 **"id="** + **id** +  
 **", work='"** + **work\_name** + **'\''** +  
 **", emp='"** + **emp\_name** + **'\''** +  
 **", comment='"** + **comment** + **'\''** +  
 **'}'**;  
 }  
  
 **public int** getId() {  
 **return id**;  
 }  
  
 **public** String getWork() {  
 **return work\_name**;  
 }  
  
 **public** String getEmp() {  
 **return emp\_name**;  
 }  
  
 **public** String getComment() {  
 **return comment**;  
 }  
  
 **public void** setId(**int** id) {  
 **this**.**id** = id;  
 }  
  
 **public void** setEmp(String emp) {  
 **this**.**emp\_name** = **emp\_name**;  
 }  
  
 **public void** setWork(String work) {  
 **this**.**work\_name** = **work\_name**;  
 }  
  
 **public void** setComment(String comment) {  
 **this**.**comment** = comment;  
 }  
  
 **public** String getCar() {  
 **return car**;  
 }  
}

# 7.5 Класс Clients

**public class** Clients {  
 **private int id**;  
 **private** String **name**;  
 **private** String **phone**;  
 **private** String **car**;  
 **public** Clients(**int** id, String name, String phone, String car){  
 **this**.**id**=id;  
 **this**.**name**=name;  
 **this**.**phone**=phone;  
 **this**.**car**=car;  
 }  
  
 **public** String getPhone() {  
 **return phone**;  
 }  
  
 **public void** setPhone(String phone) {  
 **this**.**phone** = phone;  
 }  
  
 **public int** getId() {  
 **return id**;  
 }  
  
 **public** String getName() {  
 **return name**;  
 }  
  
 **public void** setId(**int** id) {  
 **this**.**id** = id;  
 }  
  
 **public void** setName(String name) {  
 **this**.**name** = name;  
 }  
  
 **public** String getCar() {  
 **return car**;  
 }  
  
 @Override  
 **public** String toString() {  
 **return** getId()+**" "**+getName();  
 }  
}

# 7.6 Класс Employees

**public class** Employees {  
 **private int id**;  
 **private** String **name**;  
 **private** String **job**;  
 **public** Employees(**int** id, String name, String job){  
 **this**.**id**=id;  
 **this**.**name**=name;  
 **this**.**job**=job;  
 }  
  
 **public int** getId() {  
 **return id**;  
 }  
  
 **public** String getName() {  
 **return name**;  
 }  
  
 **public** String getJob() {  
 **return job**;  
 }  
  
 **public void** setId(**int** id) {  
 **this**.**id** = id;  
 }  
  
 **public void** setName(String name) {  
 **this**.**name** = name;  
 }  
  
 **public void** setJob(String job) {  
 **this**.**job** = job;  
 }  
  
 @Override  
 **public** String toString() {  
 **return** getId()+**" "**+getJob()+**" "**+getName();  
 }  
}

# 7.7 Класс Works

**public class** Works {  
 **private int id**;  
 **private** String **title**;  
 **private float hours**;  
 **private float price**;  
 **public** Works( **int** id, String title, **float** hours, **float** price){  
 **this**.**id**=id;  
 **this**.**title**=title;  
 **this**.**hours**=hours;  
 **this**.**price**=price;  
 }  
  
 **public int** getId() {  
 **return id**;  
 }  
  
 **public float** getHours() {  
 **return hours**;  
 }  
  
 **public float** getPrice() {  
 **return price**;  
 }  
  
 **public** String getTitle() {  
 **return title**;  
 }  
  
  
 @Override  
 **public** String toString() {  
 **return** getId()+**" "**+getTitle()+**" "**+getPrice()+**" "**+getHours();  
 }  
}

# 7.8 Класс OrdersTable

**import** com.company.DBWorker;  
**import** com.company.basicClasses.Orders;  
**import** javax.swing.table.AbstractTableModel;  
**import** java.sql.SQLException;  
**import** java.util.List;  
  
**public class** OrdersTable **extends** AbstractTableModel {  
 **private** List<Orders> **data**;  
 **public** OrdersTable () **throws** SQLException {  
 DBWorker.*initDB*();  
 **data**=DBWorker.*getAllorders*();  
 DBWorker.*closeConnection*();  
 }  
 **public void** update() **throws** SQLException {  
 DBWorker.*initDB*();  
 **data**=DBWorker.*getAllorders*();  
 DBWorker.*closeConnection*();  
 **this**.fireTableDataChanged();  
 }  
 @Override  
 **public int** getRowCount() {  
 **return data**.size();  
 }  
 **public** String getColumnName(**int** column) {  
 **switch** (column){  
 **case** 0: **return "id"**;  
 **case** 1: **return "Наименование услуги"**;  
 **case** 2: **return "Имя работника"**;  
 **case** 3: **return "Номер машины"**;  
 **case** 4: **return "Комментарий"**;  
 }  
 **return ""**;  
 }  
 @Override  
 **public int** getColumnCount() {  
 **return** 5;  
 }  
  
