

Федеральное государственное бюджетное образовательное
учреждение высшего образования
«Национальный исследовательский университет «МЭИ»

Институт: ИнЭИ Кафедра: БИТ
Направление
подготовки/специальность: 09.03.03 Прикладная информатика

ЗАДАНИЕ
на выполнение КП/КР по дисциплине
«Объектно-ориентированный анализ и программирование»

Тема КП/КР: Разработка объектно-ориентированной программы на языке C++

Студент: Кагарманов Карим Эдуардович
(Фамилия, имя, отчество (при наличии) полностью)

Группа: ИЭ-61-23
(номер учебной группы)

Содержание задания:

1. Постановка задачи
2. Разработка иерархии классов
3. Программная реализация задачи
4. Разработка диаграмм
5. Оформление курсовой работы
6. Подготовка к защите курсовой работы

(вопросы, подлежащие изучению в соответствии с планируемыми результатами обучения, заполняются руководителем КП/КР)

Руководитель 14.12.2024 М.В. Раскатова
(дата) (Фамилия и инициалы)

Студент 14.12.2024 К.Э. Кагарманов
(дата) (Фамилия и инициалы)

ОГЛАВЛЕНИЕ

ОГЛАВЛЕНИЕ	2
ВВЕДЕНИЕ	3
ЗАДАНИЕ.....	4
Глава 1. Постановка задачи	6
1.1 Предметная область	6
Глава 2. Разработка программы	8
2.1 Диаграммы UML.....	8
2.2 Описание классов.....	9
2.2.1 Диаграмма классов	9
2.2.2 Используемые классы	10
2.3 Описание пользовательского интерфейса	19
Глава 3. Реализация и тестирование программы.....	26
3.1 Описание разработанной программы.....	26
3.2 Тестирование программы	26
ЗАКЛЮЧЕНИЕ	31
Приложения.....	32
Приложение 1	32
Приложение 2	39
Приложение 3	40

ВВЕДЕНИЕ

Целью настоящей работы является ознакомление на практике с принципами объектно-ориентированного программирования, такими как инкапсуляция и наследование. Для демонстрации принципов и особенностей объектно-ориентированного способа программирования планируется проектирование и разработка объектно-ориентированной программы на языке C++.

Программа представляет собой меню для администратора, который ведёт учёт сотрудников в системе, он может производить действия, как над отделом в целом, так и с позиции отдельного сотрудника в отделе. Это сделано в демонстрационных целях.

ЗАДАНИЕ

- Построить и реализовать иерархию, содержащую классы: Человек, Директор, Бухгалтер, Охранник, Электрик.

- Директоров может быть несколько и для каждого директора должны быть свои сотрудники.

- Бухгалтер или секретарь должен быть один.

- Класс Директор:

Должен содержать имя, фамилию, отчество и зарплату. Данные поля должны находиться в закрытой области класса.

Также класс должен содержать поле, содержащее ФИО сотрудников, которые находятся в подчинении у директора, и их заработную плату.

Реализовать методы для увольнения и принятия работников.

Реализовать методы, позволяющие читать/писать из/в полей класса.

- Класс Бухгалтер:

Должен содержать имя, фамилию, отчество и зарплату. Данные поля должны находиться в закрытой области класса.

Также класс должен содержать поле, содержащее ставку для всех должностей.

Реализовать метод для расчета заработной платы работникам, исходя из размера ставки (Полная ставка равна окладу).

Реализовать методы, позволяющие читать/писать из/в полей класса.

- Класс Секретарь:

Должен содержать имя, фамилию, отчество и зарплату. Данные поля должны находиться в закрытой области класса.

Реализовать метод, который для данного директора выводит в виде таблицы список сотрудников.

Если в вашем варианте есть класс Охранник, то реализовать метод, который принимает массив охранников и выводит их в таблицу.

Если в вашем варианте есть класс Электрик, то реализовать метод, который принимает массив электриков и выводит их в таблицу.

- Класс Водитель:

Должен содержать имя, фамилию, отчество и зарплату. Данные поля должны находиться в закрытой области класса.

Класс должен содержать поле, которое хранит в себе массив из категорий прав.

Также должен содержать поле, содержащее массив транспортных средств, которыми управляет водитель.

Реализовать методы, позволяющие читать/писать из/в полей класса.

- Класс Электрик:

Должен содержать имя, фамилию, отчество и зарплату. Данные поля должны находиться в закрытой области класса.

Класс должен содержать поле, которое хранит разряд электрика.

Класс должен содержать поле, которое будет содержать список оборудования, которыми владеет электрик.

Реализовать методы, позволяющие читать/писать из/в полей класса.

- Разработать свои методы и свойства для представленных классов, должны соответствовать предметной области – "энерготехсервис".

- Реализовать программу на языке программирования C++.

- Программа должна содержать обработку исключительных ситуаций.

- Подготовить набор тестовых данных.

- Провести тестирование программы на предмет границ вводимых значений и выброса исключений.

ГЛАВА 1. ПОСТАНОВКА ЗАДАЧИ

Необходимо реализовать программу, в которой будет использована приведённая выше иерархия классов. Для классов необходимо разработать методы и свойства, на основе которых в программе будет реализован пользовательский интерфейс взаимодействия программы на основе текстового меню в консоли.

Основой иерархии должен стать абстрактный базовый класс **Работник**. Его наследниками будут **Директор**, **Бухгалтер**, **Секретарь**, **Электрик** и **Водитель**.

Программа будет представлять собой средство для контроля информации о персонале организации.

1.1 Предметная область

Предметная область: энерготехсервис.

Название компании: "ЭнергоТехСервис".

Компания "ЭнергоТехСервис" занимается проектированием и установкой систем для управления энергоснабжением и модернизации инфраструктуры, касающейся электрики.

При обращении клиента в компанию он обслуживается секретарём, который сопровождает проект от первого контакта до завершения установки. На первом этапе оцениваются требования клиента, потребности и предлагают оптимальные решения для модернизации или установки оборудования, если это не аварийная заявка, где все действия выполняются экстренно. Если клиент решает заключить договор, то составляется договор и техническое задание (ТЗ).

После утверждения проекта системой с клиентом, проект передается в технический отдел. Электрики в этом отделе занимаются установкой оборудования и подключением его к электросетям. Выезд электрика на объект может происходить индивидуально или с задействованием водителя. Задание водителю, как и электрику, формируется секретарем. Водитель получает задание в случае, когда электрику при работе на объекте понадобится оборудование,

которое он не в состоянии перевезти индивидуально. При монтаже оборудования всегда формируется заявка и водителю. При ремонте это делается по необходимости. Также водитель может потребоваться в случае отсутствия личного транспорта у электрика. Если электрику на объекте необходим специальный транспорт, который будет участвовать в монтаже, то водителю также формируется задание от секретаря. На месте производится монтаж или ремонт электросетей, после чего электрик устно сообщает о выполнении очередного задания. Водитель также при завершении транспортировки оборудования или электрика на место отмечается секретарю о выполнении. Каждая выполненная заявка отмечается в системе. При необходимости секретарем формируются отчеты за день, неделю, месяц, квартал и т.д.

Директор может запрашивать отчёты у секретаря, который в свою очередь может обращаться за данными к бухгалтеру или системе.

Перед сдачей системы клиенту проводится тестирование всех компонентов системы для проверки их функциональности и эффективности. После тестирования клиенту предоставляется обучение по эксплуатации системы, объясняются все функции и возможности.

Компания предоставляет услуги технического обслуживания, включая замену оборудования по сроку службы.

Секретарь также имеет возможность просматривать список сотрудников, получать отчёты о деятельности электриков и водителей.

Бухгалтер занимается расчётом заработной платы сотрудников, исходя из ставки, где полная ставка соответствует окладу. В расчёт также включаются бонусы, назначаемые директорами для сотрудников по личной инициативе или ходатайству секретаря.

ГЛАВА 2. РАЗРАБОТКА ПРОГРАММЫ

2.1 Диаграммы UML



Рис. 2.1. Диаграмма вариантов использования

На рисунке 2.1 представлена диаграмма вариантов использования. На ней описано, какие действия могут быть совершены сотрудниками в компании. Каждый овал на диаграмме представляет собой отдельный вариант использования, а линии, соединяющие пользователей (актёров) с функциями, показывают, какие действия доступны каждому из них.

Диаграмма помогает понять, какие функции относятся к определённым актёрам.

Различные диаграммы UML для вариантов использования приведены в приложении 1.

2.2 Описание классов

2.2.1 Диаграмма классов

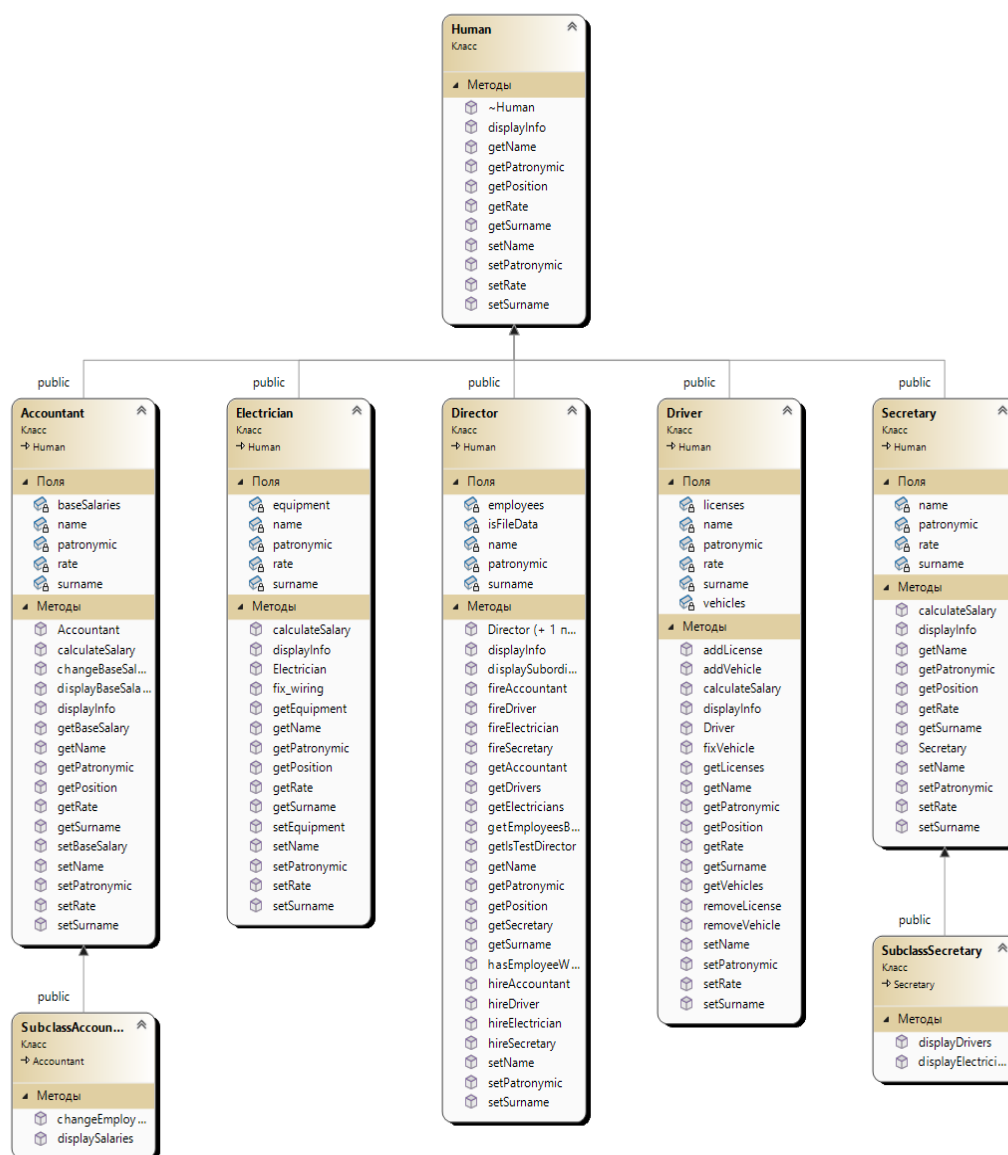


Рис. 2.2. Диаграмма классов

Диаграмма классов представлена на рисунке 2.2.

Перевод: Human – Человек, Accountant – Бухгалтер, Director – Директор, Secretary – Секретарь, Electrician – Электрик, Driver – Водитель. Заметим, что были добавлены классы "SubclassAccountant" и "SubclassSecretary", которые расширяют классы Бухгалтера и Секретаря соответственно.

2.2.2 Используемые классы

Дальше будет представлено описание разработанной программы, код программы находится в приложении 3.

```
class Human
```

Методы:

```
public:
```

- `virtual ~Human ()` – конструктор класса Human;
- `virtual string getName () const` – функция, которая возвращает имя сотрудника;
- `virtual string getSurname () const` – функция, которая возвращает фамилию сотрудника;
- `virtual string getPatronymic () const` – функция, которая возвращает отчество сотрудника;
- `virtual double getRate () const` – функция, которая возвращает ставку сотрудника;
- `virtual string getPosition () const` – функция, которая возвращает должность сотрудника;
- `virtual void setName (const string&)` – функция, которая записывает имя сотрудника;
- `virtual void setSurname (const string&)` – функция, которая записывает фамилию сотрудника;
- `virtual void setPatronymic (const string&)` – функция, которая записывает отчество сотрудника;
- `virtual void setRate (double)` – функция, которая записывает ставку сотрудника;
- `virtual void displayInfo () const = 0` – чисто виртуальная функция, которая подразумевает отображение исчерпывающих данных о сотруднике в наследуемых классах.

```
class Electrician : public Human
```

Свойства:

private:

- string name – поле, которое хранит значение имени электрика;
- string surname – поле, которое хранит значение фамилии электрика;
- string patronymic – поле, которое хранит значение отчества электрика;
- double rate – поле, которое хранит значение ставки электрика;
- string equipment – поле, которое хранит значение оборудования электрика;

Методы:

public:

- Electrician (string n, string s, string p, double r, const string& eq) : name (n), surname (s), patronymic (p), rate (r), equipment (eq) – конструктор класса Electrician;
- string getName () const override – функция, которая возвращает имя электрика;
- string getSurname () const override – функция, которая возвращает фамилию электрика;
- string getPatronymic () const override – функция, которая возвращает отчество электрика;
- double getRate () const override – функция, которая возвращает ставку электрика;
- string getPosition () const override – функция, которая возвращает позицию электрика;
- void setName (const string& n) override – функция, которая записывает имя электрика;
- void setSurname (const string& s) override – функция, которая записывает фамилию электрика;

- `void setPatronymic (const string& p) override` – функция, которая записывает отчество электрика;
- `void setRate (double r) override` – функция, которая записывает ставку электрика;
- `void setEquipment (const string& eq)` – функция, которая записывает оборудование электрика;
- `string getEquipment () const` – функция, которая возвращает оборудование электрика;
- `bool fix_wiring () const` – уникальная функция электрика, который чинит проводку (электрик может быть убит током, после чего уволен);
- `void displayInfo () const override` – функция вывода данных об электрике;
- `double calculateSalary (double baseSalary) const` – функция расчёта зарплаты электрика в соответствии со ставкой.