 @Override  
 **public** Object getValueAt(**int** rowIndex, **int** columnIndex) {  
 Orders orders=**data**.get(rowIndex);  
 **switch** (columnIndex){  
 **case** 0: **return** orders.getId();  
 **case** 1: **return** orders.getWork();  
 **case** 2: **return** orders.getEmp();  
 **case** 3: **return** orders.getCar();  
 **case** 4: **return** orders.getComment();  
 }  
 **return ""**;  
 }  
 **public** Orders getOrder(**int** row){  
 **return data**.get(row);  
 }  
  
 **public void** add(Orders orders) **throws** SQLException {  
 DBWorker.*initDB*();  
 DBWorker.*addOrder*(orders);  
 update();  
 DBWorker.*closeConnection*();  
 }  
 **public void** remove(**int** id) **throws** SQLException {  
 DBWorker.*initDB*();  
 DBWorker.*deleteOrders*(getOrder(id));  
 update();  
 DBWorker.*closeConnection*();  
 }  
 **public void** updateO(**int** id, Orders orders) **throws** SQLException {  
 DBWorker.*initDB*();  
 DBWorker.*updateOrder*(id, orders);  
 update();  
 DBWorker.*closeConnection*();  
 }  
 **public void** deleteAll() **throws** SQLException {  
 DBWorker.*initDB*();  
 DBWorker.*deleteAllorders*();  
 update();  
 DBWorker.*closeConnection*();  
 }  
}

# 7.9 Класс ClientTable

**import** com.company.DBWorker;  
**import** com.company.basicClasses.Clients;  
**import** javax.swing.table.AbstractTableModel;  
  
**import** java.sql.SQLException;  
**import** java.util.List;  
  
**public class** ClientTable **extends** AbstractTableModel {  
 **private** List <Clients> **data**;  
 **public** ClientTable () **throws** SQLException {  
 DBWorker.*initDB*();  
 **data**=DBWorker.*getAllclients*();  
 DBWorker.*closeConnection*();  
 }  
 **public void** update() **throws** SQLException {  
 DBWorker.*initDB*();  
 **data**=DBWorker.*getAllclients*();  
 DBWorker.*closeConnection*();  
 **this**.fireTableDataChanged();  
 }  
 @Override  
 **public int** getRowCount() {  
 **return data**.size();  
 }  
  
 @Override  
 **public int** getColumnCount() {  
 **return** 4;  
 }  
 **public** String getColumnName(**int** column) {  
 **switch** (column){  
 **case** 0: **return "id"**;  
 **case** 1: **return "Имя заказчика"**;  
 **case** 2: **return "Номер телефона"**;  
 **case** 3: **return "Номер машины"**;  
 }  
 **return ""**;  
 }  
 @Override  
 **public** Object getValueAt(**int** rowIndex, **int** columnIndex) {  
 Clients client=**data**.get(rowIndex);  
 **switch** (columnIndex){  
 **case** 0: **return** client.getId();  
 **case** 1: **return** client.getName();  
 **case** 2: **return** client.getPhone();  
 **case** 3: **return** client.getCar();  
 }  
 **return ""**;  
 }  
 **public** Clients getClient(**int** row){  
 **return data**.get(row);  
 }  
  
 **public void** add(Clients clients) **throws** SQLException {  
 DBWorker.*initDB*();  
 DBWorker.*addClient*(clients);  
 update();  
 DBWorker.*closeConnection*();  
 }  
 **public void** remove(**int** id) **throws** SQLException {  
 DBWorker.*initDB*();  
 DBWorker.*deleteClients*(getClient(id));  
 update();  
 DBWorker.*closeConnection*();  
 }  
 **public void** updateCl(**int** id, Clients clients) **throws** SQLException {  
 DBWorker.*initDB*();  
 DBWorker.*updateClient*(id, clients);  
 update();  
 DBWorker.*closeConnection*();  
 }  
 **public void** deleteAll() **throws** SQLException {  
 DBWorker.*initDB*();  
 DBWorker.*deleteAllclients*();  
 update();  
 DBWorker.*closeConnection*();  
 }  
}

# 7.10 Класс WorksTable

**import** com.company.DBWorker;  
**import** com.company.basicClasses.Works;  
  
**import** javax.swing.table.AbstractTableModel;  
**import** java.sql.SQLException;  
**import** java.util.List;  
  
**public class** WorksTable **extends** AbstractTableModel {  
 **private** List<Works> **data**;  
 **public** WorksTable () **throws** SQLException {  
 DBWorker.*initDB*();  
 **data**=DBWorker.*getAllworks*();  
 DBWorker.*closeConnection*();  
 }  
 **public void** update() **throws** SQLException {  
 DBWorker.*initDB*();  
 **data**=DBWorker.*getAllworks*();  
 DBWorker.*closeConnection*();  
 **this**.fireTableDataChanged();  
 }  
 @Override  
 **public int** getRowCount() {  
 **return data**.size();  
 }  
  
 @Override  
 **public int** getColumnCount() {  
 **return** 4;  
 }  
 **public** String getColumnName(**int** column) {  
 **switch** (column){  
 **case** 0: **return "id"**;  
 **case** 1: **return "Наименование услуги"**;  
 **case** 2: **return "Время выполнения"**;  
 **case** 3: **return "Цена"**;  
 }  
 **return ""**;  
 }  
 @Override  
 **public** Object getValueAt(**int** rowIndex, **int** columnIndex) {  
 Works works=**data**.get(rowIndex);  
 **switch** (columnIndex){  
 **case** 0: **return** works.getId();  
 **case** 1: **return** works.getTitle();  
 **case** 2: **return** works.getHours();  
 **case** 3: **return** works.getPrice();  
 }  
 **return ""**;  
 }  
 **public** Works getWork(**int** row){  
 **return data**.get(row);  
 }  
  
 **public void** add(Works works) **throws** SQLException {  
 DBWorker.*initDB*();  
 DBWorker.*addWork*(works);  
 update();  
 DBWorker.*closeConnection*();  
 }  
 **public void** remove(**int** id) **throws** SQLException {  
 DBWorker.*initDB*();  
 DBWorker.*deleteWorks*(getWork(id));  
 update();  
 DBWorker.*closeConnection*();  
 }  
 **public void** updateW(**int** id, Works works) **throws** SQLException {  
 DBWorker.*initDB*();  
 DBWorker.*updateWork*(id, works);  
 update();  
 DBWorker.*closeConnection*();  
 }  
 **public void** deleteAll() **throws** SQLException {  
 DBWorker.*initDB*();  
 DBWorker.*deleteAllworks*();  
 update();  
 DBWorker.*closeConnection*();  
 }  
}

# 7.11 Класс EmployeesTable

**import** com.company.DBWorker;  
**import** com.company.basicClasses.Employees;  
  
**import** javax.swing.table.AbstractTableModel;  
**import** java.sql.SQLException;  
**import** java.util.List;  
  
**public class** EmployeesTable **extends** AbstractTableModel {  
 **private** List<Employees> **data**;  
  
 **public** EmployeesTable() **throws** SQLException {  
 DBWorker.*initDB*();  
 **data**=DBWorker.*getAllemps*();  
 DBWorker.*closeConnection*();  
  
 }  
 **public void** update() **throws** SQLException {  
 DBWorker.*initDB*();  
 **data**=DBWorker.*getAllemps*();  
 DBWorker.*closeConnection*();  
 **this**.fireTableDataChanged();  
 }  
 **public int** getRowCount() {  
 **return data**.size();  
 }  
  
 @Override  
 **public int** getColumnCount() {  
 **return** 3;  
 }  
 **public** String getColumnName(**int** column) {  
 **switch** (column){  
 **case**(0):  
 **return "Номер работника"**;  
 **case**(1):  
 **return "Имя работника"**;  
 **case**(2):  
 **return "Должность"**;  
 }  
 **return ""**;  
 }  
 @Override  
 **public** Object getValueAt(**int** rowIndex, **int** columnIndex) {  
 Employees employees=**data**.get(rowIndex);  
 **switch** (columnIndex){  
 **case** 0: **return** employees.getId();  
 **case** 1: **return** employees.getName();  
 **case** 2: **return** employees.getJob();  
 }  
 **return ""**;  
 }  
 **public** Employees getEmp(**int** row){  
 **return data**.get(row);  
 }  
  
 **public void** add(Employees emp) **throws** SQLException {  
 DBWorker.*initDB*();  
 DBWorker.*addEmp*(emp);  
 update();  
 DBWorker.*closeConnection*();  
 }  
 **public void** remove(**int** id) **throws** SQLException {  
 DBWorker.*initDB*();  
 DBWorker.*deleteEmps*(getEmp(id));  
 update();  
 DBWorker.*closeConnection*();  
 }  
 **public void** updateEm(**int** id, Employees emp) **throws** SQLException {  
 DBWorker.*initDB*();  
 DBWorker.*updateEmp*(id, emp);  
 update();  
 DBWorker.*closeConnection*();  
 }  
 **public void** deleteAll() **throws** SQLException {  
 DBWorker.*initDB*();  
 DBWorker.*deleteAllemps*();  
 update();  
 DBWorker.*closeConnection*();  
 }  
}

# Список литературы

1. Java. Экспресс-курс [электронный ресурс] // Сайт Александра Климова [сайт], URL: http://developer.alexanderklimov.ru/android/java/java.php

2. API Specification for the Java 7 SE. [официальный сайт] URL: http://docs.oracle.com/javase/7/docs/api/

3. The Java Tutorials. SE [электронный ресурс], URL: http://docs.oracle.com/javase/tutorial/index.html

4. Гради Буч, Роберт А. Максимчук, Майкл У. Энгл, Бобби Дж. Янг, Джим Коналлен, Келли А. Хьюстон. Объектно-ориентированны анализ и проек-тирование с примерами приложений. Третье издание. М.: "Вильямс", 2010.

5.JavaRush [электронный ресурс], URL: <https://javarush.ru/groups/posts/massivy-java>

6. Объектно-ориентированное программирование. Метод. указания по

выполнению лабораторных работ / сост.: В.Л. Аршинский. – Иркутск : Изд-во

ИРНИТУ, 2017. – 24 c.