`class Driver : public Human`

Свойства:

`private:`

- `string name` – поле, которое хранит значение имени водителя;
- `string surname` – поле, которое хранит значение фамилии водителя;
- `string patronymic` – поле, которое хранит значение отчества водителя;
- `double rate` – поле, которое хранит значение ставки водителя;
- `vector<string> licenses` – вектор, в котором хранятся водительские удостоверения водителя;
- `vector<string> vehicles` – вектор, в котором хранятся транспортные средства водителя;

Методы:

`public:`

- `Driver (string n, string s, string p, double r) : name (n), surname (s), patronymic (p), rate (r) bool ch_surname () override` – конструктор класса `Driver`;

- `string getName () const override` – функция, которая возвращает имя водителя;
- `string getSurname () const override` – функция, которая возвращает фамилию водителя;
- `string getPatronymic () const override` – функция, которая возвращает отчество водителя;
- `double getRate () const override` – функция, которая возвращает ставку водителя;
- `string getPosition () const override` – функция, которая возвращает позицию водителя;
- `void setName (const string& n) override` – функция, которая записывает имя водителя;
- `void setSurname (const string& s) override` – функция, которая записывает фамилию водителя;
- `void setPatronymic (const string& p) override` – функция, которая записывает отчество водителя;
- `void setRate (double r) override` – функция, которая записывает ставку водителя;
- `void addLicense (const string& license)` – функция добавления водительского удостоверения;
- `void removeLicense (int index)` – функция удаления водительского удостоверения;
- `void addVehicle (const string& vehicle)` – функция добавления транспортного средства;
- `void removeVehicle (int index)` – функция удаления транспортного средства;
- `void fixVehicle ()` – функция починки транспортного средства;
- `void displayInfo () const override` – функция вывода информации о водителе;

- `double calculateSalary (double baseSalary) const` – функция расчёта зарплаты водителя в соответствии со ставкой.

`class Accountant : public Human`

Свойства:

`private:`

- `string name` – поле, которое хранит значение имени водителя;
- `string surname` – поле, которое хранит значение фамилии водителя;
- `string patronymic` – поле, которое хранит значение отчества водителя;
- `double rate` – поле, которое хранит значение ставки водителя;
- `map<string, double> baseSalaries` – контейнер, который хранит базовые зарплаты для разных должностей;

Методы:

- `public:`
- `Accountant (string n, string s, string p, double r) : name (n), surname (s), patronymic (p), rate (r)` – конструктор класса `Accountant`;
- `string getName() const override` – функция, которая возвращает имя бухгалтера;
- `string getSurname () const override` – функция, которая возвращает фамилию бухгалтера;
- `string getPatronymic () const override` – функция, которая возвращает отчество бухгалтера;
- `double getRate () const override` – функция, которая возвращает ставку бухгалтера;
- `string getPosition () const override` – функция, которая возвращает позицию бухгалтера;
- `void setName (const string& n) override` – функция, которая записывает имя бухгалтера;
- `void setSurname (const string& s) override` – функция, которая записывает фамилию бухгалтера;

- `void setPatronymic (const string& p) override` – функция, которая записывает отчество бухгалтера;
- `void setRate (double r) override` – функция, которая записывает ставку бухгалтера;
- `void setBaseSalary (const string& position, double salary)` – функция, которая записывает базовую зарплату для должности;
- `void setBaseSalary (const string& position, double salary)` – функция, которая выводит базовую зарплату для должности;
- `void displayInfo () const override` – функция вывода информации о бухгалтере;
- `void displayBaseSalaries () const` – функция вывода окладов для каждой должности;
- `void changeBaseSalary ()` – функция для смены оклада для определённой должности;
- `double calculateSalary (double baseSalary) const` – функция расчёта зарплаты бухгалтера в соответствии со ставкой;

`class Secretary : public Human`

Свойства:

`private:`

- `string name` – поле, которое хранит значение имени секретаря;
- `string surname` – поле, которое хранит значение фамилии секретаря;
- `string patronymic` – поле, которое хранит значение отчества секретаря;
- `double rate` – поле, которое хранит значение ставки секретаря;

Методы:

`public:`

- `Secretary (string n, string s, string p, double r) : name (n), surname (s), patronymic (p), rate (r)` – конструктор класса `Secretary`;
- `string getName () const override` – функция, которая возвращает имя секретаря;

- `string getSurname () const override` – функция, которая возвращает фамилию секретаря;
- `string getPatronymic () const override` – функция, которая возвращает отчество секретаря;
- `double getRate () const override` – функция, которая возвращает ставку секретаря;
- `string getPosition () const override` – функция, которая возвращает позицию секретаря;
- `void setName (const string& n) override` – функция, которая записывает имя секретаря;
- `void setSurname (const string& s) override` – функция, которая записывает фамилию секретаря;
- `void setPatronymic (const string& p) override` – функция, которая записывает отчество секретаря;
- `void setRate (double r) override` – функция, которая записывает ставку секретаря;
- `void displayInfo () const override` – функция вывода информации о секретаре;
- `double calculateSalary (double baseSalary) const` – функция расчёта зарплаты секретаря в соответствии со ставкой.

```
class Director : public Human
```

Свойства:

```
private:
```

- `string name` – поле, которое хранит значение имени директора;
- `string surname` – поле, которое хранит значение фамилии директора;
- `string patronymic` – поле, которое хранит значение отчества директора;
- `vector<Human*> employees` – контейнер для хранения всех сотрудников;
- `bool isFileData` – атрибут для тестовых данных;

Методы:

public:

- Director (string n, string s, string p) : name (n), surname (s), patronymic (p)
Director (string n, string s, string p, bool testFlag) : name (n), surname (s), patronymic (p), isFileData (testFlag) – конструктор класса Director;
- string getName () const override – функция, которая возвращает имя директора;
- string getSurname () const override – функция, которая возвращает фамилию директора;
- string getPatronymic () const override – функция, которая возвращает отчество директора;
- string getPosition () const override – функция, которая возвращает позицию директора;
- bool getIsTestDirector () const – функция для проверки тестовых данных;
- void setName (const string& n) override – функция, которая записывает имя директора;
- void setSurname (const string& s) override – функция, которая записывает фамилию директора;
- void setPatronymic (const string& p) override – функция, которая записывает отчество директора;
- bool hasEmployeeWithPosition (const string& position) const – функция, которая проверяет, есть ли сотрудник с данной должностью;
- vector<Human*> getEmployeesByPosition(const string& position) const – функция для получения списка сотрудников по должности;
- Accountant* getAccountant () const – функция для получения бухгалтера;
- Secretary* getSecretary () const – функция для получения секретаря;
- vector<Human*> getElectricians () const – функция для получения всех электриков;

- void hireElectrician (Electrician* electrician) – функция для найма электрика;
- void hireDriver (Driver* driver) – функция для найма водителя;
- void hireSecretary (Secretary* secretary) – функция для найма секретаря с проверкой на существование;
- void hireAccountant (Accountant* accountant) – функция для найма бухгалтера с проверкой на существование;
- void fireElectrician (const string& electricianName) – функция для увольнения электрика по имени;
- void fireDriver (const string& driverName) – функция для увольнения водителя по имени;
- void fireSecretary () – функция для увольнения секретаря;
- void fireAccountant () – функция для увольнения бухгалтера;
- void displaySubordinates () const – функция для вывода информации о подчинённых.

class SubclassAccountant : public Accountant

Методы:

public:

- void displaySalaries (Director& director) const – функция, которая выводит зарплаты сотрудников;
- void changeEmployeeRate (Director& director) – функция, которая позволяет сменить ставку сотрудника.

class SubclassSecretary : public Secretary

Методы:

public:

- void displayElectricians (const Director& director) const – функция, которая выводит список электриков;
- void displayDrivers (const Director& director) const – функция, которая выводит список водителей.

2.3 Описание пользовательского интерфейса

Программа представляет собой консольный интерфейс для управления сотрудниками компании. Пользователь может управлять директорами и их сотрудниками.

Приложение консольное, следовательно, взаимодействие с ним осуществляется посредством выборов чисел из меню (главного или же ведомых).

Начало работы:

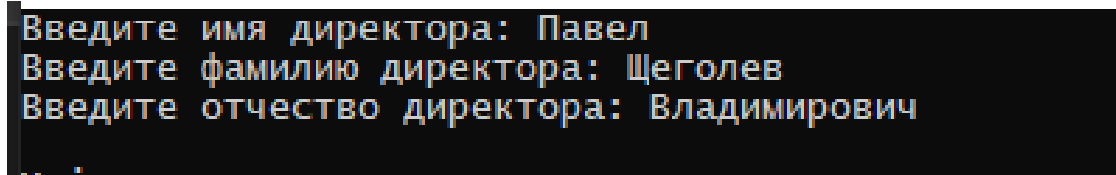
- В самом начале своей работы программа считывает данные из заранее подготовленного файла, после чего пользователь имеет возможность приступить к ручному добавлению сотрудников компании (рис. 2.3). При выборе пункта с ручным добавлением директора отдела компании пользователю будет необходимо ввести информацию о новом директоре (рис. 2.4).

- После чтения данных из файла или последующего ручного добавления отдела в меню можно выбрать пункт, который предоставляет пользователю возможность выбрать директора отдела компании для совершения дальнейших действий над ним (рис. 2.5).

- После выбора отдела появится меню с должностями работников отдела, от имени которых мы можем совершать действия. Для дальнейшей работы необходимо выбрать один из пунктов в меню.

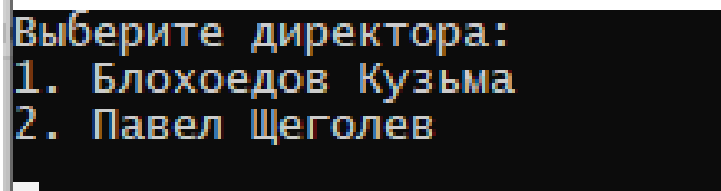
```
Бухгалтер Нестеров нанят.  
Секретарь Сидорова нанят.  
Водитель Антипов нанят.  
Электрик Ивахненко нанят.  
Данные по умолчанию загружены из файла example.txt.  
  
Main меню:  
1. Добавить нового директора  
2. Выбрать директора для управления компанией  
3. Уволить директора  
4. Выход  
Выберите действие: _
```

Рис. 2.3. Стартовое меню программы



```
Введите имя директора: Павел
Введите фамилию директора: Щеголев
Введите отчество директора: Владимирович
```

Рис. 2.4. Добавление нового директора



```
Выберите директора:
1. Блохоедов Кузьма
2. Павел Щеголев
```

Рис. 2.5. Выбор директора для управления

Меню для должностей:

Директор (рис. 2.6):

- Показать информацию о директоре – выводит всю информацию о директоре.
- Показать подчиненных – выводит список всех сотрудников в отделе с их ставками, оборудованием и тс (при наличии последних двух).
- Нанять сотрудника – выводит список всех сотрудников в отделе с их номерами, после этого пользователю предоставляется выбор должности, на которую будет нанят новый сотрудник. Когда пользователь выберет должность, приложение запросит ввод информации о новом сотруднике, пример для должности электрик представлен на рисунке 2.7.
- Уволить сотрудника – выводит список всех сотрудников в отделе с их номерами, после этого пользователю предоставляется выбор должности, с которой будет уволен сотрудник. Когда пользователь выберет должность, приложение запросит ввод порядкового номера сотрудника для увольнения, пример для должности электрик представлен на рисунке 2.8.
- Действия для водителя, действия для электрика, действия для секретаря, действия для бухгалтера – пункты меню, при выборе которых программа просит выбрать конкретного сотрудника из должности, после чего выводятся действия, доступные для манипуляций над конкретным сотрудником.

- Вернуться в main меню – возврат в главное меню.

```

Возврат в меню директора.

Меню директора Блохоедов:
1. Показать информацию о директоре
2. Показать подчинённых
3. Нанять сотрудника
4. Уволить сотрудника
5. Действия для водителя
6. Действия для электрика
7. Действия для секретаря
8. Действия для бухгалтера
9. Вернуться в main меню
Выберите действие: _

```

Рис. 2.6 Меню для директора

```

Кого вы хотите нанять?
1. Электрика
2. Водителя
3. Бухгалтера
4. Секретаря
Выберите должность: 1
Введите имя электрика: Каменщик
Введите фамилию электрика: Каменщиков
Введите отчество электрика: Каменчикович
Введите ставку электрика (от 0 до 1): 0.7
Введите оборудование электрика: отвертка
Ошибка: слово должно начинаться с заглавной буквы, а остальные буквы должны быть строчными. Попробуйте снова: Отвертка
Электрик Каменщик нанят.

```

Рис. 2.7 Меню для директора – пример ввода информации об электрике при найме на работу

```

Кого вы хотите уволить?
1. Электрика
2. Водителя
3. Бухгалтера
4. Секретаря
Выберите должность: 1

```

Имя	Фамилия	Зарплата
Ивахненко	Рустам	40200
Каменщик	Каменщиков	46900

```

Выберите электрика для увольнения (номер): 1
Электрик Ивахненко уволен.
Электрик уволен.

```

Рис. 2.8 Меню для директора – пример увольнения электрика

Секретарь (рис. 2.9):

- Показать всех электриков – выводит всех электриков, работающих в отделе, в таблицу, пример представлен в приложении 2.
- Показать всех охранников – выводит всех охранников, работающих в отделе, в таблицу, пример представлен в приложении 2.

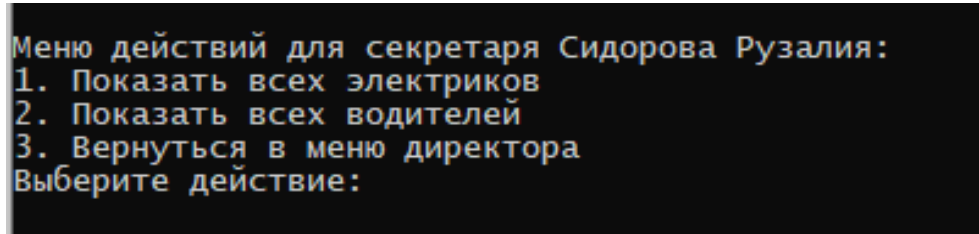


Рис. 2.9 Меню для секретаря

Бухгалтер (рис. 2.10):

- Рассчитать зарплату сотрудников – выводит список сотрудников с рассчитанными зарплатами в соответствии со ставками сотрудников и окладами по должностям.
- Показать оклады по должностям – показывает оклады по каждой должности (рис. 2.11).
- Поменять оклады у должности – предоставляется выбор должности для изменения оклада у одной (рис. 2.12).
- Установить ставку для сотрудника – выводится список всех сотрудников, а также ставок, после чего программа просит ввести номер сотрудника для изменения ставки (рис. 2.13).
- Вернуться в меню директора – позволяет вернуться в меню с действиями для директора.

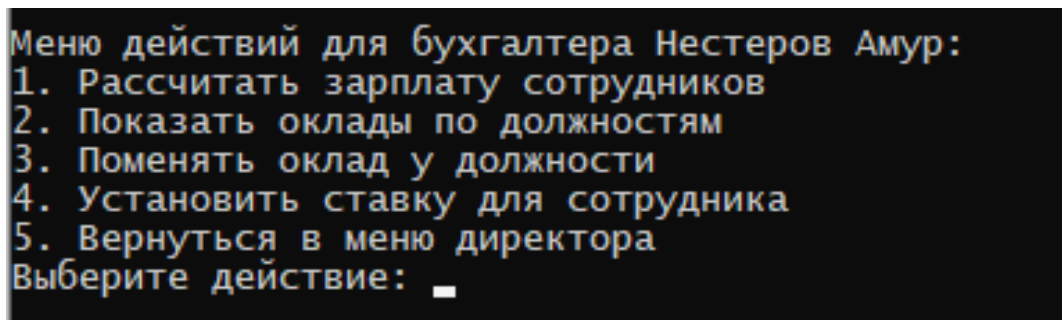


Рис. 2.10 Меню бухгалтера

```
Оклад для каждой должности:
Бухгалтер: 93000
Водитель: 71000
Директор: 111111
Секретарь: 87000
Электрик: 67000
```

Рис. 2.11 Меню бухгалтера – Оклады

```
Выберите должность для изменения базовой зарплаты:
1. Электрик
2. Водитель
3. Секретарь
4. Бухгалтер
5. Директор
1
Введите новую базовую зарплату для должности Электрик: 0.7
Зарплата должна быть выше прожиточного минимума.
```

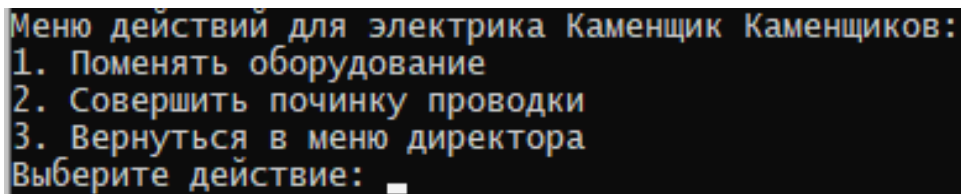
Рис. 2.12 Меню бухгалтера – Изменение оклада

```
Выберите сотрудника для изменения ставки:
1. Нестеров Амур - Бухгалтер, Ставка: 0.95
2. Сидорова Рузалия - Секретарь, Ставка: 0.6
3. Каменщик Каменщиков - Электрик, Ставка: 0.7
4. Антипов Алексей - Водитель, Ставка: 0.8
Введите номер сотрудника для изменения ставки: 3
Введите новую ставку (от 0 до 1): 0.9
Ставка электрика изменена на 0.9
```

Рис. 2.13 Меню бухгалтера – Ставки

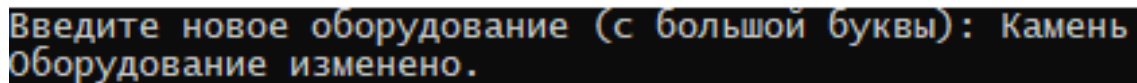
Электрик (рис. 2.14):

- Поменять оборудование – программа запрашивает ввод нового спец. инструмента электрика. Пользователь вводит название нового инструмента (рис. 2.15), после чего спец. инструмент охранника изменится.
- Совершить починку проводки – программа выводит степень успешности выполнения починки проводки (рис. 2.16).
- Вернуться в меню директора – позволяет вернуться в меню с действиями для директора.



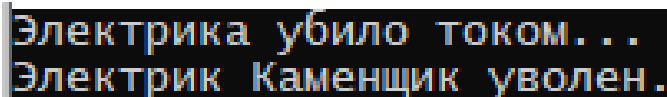
Меню действий для электрика Каменщик Каменщиков:
1. Поменять оборудование
2. Совершить починку проводки
3. Вернуться в меню директора
Выберите действие: █

Рис. 2.14 Меню электрика



Введите новое оборудование (с большой буквы): Камень
Оборудование изменено.

Рис. 2.15 Меню электрика - Сменить оборудование

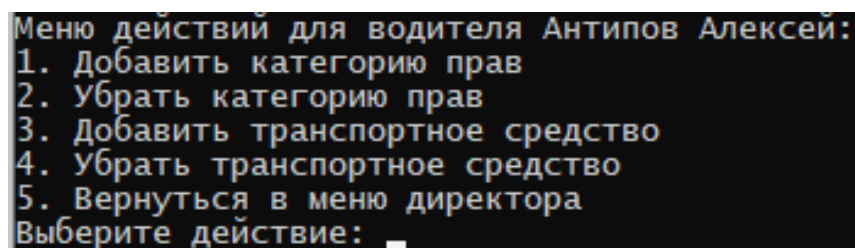


Электрика убило током...
Электрик Каменщик уволен.

Рис. 2.16 Меню электрика - Выполнить починку проводки

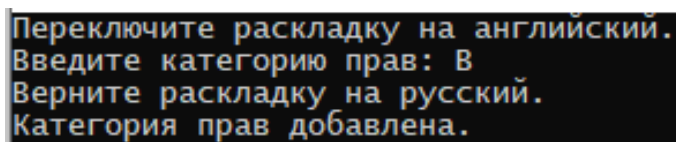
Водитель (рис. 2.17):

- Добавить категорию прав – программа просит ввести категорию прав для добавления (рис. 2.18).
- Убрать категорию прав – программа просит выбрать категорию прав для удаления (рис. 2.19).
- Добавить транспортное средство – программа просит ввести название транспортного средства для добавления, после чего подтверждает успешное добавление (рис. 2.20).
- Убрать транспортное средство – программа просит выбрать транспортное средство для удаления, после чего подтверждает успешное удаление (рис. 2.21).
- Вернуться в меню директора – позволяет вернуться в меню с действиями для директора.



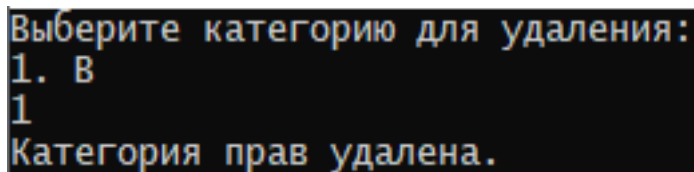
Меню действий для водителя Антипов Алексей:
1. Добавить категорию прав
2. Убрать категорию прав
3. Добавить транспортное средство
4. Убрать транспортное средство
5. Вернуться в меню директора
Выберите действие: █

Рис. 2.17 Меню водителя



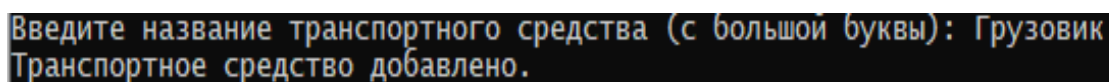
Переключите раскладку на английский.
Введите категорию прав: В
Верните раскладку на русский.
Категория прав добавлена.

Рис. 2.18 Меню водителя – Добавить категорию прав



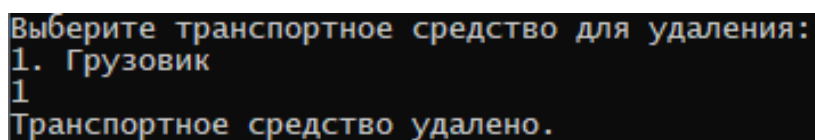
Выберите категорию для удаления:
1. В
1
Категория прав удалена.

Рис. 2.19 Меню электрика – Убрать категорию прав



Введите название транспортного средства (с большой буквы): Грузовик
Транспортное средство добавлено.

Рис. 2.20 Меню электрика – Добавить ТС



Выберите транспортное средство для удаления:
1. Грузовик
1
Транспортное средство удалено.

Рис. 2.21 Меню электрика – Убрать ТС

ГЛАВА 3. РЕАЛИЗАЦИЯ И ТЕСТИРОВАНИЕ ПРОГРАММЫ

3.1 Описание разработанной программы

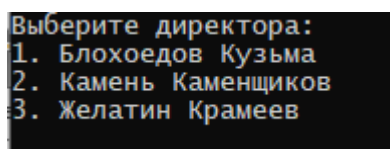
Код программы приведён в приложении 2.

В консоли появляется меню, с которым пользователь может начинать работу.

3.2 Тестирование программы

При запуске программы появляется стартовое меню (рис. 2.3), где программа уже прочитала тестовые данные из файла. Программа оповещает об успешном чтении из файла и в меню предоставляется выбор пунктов.

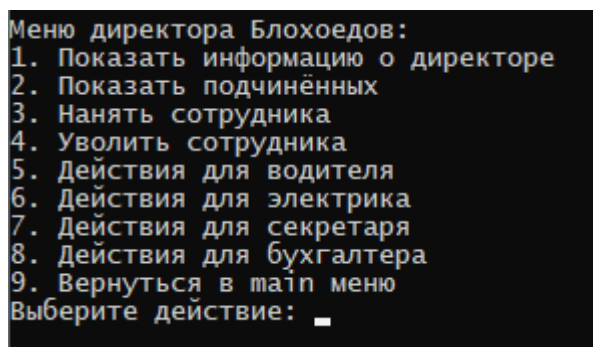
После выбора второго пункта и выбора директора для управления перед пользователем появляется меню выбора действий для отдела выбранного директора.



```
Выберите директора:
1. Блохоедов Кузьма
2. Камень Каменщиков
3. Желатин Крамеев
```

Рис. 3.3. Меню выбора директора для управления его отделом

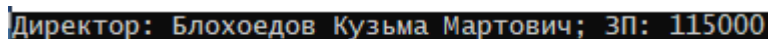
Для дальнейшего тестирования выберем отдел директора "Блохоедов". Перед пользователем предстаёт меню (рис. 3.4).



```
Меню директора Блохоедов:
1. Показать информацию о директоре
2. Показать подчинённых
3. Нанять сотрудника
4. Уволить сотрудника
5. Действия для водителя
6. Действия для электрика
7. Действия для секретаря
8. Действия для бухгалтера
9. Вернуться в main меню
Выберите действие: _
```

Рис. 3.4. Меню отдела директора "Блохоедов"

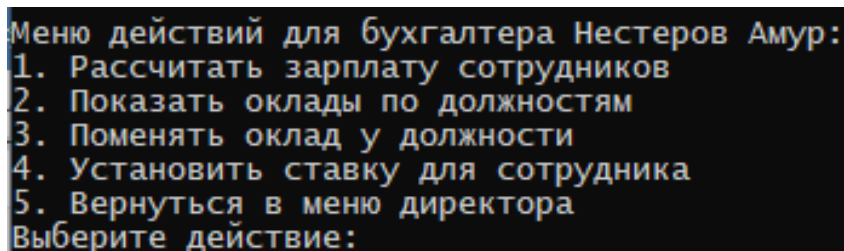
Предположим, что пользователь захотел изменить оклад директора на 100000 руб. Зайдём в пункт под цифрой 1 (рис. 3.5). Можем увидеть, что зарплата Директора 115000 руб.



```
Директор: Блохоедов Кузьма Мартович; ЗП: 115000
```

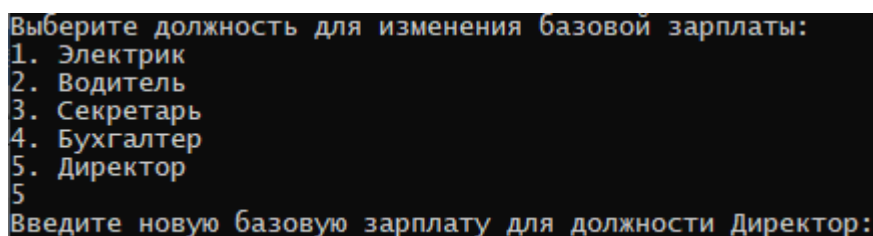
Рис. 3.5. Информация о директоре до изменения оклада

Далее зайдём в меню бухгалтера (рис. 3.6) и перейдём в пункт под цифрой 3. Выберем должность "Директор" нажатием клавиши "5" и введём новый оклад 150000 рублей (рис. 3.7, 3.8).



```
Меню действий для бухгалтера Нестеров Амур:  
1. Рассчитать зарплату сотрудников  
2. Показать оклады по должностям  
3. Поменять оклад у должности  
4. Установить ставку для сотрудника  
5. Вернуться в меню директора  
Выберите действие:
```

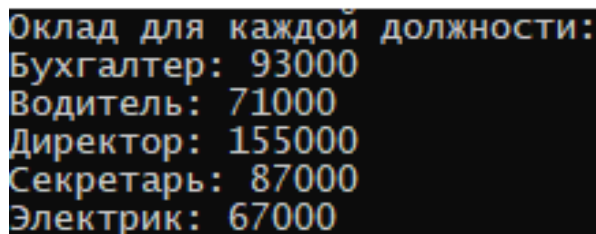
Рис. 3.6. Меню бухгалтера



```
Выберите должность для изменения базовой зарплаты:  
1. Электрик  
2. Водитель  
3. Секретарь  
4. Бухгалтер  
5. Директор  
5  
Введите новую базовую зарплату для должности Директор:
```

Рис. 3.7. Меню бухгалтера – Изменение оклада директора

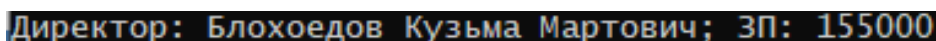
Оклад директора стал 155000 рублей (рис. 3.8).



```
Оклад для каждой должности:  
Бухгалтер: 93000  
Водитель: 71000  
Директор: 155000  
Секретарь: 87000  
Электрик: 67000
```

Рис. 3.8. Меню бухгалтера – Оклад, после подтверждения изменений

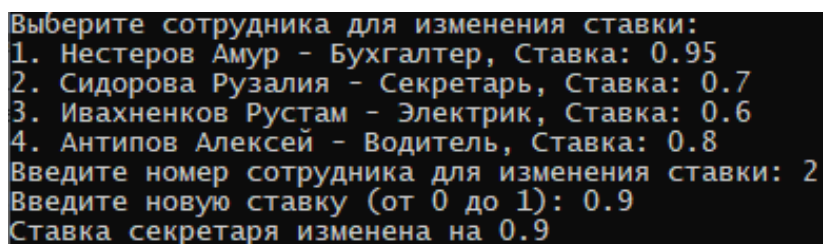
Зайдём в меню директора, его зарплата стала равна 155000 рублей.



```
Директор: Блохоедов Кузьма Мартович; ЗП: 155000
```

Рис. 3.9. Информация о директоре после изменения оклада

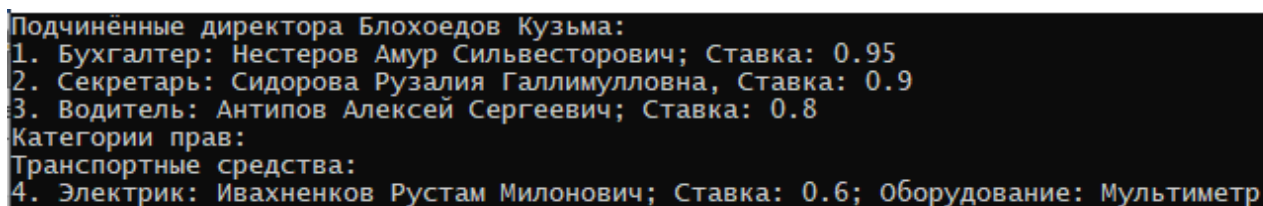
Теперь попробуем изменить ставку сотрудника, для этого снова зайдём в меню бухгалтера и выберем пункт 4 (рис. 3.10).



```
Выберите сотрудника для изменения ставки:
1. Нестеров Амур - Бухгалтер, Ставка: 0.95
2. Сидорова Рузалия - Секретарь, Ставка: 0.7
3. Ивахненко Рустам - Электрик, Ставка: 0.6
4. Антипов Алексей - Водитель, Ставка: 0.8
Введите номер сотрудника для изменения ставки: 2
Введите новую ставку (от 0 до 1): 0.9
Ставка секретаря изменена на 0.9
```

Рис. 3.10. Меню бухгалтера – Изменение ставки сотрудника

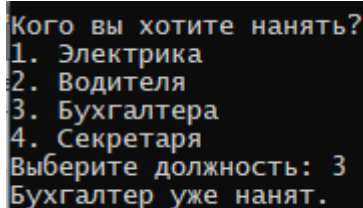
Зайдём в меню данных всех сотрудников и проверим зарплату Сидоровой Рузалии. Видим, что ставка стала выше (рис. 3.11).



```
Подчинённые директора Блохоедов Кузьма:
1. Бухгалтер: Нестеров Амур Сильвесторович; Ставка: 0.95
2. Секретарь: Сидорова Рузалия Галлимулловна, Ставка: 0.9
3. Водитель: Антипов Алексей Сергеевич; Ставка: 0.8
Категории прав:
Транспортные средства:
4. Электрик: Ивахненко Рустам Милонович; Ставка: 0.6; Оборудование: Мультиметр
```

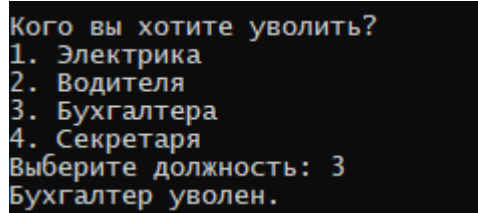
Рис. 3.11. Информация о сотрудниках после изменения ставки

Попробуем добавить нового сотрудника в отдел. Для этого выберем пункт под цифрой 3, перед пользователем предстаёт меню найма сотрудников (рис. 3.12). Предположим, мы хотим нанять нового бухгалтера, как мы видим сейчас это сделать невозможно, так как пост бухгалтера уже занят. Выйдем из меню найма и зайдём в меню увольнения сотрудников, выберем пункт 3, уволить бухгалтера (рис. 3. 13).



```
Кого вы хотите нанять?
1. Электрика
2. Водителя
3. Бухгалтера
4. Секретаря
Выберите должность: 3
Бухгалтер уже нанят.
```

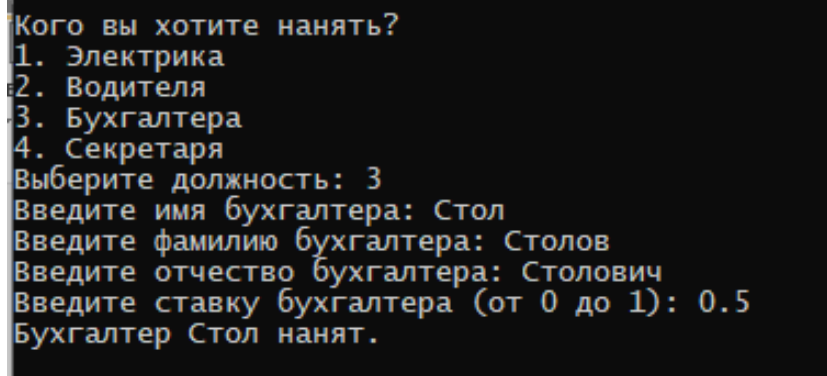
Рис. 3.12. Меню найма сотрудников



```
Кого вы хотите уволить?  
1. Электрика  
2. Водителя  
3. Бухгалтера  
4. Секретаря  
Выберите должность: 3  
Бухгалтер уволен.
```

Рис. 3.13. Меню увольнения сотрудников, процесс увольнения бухгалтера

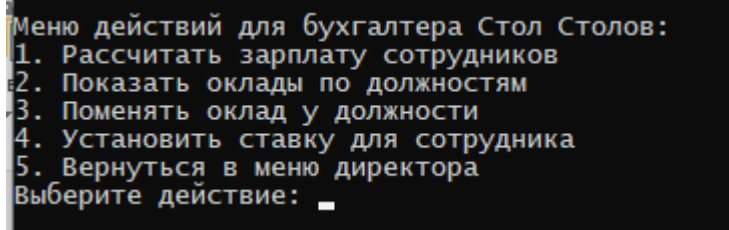
Переместимся обратно в меню найма сотрудников. Попробуем нанять бухгалтера (рис. 3.14).



```
Кого вы хотите нанять?  
1. Электрика  
2. Водителя  
3. Бухгалтера  
4. Секретаря  
Выберите должность: 3  
Введите имя бухгалтера: Стол  
Введите фамилию бухгалтера: Столов  
Введите отчество бухгалтера: Столович  
Введите ставку бухгалтера (от 0 до 1): 0.5  
Бухгалтер Стол нанят.
```

Рис. 3.14. Меню найма сотрудников - Процесс найма нового бухгалтера

Зайдём в меню нового бухгалтера (рис. 3.15), как мы видим бухгалтер был успешно изменён.



```
Меню действий для бухгалтера Стол Столов:  
1. Рассчитать зарплату сотрудников  
2. Показать оклады по должностям  
3. Поменять оклад у должности  
4. Установить ставку для сотрудника  
5. Вернуться в меню директора  
Выберите действие: 5
```

Рис. 3.15. Меню нового бухгалтера

Попробуем починить проводку электриком в подчинении текущего директора. Для этого в меню действий для директора выберем меню действий для электрика, перед этим выбрав одного из доступных электриков. В ходе проведения починки оборудования электрик может быть не в состоянии проводить работы, а также он может быть убит током, после чего, а также впоследствии чего – уволен (рис. 3.16 - 3.18).

```
Выберите электрика:  
1. Ивахненко Рустам  
2. Рустам Нургалеев  
3. Мавлей Амиров
```

Рис. 3.16. Выбор интересующего электрика

```
Электрик проверил проводку - устранил неисправности.
```

Рис. 3.17. Выполнение починки оборудования – устранение неисправностей

```
Электрик сейчас не в состоянии выполнять свою работу.
```

Рис. 3.18. Выполнение починки оборудования – электрик не в состоянии

```
Электрика убило током...  
Электрик Федоров уволен.
```

Рис. 3.19. Выполнение починки оборудования – смерть электрика

Попробуем поменять оборудование у электрика Ивахненко Рустам. Проверим изменения. Заметим то, что вводить все данные мы можем только с заглавной буквы, после которой идут строчные. Это исключение помогает обеспечить единообразный формат заполнения данных для удобного хранения в базе данных и наглядного вывода в таблицы или в строчки (рис.3.20, 3.21).

```
Подчинённые директора Блохоедов Кузьма:  
1. Бухгалтер: Нестеров Амур Сильвесторович; Ставка: 0.95  
2. Секретарь: Сидорова Рузалия Галлимулловна, Ставка: 0.7  
3. Водитель: Антипов Алексей Сергеевич; Ставка: 0.8  
Категории прав:  
Транспортные средства:  
4. Электрик: Ивахненко Рустам Милонович; Ставка: 0.6; Оборудование: Мультиметр  
5. Водитель: Смирнов Евгений Алексеевич; Ставка: 0.85  
Категории прав:
```

Рис. 3.20. Оборудование электрика до изменения

```
Подчинённые директора Блохоедов Кузьма:  
1. Бухгалтер: Нестеров Амур Сильвесторович; Ставка: 0.95  
2. Секретарь: Сидорова Рузалия Галлимулловна, Ставка: 0.7  
3. Водитель: Антипов Алексей Сергеевич; Ставка: 0.8  
Категории прав:  
Транспортные средства:  
4. Электрик: Ивахненко Рустам Милонович; Ставка: 0.6; Оборудование: Паяльник  
5. Водитель: Смирнов Евгений Алексеевич; Ставка: 0.85  
Категории прав:
```

Рис. 3.21. Оборудование электрика после изменения

ЗАКЛЮЧЕНИЕ

В ходе данной работы мне удалось самостоятельно разработать программу на языке программирования C++ с использованием методов и принципов объектно-ориентированного программирования.

ПРИЛОЖЕНИЯ

ПРИЛОЖЕНИЕ 1

Демонстрация диаграмм:

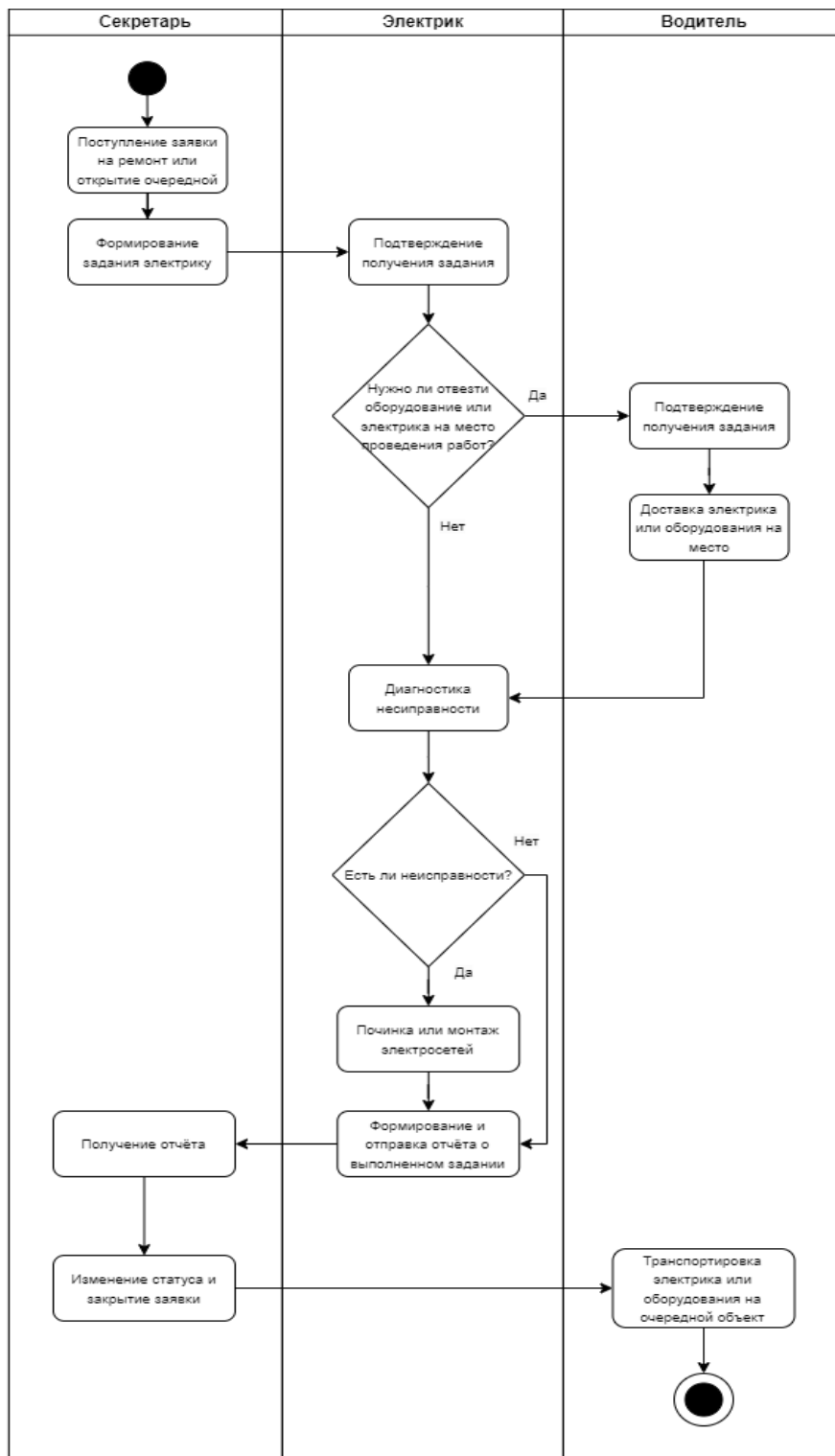


Рис. 1. Диаграмма для варианта использования "Ремонт оборудования на объекте по очередной заявке"

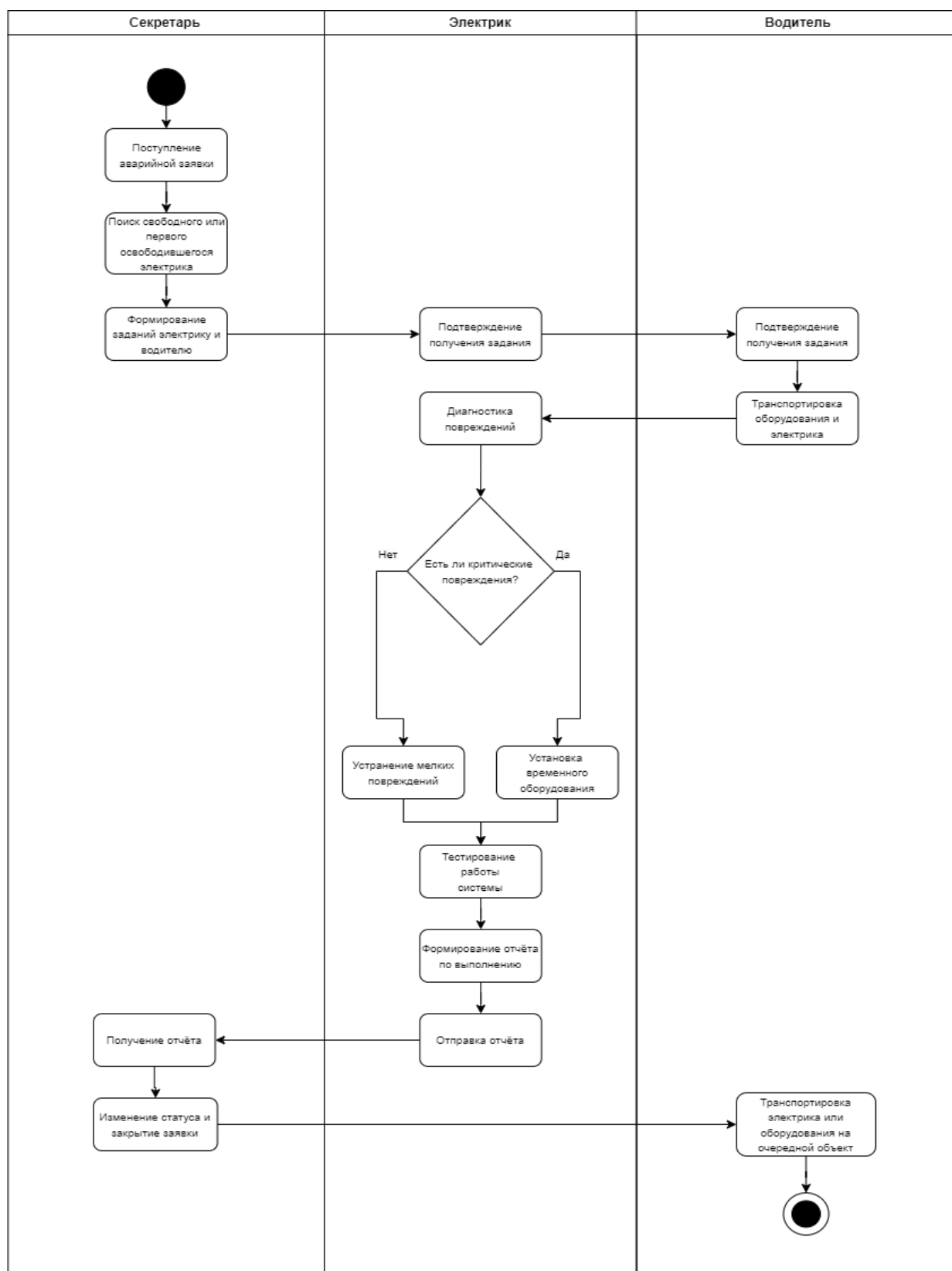


Рис. 2. Диаграмма деятельности для варианта использования "Ремонт или замена оборудования на объекте по аварийной заявке"

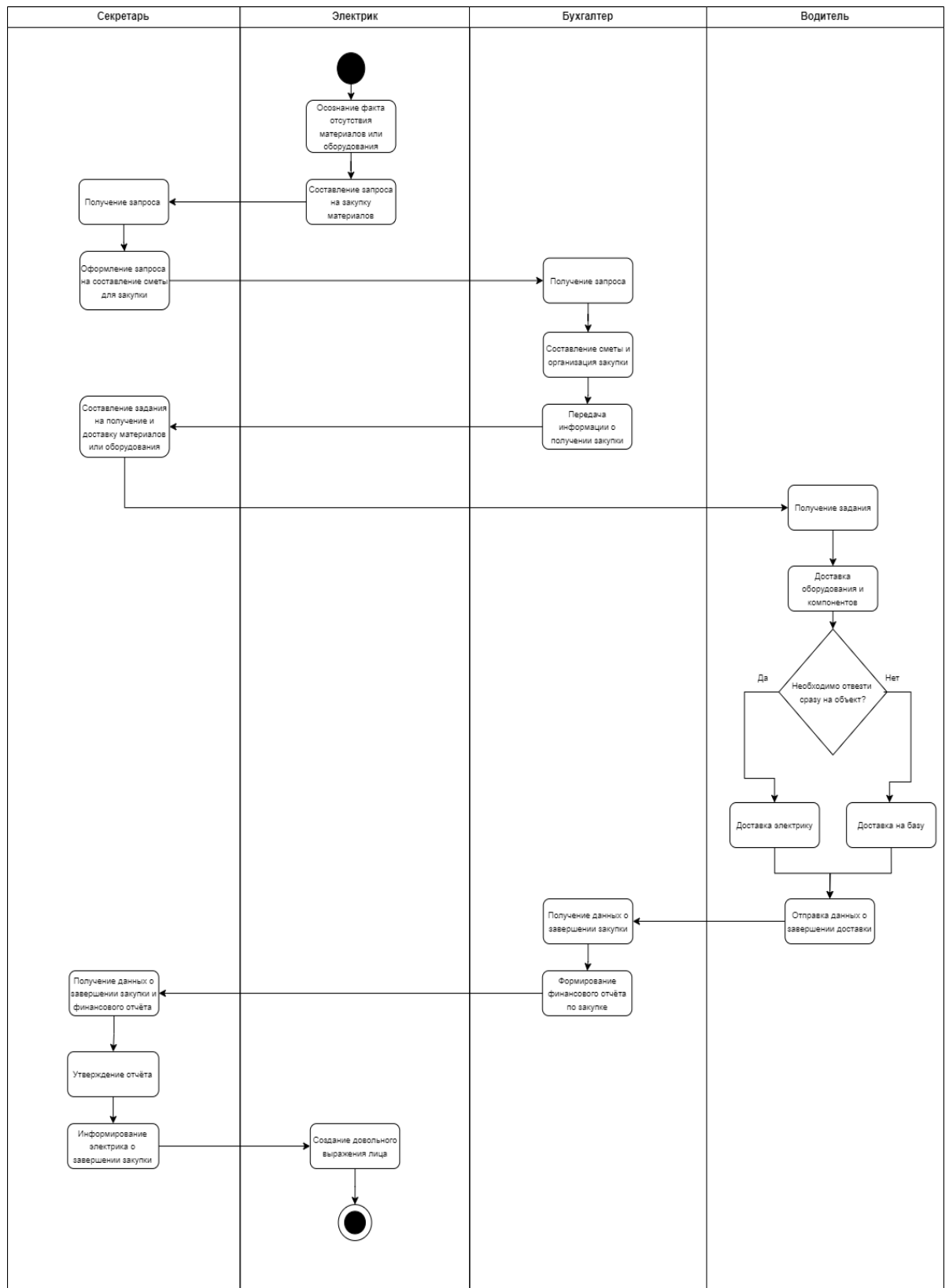


Рис. 3. Диаграмма деятельности для варианта использования "Отсутствие материалов или оборудования на складе или в наличии у электрика"

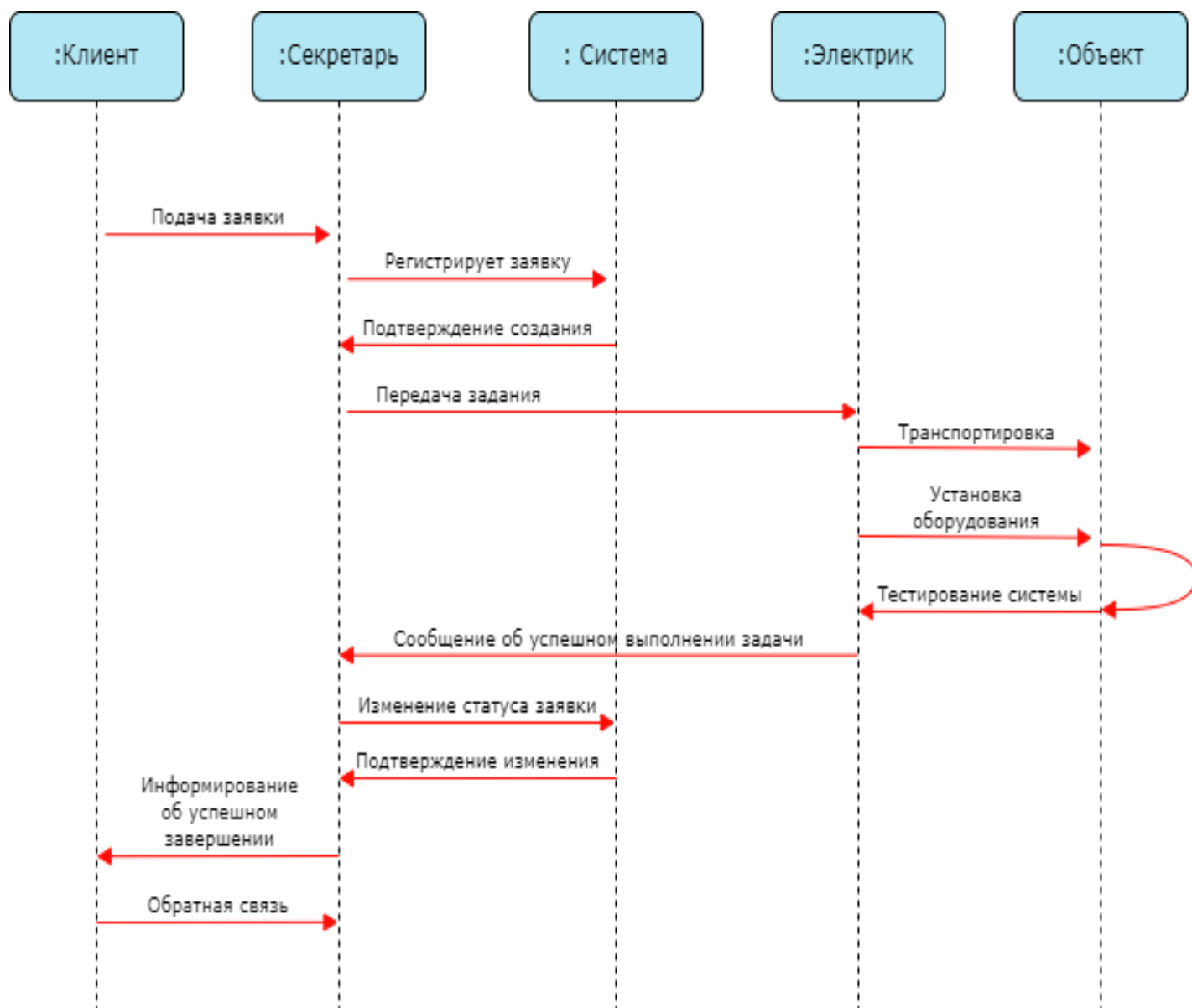


Рис. 4. Диаграмма последовательности для варианта использования "Обработка заявки на модернизацию оборудования на объекте"

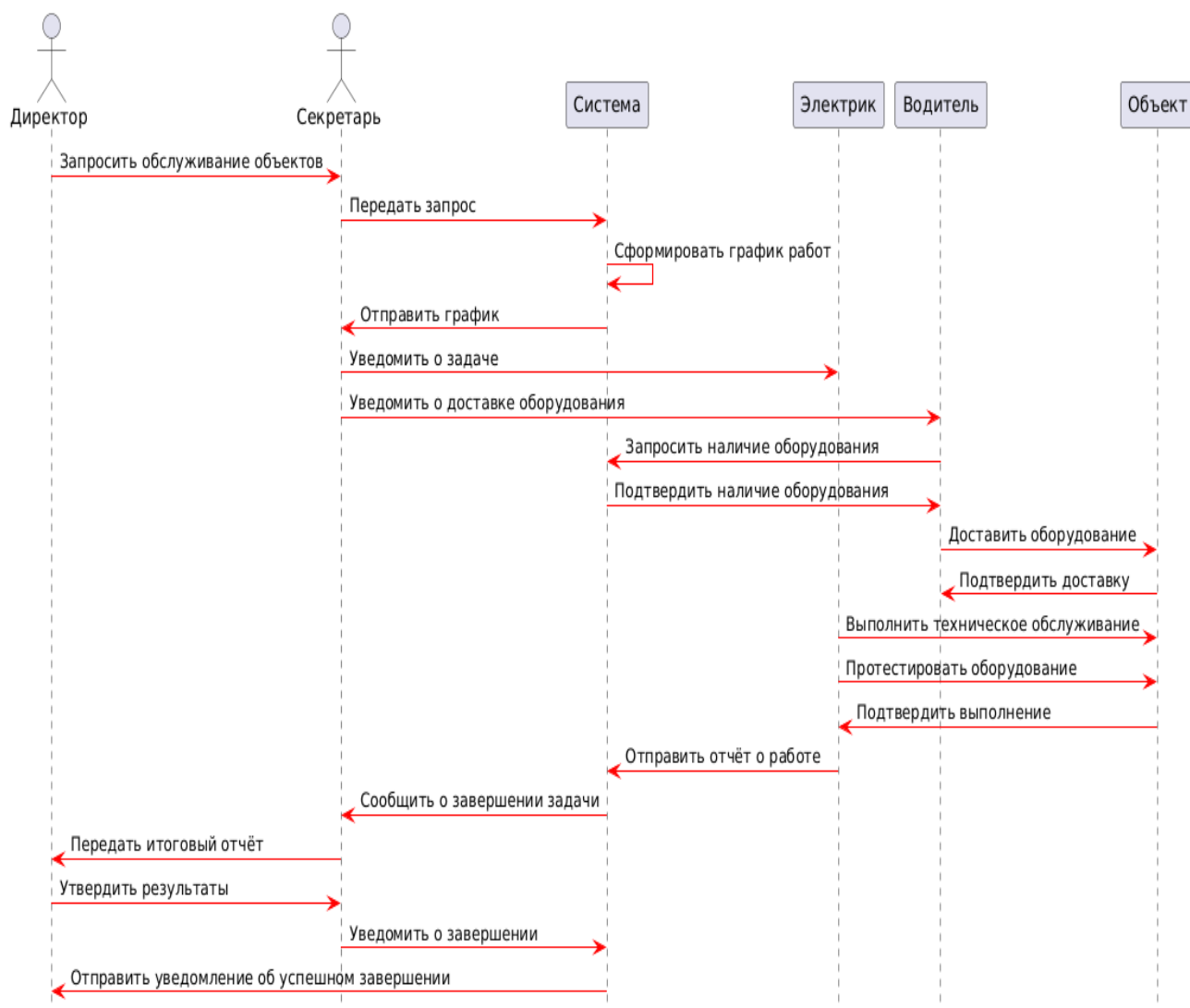


Рис. 5. Диаграмма последовательности для варианта использования "Организация обслуживания объектов гос. компании"

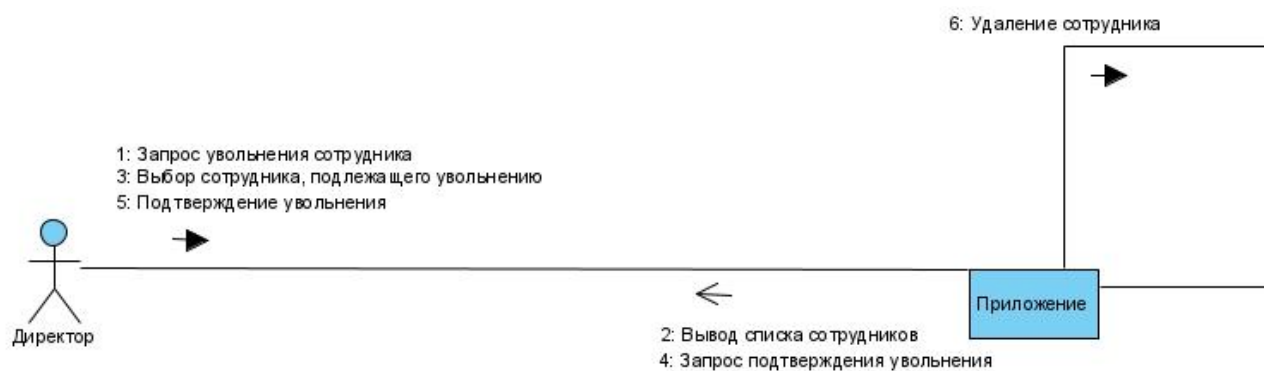


Рис. 6. Кооперативная диаграмма для варианта использования "Увольнение сотрудников"

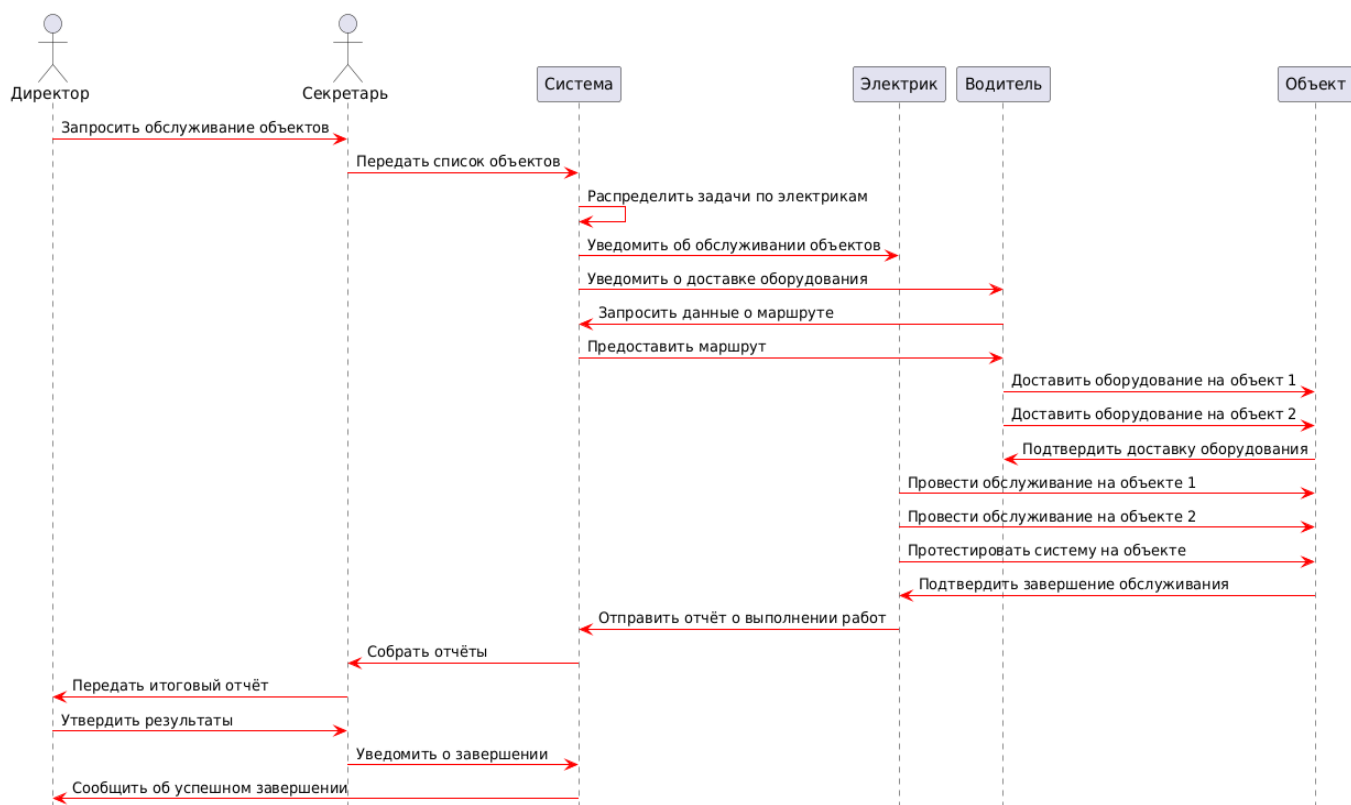


Рис. 7. Диаграмма последовательности для варианта использования "Масштабное техническое обслуживание нескольких объектов"

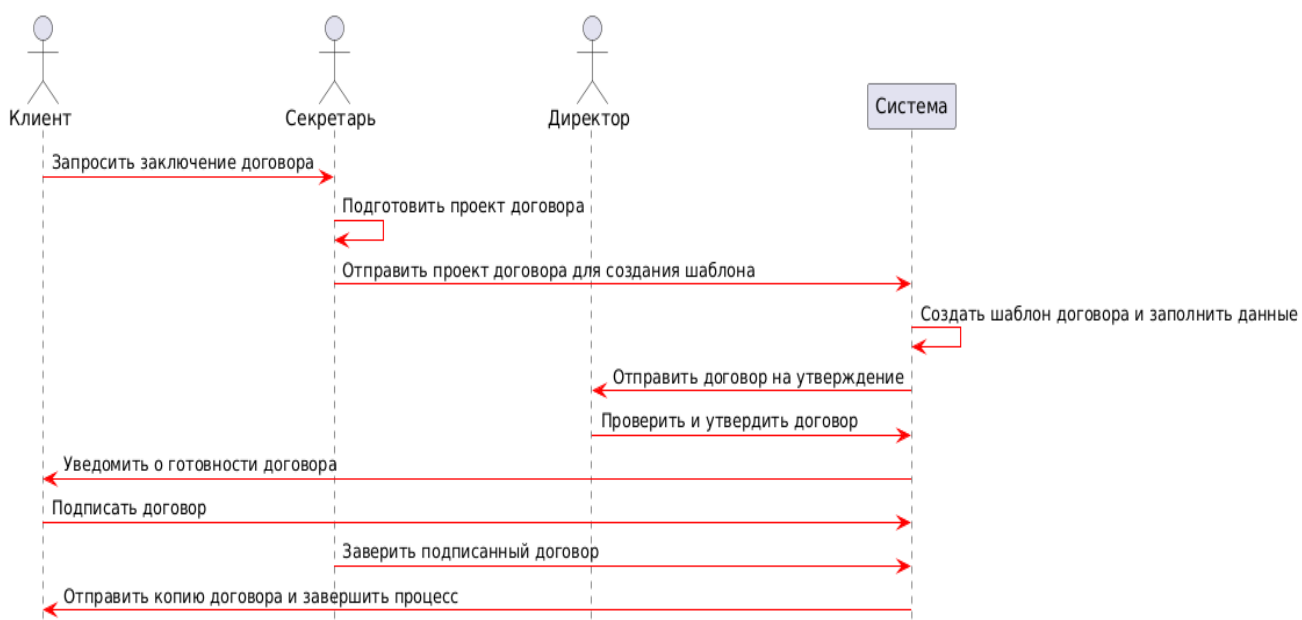


Рис. 8. Диаграмма последовательности для варианта использования "Согласование и подписание договора на обслуживание"

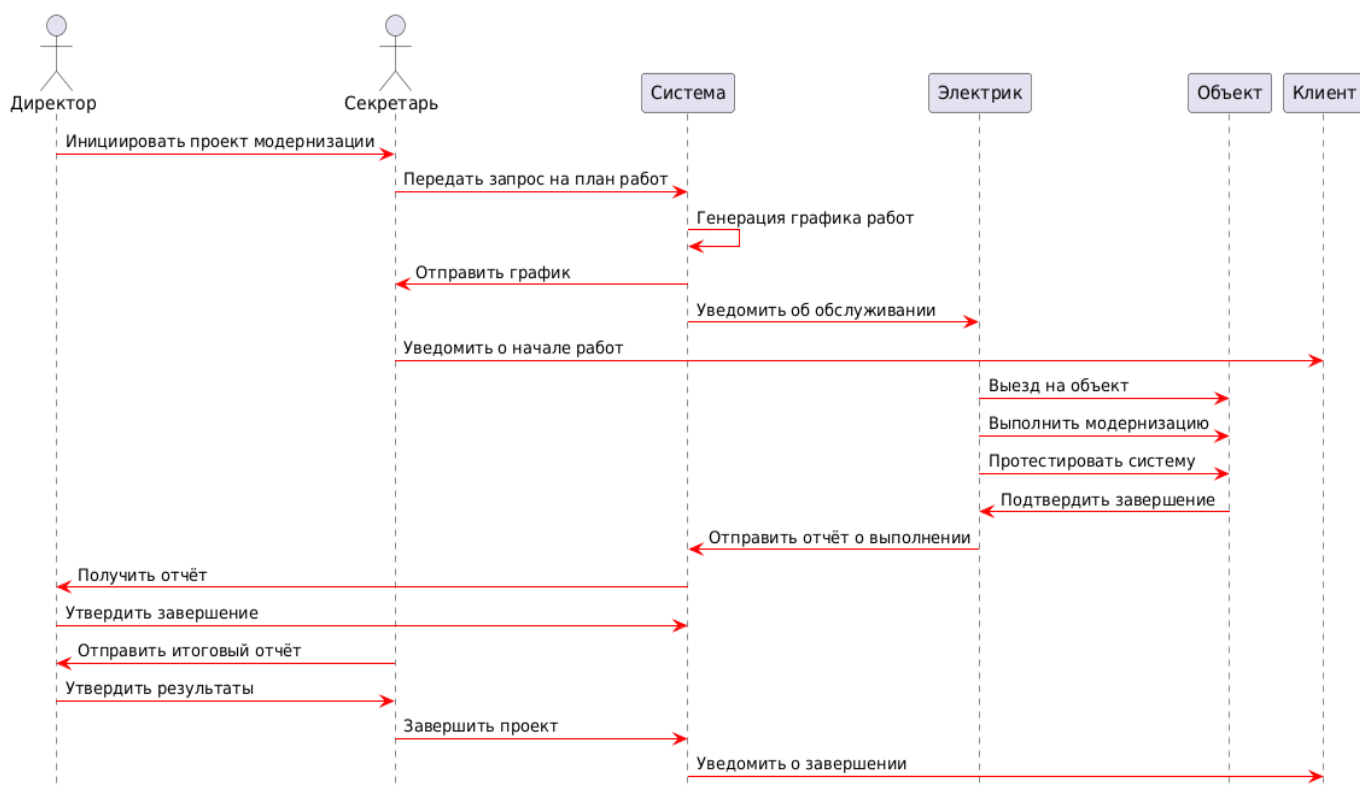


Рис. 9. Диаграмма последовательности для варианта использования "Модернизация энергосетей на объекте"

ПРИЛОЖЕНИЕ 2

Выводы таблиц:

Пример таблицы вывода электриков в отделе "Блохоедова".

Имя	Фамилия	Зарплата
Ивахненко	Рустам	40200
Орлов	Евгений	46230
Гусев	Илья	46900
Киселев	Семен	43550
Макаров	Владимир	48910
Андреев	Алексей	45560
Ковалев	Иван	40200
Ильин	Дмитрий	42880
Громов	Петр	48240
Мельников	Никита	44890
Григорьев	Антон	40870
Лазарев	Егор	44220
Мартынов	Борис	46900
Лебедев	Станислав	42210
Климов	Виктор	47570

Меню действий для секретаря Сидорова Рузалия:
1. Показать всех электриков
2. Показать всех водителей
3. Вернуться в меню директора
Выберите действие:

Рис. 1.1 Меню секретаря – Вывести всех электриков отдела "Блохоедова" в таблицу

Пример вывода всех охранников в таблицу.

Имя	Фамилия	Зарплата
Антипов	Алексей	56800
Смирнов	Евгений	60350
Кузнецов	Игорь	53960
Петров	Николай	63190
Иванов	Олег	58220
Соколов	Андрей	55380
Лебедев	Павел	63900
Новиков	Сергей	57510
Морозов	Виктор	52540
Попов	Александр	58930
Васильев	Денис	62480
Зайцев	Георгий	56090
Соловьев	Максим	61060
Волков	Дмитрий	53250
Медведев	Артур	59640
Крылов	Константин	56800

Меню действий для секретаря Сидорова Рузалия:
1. Показать всех электриков
2. Показать всех водителей
3. Вернуться в меню директора
Выберите действие: ☐

Рис. 1.2 Меню секретаря – Вывести всех водителей отдела "Блохоедова" в таблицу

ПРИЛОЖЕНИЕ 3

Листинг кода программы:

```
#include <iostream>
#include <string>
#include <vector>
#include <map>
#include <stdexcept>
#include <cctype>
#include <cstdlib>
#include <ctime>
#include <locale>
#include <algorithm>
#include <iomanip>
#include <fstream>
#include <sstream>

using namespace std;

bool isAlphaString(const string& str) {
    for (char c : str) {
        if (!isalpha(static_cast<unsigned char>(c))) {
            return false;
        }
    }
    return true;
}

bool isUpper(char c) {
    return (c >= 'A' && c <= 'Z') || (c >= 'А' && c <= 'Я');
}

bool isLower(char c) {
    return (c >= 'a' && c <= 'z') || (c >= 'а' && c <= 'я');
}

void getAlphaString(string& variable) {
    while (true) {
        cin >> variable;
```



```

        if (isAlphaString(variable)) {
            if (variable.length() > 20) {
                cout << "Слишком длинный ввод. Попробуйте снова: ";
            }
            else {
                if (isUpper(variable[0]) && all_of(variable.begin() + 1, variable.end(),
[] (char c) { return isLower(c); })) {
                    break;
                }
                else {
                    cout << "Ошибка: слово должно начинаться с заглавной буквы, а
остальные буквы должны быть строчными. Попробуйте снова: ";
                }
            }
        }
        else {
            cout << "Ошибка: поле должно содержать только буквы. Попробуйте снова: ";
        }
    }
}

bool isValidLicense(const string& license) {
    vector<string> validLicenses = { "A", "A1", "B", "B1", "C", "C1", "D", "D1", "BE",
"CE", "DE", "C1E", "D1E", "M", "Tm", "Tb" };
    for (const string& valid : validLicenses) {
        if (license == valid) {
            return true;
        }
    }
    return false;
}

template<typename T>
void getInput(T& variable) {
    while (!(cin >> variable) || variable > 1000000 || variable < 0) {
        cout << "Некорректный ввод. Попробуйте снова: ";
        cin.clear();
        cin.ignore(numeric_limits<streamsize>::max(), '\n');
    }
}

```

```

// виртуальный класс Human
class Human {
public:
    virtual ~Human() {}

    virtual string getName() const { return ""; }
    virtual string getSurname() const { return ""; }
    virtual string getPatronymic() const { return ""; }
    virtual double getRate() const { return 0.0; }
    virtual string getPosition() const { return ""; }

    virtual void setName(const string&) {}
    virtual void setSurname(const string&) {}
    virtual void setPatronymic(const string&) {}
    virtual void setRate(double) {}

    virtual void displayInfo() const {}
};

// Класс "Электрик"
class Electrician : public Human {
private:
    string name, surname, patronymic;
    double rate;
    string equipment;

public:
    Electrician(string n, string s, string p, double r, const string& eq)
        : name(n), surname(s), patronymic(p), rate(r), equipment(eq) {}

    string getName() const override { return name; }
    string getSurname() const override { return surname; }
    string getPatronymic() const override { return patronymic; }
    double getRate() const override { return rate; }
    string getPosition() const override { return "Электрик"; }

    void setName(const string& n) override { name = n; }
    void setSurname(const string& s) override { surname = s; }
    void setPatronymic(const string& p) override { patronymic = p; }

```

```

void setRate(double r) override {
    if (r < 0.1 || r > 1) {
        throw runtime_error("Ставка должна быть в диапазоне от 0.1 до 1.");
    }
    rate = r;
}

void setEquipment(const string& eq) {
    equipment = eq;
}

string getEquipment() const {
    return equipment;
}

bool fix_wiring() const {
    int randomOutcome = rand() % 4 + 1;
    switch (randomOutcome) {
        case 1:
            cout << "Электрик не обнаружил неисправности.\n";
            break;
        case 2:
            cout << "Электрик проверил проводку - устранил неисправности.\n";
            break;
        case 3:
            cout << "Электрик сейчас не в состоянии выполнять свою работу.\n";
            break;
        case 4:
            cout << "Электрика убило током...\n";
            return true;
        case 5:
            int item = rand() % 3;
            switch (item) {
                case 0: cout << "Электрик обнаружил короткое замыкание.\n"; break;
                case 1: cout << "Электрик обнаружил возгорание проводки.\n"; break;
                case 2: cout << "Электрик случайно обесточил весь город.\n"; break;
            }
            break;
    }
    return false;
}

```

```

    }

    void displayInfo() const override {
        cout << "Электрик: " << name << " " << surname << " " << patronymic << "; Ставка:
" << rate
        << "; Оборудование: " << equipment << "\n";
    }

    double calculateSalary(double baseSalary) const {
        return rate * baseSalary;
    }
};

// Класс "Водитель"
class Driver : public Human {
private:
    string name, surname, patronymic;
    double rate;
    vector<string> licenses;
    vector<string> vehicles;

public:
    Driver(string n, string s, string p, double r)
        : name(n), surname(s), patronymic(p), rate(r) {}

    string getName() const override { return name; }
    string getSurname() const override { return surname; }
    string getPatronymic() const override { return patronymic; }
    double getRate() const override { return rate; }
    string getPosition() const override { return "Водитель"; }

    void setName(const string& n) override { name = n; }
    void setSurname(const string& s) override { surname = s; }
    void setPatronymic(const string& p) override { patronymic = p; }
    void setRate(double r) override {
        if (r < 0.1 || r > 1) {
            throw runtime_error("Ставка должна быть в диапазоне от 0.1 до 1.");
        }
        rate = r;
    }
}

```

```

void addLicense(const string& license) {
    if (find(licenses.begin(), licenses.end(), license) != licenses.end()) {
        throw runtime_error("Эта категория прав уже добавлена.");
    }
    if (!isValidLicense(license)) {
        throw runtime_error("Недействительная категория прав.");
    }
    licenses.push_back(license);
}

void removeLicense(int index) {
    if (index >= 0 && index < licenses.size()) {
        licenses.erase(licenses.begin() + index);
    }
}

void addVehicle(const string& vehicle) {
    if (find(vehicles.begin(), vehicles.end(), vehicle) != vehicles.end()) {
        throw runtime_error("Это транспортное средство уже добавлено.");
    }
    if (!isAlphaString(vehicle)) {
        throw runtime_error("Наименование транспортного средства должно содержать
только буквы.");
    }
    vehicles.push_back(vehicle);
}

void removeVehicle(int index) {
    if (index >= 0 && index < vehicles.size()) {
        vehicles.erase(vehicles.begin() + index);
    }
}

vector<string> getLicenses() const { return licenses; }

vector<string> getVehicles() const { return vehicles; }

void fixVehicle() {
    if (vehicles.empty()) {

```

```

        cout << "У водителя еще нет транспортного средства.\n";
        return;
    }
    int outcome = rand() % 3;
    switch (outcome) {
        case 0: cout << "Машину быстро починили.\n"; break;
        case 1: cout << "Серьезная неисправность. Машина не подлежит починке.\n"; break;
        case 2: cout << "Машина в порядке.\n"; break;
    }
}

void displayInfo() const override {
    cout << "Водитель: " << name << " " << surname << " " << patronymic << "; Ставка: " << rate << "\n";
    cout << "Категории прав: ";
    for (const auto& license : licenses) {
        cout << license << " ";
    }
    cout << "\nТранспортные средства: ";
    for (const auto& vehicle : vehicles) {
        cout << vehicle << " ";
    }
    cout << "\n";
}

double calculateSalary(double baseSalary) const {
    return rate * baseSalary;
}

};

// Класс "Бухгалтер"
class Accountant : public Human {
private:
    string name, surname, patronymic;
    double rate;
    map<string, double> baseSalaries; // базовые зарплаты для разных должностей

public:
    Accountant(string n, string s, string p, double r)
        : name(n), surname(s), patronymic(p), rate(r) {

```

```

        baseSalaries["Электрик"] = 67000;
        baseSalaries["Водитель"] = 71000;
        baseSalaries["Секретарь"] = 87000;
        baseSalaries["Бухгалтер"] = 93000;
        baseSalaries["Директор"] = 115000;
    }

    string getName() const override { return name; }
    string getSurname() const override { return surname; }
    string getPatronymic() const override { return patronymic; }
    double getRate() const { return rate; }
    string getPosition() const { return "Бухгалтер"; }

    void setName(const string& n) override { name = n; }
    void setSurname(const string& s) override { surname = s; }
    void setPatronymic(const string& p) override { patronymic = p; }
    void setRate(double r) override {
        if (r < 0.1 || r > 1) {
            throw runtime_error("Ставка должна быть в диапазоне от 0.1 до 1.");
        }
        rate = r;
    }

    void setBaseSalary(const string& position, double salary) {
        if (salary < 17000) {
            throw runtime_error("Зарплата должна быть выше прожиточного минимума.");
        }
        baseSalaries[position] = salary;
    }

    double getBaseSalary(const string& position) const {
        if (baseSalaries.count(position)) {
            return baseSalaries.at(position);
        }
        return 0.0;
    }

    void displayInfo() const override {
        cout << "Бухгалтер: " << name << " " << surname << " " << patronymic << "; Ставка: " << rate << "\n";
    }

```

```

    }

    void displayBaseSalaries() const {
        cout << "Оклад для каждой должности:\n";
        for (const auto& baseSalary : baseSalaries) {
            cout << baseSalary.first << ": " << baseSalary.second << "\n";
        }
    }

    void changeBaseSalary() {
        cout << "Выберите должность для изменения базовой зарплаты:\n";
        cout << "1. Электрик\n";
        cout << "2. Водитель\n";
        cout << "3. Секретарь\n";
        cout << "4. Бухгалтер\n";
        cout << "5. Директор\n";

        int choice;
        getInput(choice);

        string position;
        switch (choice) {
            case 1: position = "Электрик"; break;
            case 2: position = "Водитель"; break;
            case 3: position = "Секретарь"; break;
            case 4: position = "Бухгалтер"; break;
            case 5: position = "Директор"; break;
            default:
                cout << "Некорректный выбор.\n";
                return;
        }

        double newSalary;
        cout << "Введите новую базовую зарплату для должности " << position << ": ";
        getInput(newSalary);

        try {
            setBaseSalary(position, newSalary);
            cout << "Базовая зарплата для должности " << position << " успешно
изменена.\n";

```



```

    }
    catch (const runtime_error& e) {
        cout << e.what() << endl;
    }
}

double calculateSalary(double baseSalary) const {
    return rate * baseSalary;
}
};

// Класс "Секретарь"
class Secretary : public Human {
private:
    string name, surname, patronymic;
    double rate;

public:
    Secretary(string n, string s, string p, double r)
        : name(n), surname(s), patronymic(p), rate(r) {}

    string getName() const override { return name; }
    string getSurname() const override { return surname; }
    string getPatronymic() const override { return patronymic; }
    double getRate() const override { return rate; }
    string getPosition() const override { return "Секретарь"; }

    void setName(const string& n) override { name = n; }
    void setSurname(const string& s) override { surname = s; }
    void setPatronymic(const string& p) override { patronymic = p; }
    void setRate(double r) override {
        if (r < 0.1 || r > 1) {
            throw runtime_error("Ставка должна быть в диапазоне от 0.1 до 1.");
        }
        rate = r;
    }

    void displayInfo() const override {

```

```

        cout << "Секретарь: " << name << " " << surname << " " << patronymic << ", Ставка:
" << rate << "\n";
    }

    double calculateSalary(double baseSalary) const {
        return rate * baseSalary;
    }
};

// Класс "Директор"
class Director : public Human {
private:
    string name, surname, patronymic;
    vector<Human*> employees; // Общий контейнер для всех сотрудников
    bool isFileData; // Атрибут для тестового директора

public:
    Director(string n, string s, string p) : name(n), surname(s), patronymic(p) {}
    Director(string n, string s, string p, bool testFlag) : name(n), surname(s),
    patronymic(p), isFileData(testFlag) {}

    string getName() const override { return name; }
    string getSurname() const override { return surname; }
    string getPatronymic() const override { return patronymic; }
    string getPosition() const override { return "Директор"; }
    bool getIsTestDirector() const { return isFileData; }

    void setName(const string& n) override { name = n; }
    void setSurname(const string& s) override { surname = s; }
    void setPatronymic(const string& p) override { patronymic = p; }

    // Проверка, есть ли сотрудник с данной должностью
    bool hasEmployeeWithPosition(const string& position) const {
        for (const auto& emp : employees) {
            if (emp->getPosition() == position) {
                return true;
            }
        }
        return false;
    }
}

```

```

// Получение списка сотрудников по должности
vector<Human*> getEmployeesByPosition(const string& position) const {
    vector<Human*> filteredEmployees;
    for (Human* employee : employees) {
        if (employee->getPosition() == position) {
            filteredEmployees.push_back(employee);
        }
    }
    return filteredEmployees;
}

Accountant* getAccountant() const {
    for (Human* emp : employees) {
        if (auto accountant = dynamic_cast<Accountant*>(emp)) {
            return accountant; // Возвращаем указатель на бухгалтера
        }
    }
    return nullptr; // Если бухгалтер не найден
}

Secretary* getSecretary() const {
    for (Human* emp : employees) {
        if (auto secretary = dynamic_cast<Secretary*>(emp)) {
            return secretary; // Возвращаем указатель на секретаря
        }
    }
    return nullptr; // Если секретарь не найден
}

// Получение всех электриков
vector<Human*> getElectricians() const {
    return getEmployeesByPosition("Электрик");
}

// Получение всех водителей
vector<Human*> getDrivers() const {
    return getEmployeesByPosition("Водитель");
}

```

```

// Найм электрика
void hireElectrician(Electrician* electrician) {
    employees.push_back(electrician);
    cout << "Электрик " << electrician->getName() << " нанят.\n";
}

// Найм водителя
void hireDriver(Driver* driver) {
    employees.push_back(driver);
    cout << "Водитель " << driver->getName() << " нанят.\n";
}

// Найм секретаря с проверкой на существование
void hireSecretary(Secretary* secretary) {
    if (hasEmployeeWithPosition("Секретарь")) {
        cout << "Секретарь уже нанят.\n";
    }
    else {
        employees.push_back(secretary);
        cout << "Секретарь " << secretary->getName() << " нанят.\n";
    }
}

// Найм бухгалтера с проверкой на существование
void hireAccountant(Accountant* accountant) {
    if (hasEmployeeWithPosition("Бухгалтер")) {
        cout << "Бухгалтер уже нанят.\n";
    }
    else {
        employees.push_back(accountant);
        cout << "Бухгалтер " << accountant->getName() << " нанят.\n";
    }
}

// Увольнение электрика по имени
void fireElectrician(const string& electricianName) {
    for (auto it = employees.begin(); it != employees.end(); ) {
        if ((*it)->getPosition() == "Электрик" && (*it)->getName() ==
electricianName) {
            delete* it; // освобождаем память

```

```

        it = employees.erase(it); // удаляем сотрудника и обновляем итератор
        cout << "Электрик " << electricianName << " уволен.\n";
        return; // Выходим из функции, так как Электрик уволен
    }
    else {
        ++it; // Переходим к следующему сотруднику
    }
}
cout << "Электрик с именем " << electricianName << " не найден.\n";
}

// Увольнение водителя по имени
void fireDriver(const string& driverName) {
    for (auto it = employees.begin(); it != employees.end(); ) {
        if ((*it)->getPosition() == "Водитель" && (*it)->getName() == driverName) {
            delete* it; // освобождаем память
            it = employees.erase(it); // удаляем сотрудника и обновляем итератор
            cout << "Водитель " << driverName << " уволен.\n";
            return; // Выходим из функции, так как водитель уволен
        }
        else {
            ++it; // Переходим к следующему сотруднику
        }
    }
    cout << "Водитель с именем " << driverName << " не найден.\n";
}

// Увольнение секретаря
void fireSecretary() {
    for (auto it = employees.begin(); it != employees.end(); ) {
        if ((*it)->getPosition() == "Секретарь") {
            delete* it; // освобождаем память
            it = employees.erase(it); // удаляем сотрудника и обновляем итератор
            cout << "Секретарь уволен.\n";
            return; // Выходим из функции, так как секретарь уволен
        }
        else {
            ++it; // Переходим к следующему сотруднику
        }
    }
}

```

```

        cout << "Секретарь не был нанят.\n";
    }

    // Увольнение бухгалтера
    void fireAccountant() {
        for (auto it = employees.begin(); it != employees.end(); ) {
            if ((*it)->getPosition() == "Бухгалтер") {
                delete* it; // освобождаем память
                it = employees.erase(it); // удаляем сотрудника и обновляем итератор
                cout << "Бухгалтер уволен.\n";
                return; // Выходим из функции, так как бухгалтер уволен
            }
            else {
                ++it; // Переходим к следующему сотруднику
            }
        }
        cout << "Бухгалтер не был нанят.\n";
    }

    // Вывод информации о подчинённых
    void displaySubordinates() const {
        cout << "Подчинённые директора " << name << " " << surname << ":\n";
        if (employees.empty()) {
            cout << "Нет подчинённых.\n";
        }
        else {
            int index = 1;
            for (const auto& employee : employees) {
                if (employee != nullptr) { // Проверка указателя
                    cout << index++ << ". ";
                    employee->displayInfo();
                }
                else {
                    cout << "Ошибка: Обнаружен некорректный указатель на сотрудника.\n";
                }
            }
        }
    }

    void displayInfo() const override {

```

```

// Находим бухгалтера среди сотрудников
Human* accountant = nullptr;
for (auto* emp : employees) {
    if (emp->getPosition() == "Бухгалтер") {
        accountant = emp;
        break;
    }
}
// Если бухгалтер не найден, выводим основную информацию
if (accountant == nullptr) {
    cout << "Директор: " << name << " " << surname << " " << patronymic << "\n";
}
else {
    // Получаем базовый оклад у найденного бухгалтера
    double baseSalary = dynamic_cast<Accountant*>(accountant)-
>getBaseSalary("Директор");
    cout << "Директор: " << name << " " << surname << " " << patronymic << "; ЗП:
" << baseSalary << "\n";
}
}

};

// подкласс бухгалтера
class SubclassAccountant : public Accountant {
public:
    void displaySalaries(Director& director) const {
        cout << "Зарплаты сотрудников:\n";

        // Зарплата директора
        double directorSalary = getBaseSalary("Директор");
        cout << director.getName() << " " << director.getSurname() << " " <<
director.getPosition()
        << ": " << directorSalary << "\n";

        // Зарплата бухгалтера
        double accountantSalary = getRate() * getBaseSalary("Бухгалтер");
        cout << getName() << " " << getSurname() << " " << getPosition() << ": " <<
accountantSalary << "\n";
    }
};

```

```

// Зарплата секретаря
auto secretaries = director.getEmployeesByPosition("Секретарь");
if (!secretaries.empty()) {
    for (const auto* emp : secretaries) {
        const Secretary* secretary = dynamic_cast<const Secretary*>(emp);
        if (secretary) {
            double salary = secretary->getRate() * getBaseSalary("Секретарь");
            cout << secretary->getName() << " " << secretary->getSurname()
                << " Секретарь: " << salary << "\n";
        }
    }
}
else {
    cout << "Секретарь не найден.\n";
}

// Зарплата электриков
auto electricians = director.getEmployeesByPosition("Электрик");
if (!electricians.empty()) {
    for (const auto* electrician : electricians) {
        double salary = electrician->getRate() * getBaseSalary("Электрик");
        cout << electrician->getName() << " " << electrician->getSurname()
            << " Электрик: " << salary << "\n";
    }
}
else {
    cout << "Электрики не найдены.\n";
}

// Зарплата водителей
auto drivers = director.getEmployeesByPosition("Водитель");
if (!drivers.empty()) {
    for (const auto* driver : drivers) {
        double salary = driver->getRate() * getBaseSalary("Водитель");
        cout << driver->getName() << " " << driver->getSurname()
            << " Водитель: " << salary << "\n";
    }
}
else {
    cout << "Водители не найдены.\n";
}

```



```

    }
}

void changeEmployeeRate(Director& director) {
    auto secretaries = director.getEmployeesByPosition("Секретарь");
    auto electricians = director.getEmployeesByPosition("Электрик");
    auto drivers = director.getEmployeesByPosition("Водитель");

    // Проверка на наличие сотрудников
    if (secretaries.empty() && electricians.empty() && drivers.empty()) {
        cout << "Некому менять ставку.\n";
        return;
    }

    cout << "Выберите сотрудника для изменения ставки:\n";
    int index = 1;

    // Бухгалтер
    cout << index++ << ". " << getName() << " " << getSurname()
        << " - Бухгалтер, Ставка: " << getRate() << "\n";

    // Секретари
    for (auto* emp : secretaries) {
        Secretary* secretary = dynamic_cast<Secretary*>(emp);
        if (secretary) {
            // Доступ к данным секретаря
            cout << index++ << ". " << secretary->getName() << " " << secretary-
>getSurname()
                << " - Секретарь, Ставка: " << secretary->getRate() << "\n";
        }
    }

    // Электрики
    for (auto* Electrician : electricians) {
        cout << index++ << ". " << Electrician->getName() << " " << Electrician-
>getSurname()
                << " - Электрик, Ставка: " << Electrician->getRate() << "\n";
    }

    // Водители

```

```

for (auto* driver : drivers) {
    cout << index++ << ". " << driver->getName() << " " << driver->getSurname()
        << " - Водитель, Ставка: " << driver->getRate() << "\n";
}

int choice;
cout << "Введите номер сотрудника для изменения ставки: ";
getInput(choice);

if (choice > 0 && choice < index) {
    double newRate;
    cout << "Введите новую ставку (от 0 до 1): ";
    getInput(newRate);

    if (newRate < 0 || newRate > 1) {
        cout << "Некорректное значение ставки. Она должна быть в диапазоне от 0
до 1.\n";
        return;
    }

    if (choice == 1) {
        setRate(newRate);
        cout << "Ставка бухгалтера изменена на " << newRate << "\n";
    }

    if (choice <= secretaries.size() + 1 && choice != 1) {
        secretaries[choice - 2]->setRate(newRate);
        cout << "Ставка секретаря изменена на " << newRate << "\n";
    }
    else if (choice <= secretaries.size() + electricians.size() + 1 && choice !=
1) {
        electricians[choice - secretaries.size() - 2]->setRate(newRate);
        cout << "Ставка электрика изменена на " << newRate << "\n";
    }
    else if (choice <= secretaries.size() + electricians.size() + drivers.size()
+ 1 && choice != 1) {
        drivers[choice - secretaries.size() - electricians.size() - 2]-
>setRate(newRate);
        cout << "Ставка водителя изменена на " << newRate << "\n";
    }
}

```

```

    }
    else {
        cout << "Некорректный выбор.\n";
    }
}
};

// подкласс секретаря
class SubclassSecretary : public Secretary {
public:
    void displayElectricians(const Director& director) const {
        vector<Human*> electricians = director.getElectricians(); // Используем Human*
        Accountant* accountant = director.getAccountant(); // Добавляем метод в Director

        if (!accountant) {
            cout << "Бухгалтер не назначен.\n";
            return;
        }

        if (electricians.empty()) {
            cout << "Нет электриков для отображения.\n";
            return;
        }

        cout << "+-----+-----+-----+
+\\n";
        cout << "|          Имя          |          Фамилия          |          Зарплата          |\\n";
        cout << "+-----+-----+-----+
+\\n";

        for (const auto& emp : electricians) {
            Electrician* electrician = dynamic_cast<Electrician*>(emp); // Приведение
типов
            if (electrician) { // Проверяем, что приведение прошло успешно
                double baseSalary = accountant->getBaseSalary("Электрик");
                double salary = electrician->calculateSalary(baseSalary);
                cout << "| " << setw(20) << electrician->getName() << " | "
                    << setw(20) << electrician->getSurname() << " | "
                    << setw(20) << salary << " |\\n";
            }
        }
    }
};

```

```

    }

    cout << "+-----+-----+-----+
+\\n";
    }

    void displayDrivers(const Director& director) const {
        vector<Human*> drivers = director.getDrivers(); // Используем Human*
        Accountant* accountant = director.getAccountant(); // Добавляем метод в Director

        if (!accountant) {
            cout << "Бухгалтер не назначен.\\n";
            return;
        }

        if (drivers.empty()) {
            cout << "Нет водителей для отображения.\\n";
            return;
        }

        cout << "+-----+-----+-----+
+\\n";

        cout << "|          Имя          |          Фамилия          |          Зарплата          |\\n";
        cout << "+-----+-----+-----+
+\\n";

        for (const auto& emp : drivers) {
            Driver* driver = dynamic_cast<Driver*>(emp); // Приведение типов
            if (driver) { // Проверяем, что приведение прошло успешно
                double baseSalary = accountant->getBaseSalary("Водитель");
                double salary = driver->calculateSalary(baseSalary);
                cout << "| " << setw(20) << driver->getName() << " | "
                    << setw(20) << driver->getSurname() << " | "
                    << setw(20) << salary << " |\\n";
            }
        }

        cout << "+-----+-----+-----+
+\\n";
    }

```

```

};

// меню действий для бухгалтера
void accountantActions(Director& director) {
    if (director.getAccountant() == nullptr) {
        cout << "Нет нанятого бухгалтера.\n";
        return;
    }

    Accountant* accountant = director.getAccountant();
    SubclassAccountant* subclassAccountant =
static_cast<SubclassAccountant*>(accountant);

    int action;
    do {
        cout << "\nМеню действий для бухгалтера " << accountant->getName() << " " <<
accountant->getSurname() << ":\n";
        cout << "1. Рассчитать зарплату сотрудников\n";
        cout << "2. Показать оклады по должностям\n";
        cout << "3. Поменять оклад у должности\n";
        cout << "4. Установить ставку для сотрудника\n";
        cout << "5. Вернуться в меню директора\n";
        cout << "Выберите действие: ";
        getInput(action);
        system("cls");

        switch (action) {
            case 1:
                subclassAccountant->displaySalaries(director);
                break;
            case 2:
                accountant->displayBaseSalaries();
                break;
            case 3: {
                accountant->changeBaseSalary();
                break;
            }
            case 4:
                subclassAccountant->changeEmployeeRate(director);

```

```

        break;
    case 5:
        cout << "Возврат в меню директора\n";
        break;
    default:
        cout << "Некорректный выбор. Попробуйте снова.\n";
    }
} while (action != 5);
}

// меню действий для секретаря
void secretaryActions(Director& director) {
    if (director.getSecretary() == nullptr) {
        cout << "Нет нанятого секретаря.\n";
        return;
    }
    Secretary* secretary = director.getSecretary();
    SubclassSecretary* subclassSecretary = dynamic_cast<SubclassSecretary*>(secretary);

    int action;
    do {
        cout << "\nМеню действий для секретаря " << secretary->getName() << " " <<
secretary->getSurname() << ":\n";
        cout << "1. Показать всех электриков\n";
        cout << "2. Показать всех водителей\n";
        cout << "3. Вернуться в меню директора\n";
        cout << "Выберите действие: ";
        getInput(action);
        system("cls");

        switch (action) {
            case 1:
                subclassSecretary->displayElectricians(director);
                break;
            case 2:
                subclassSecretary->displayDrivers(director);
                break;
            case 3:
                cout << "Возврат в меню директора.\n";
                break;
        }
    } while (action != 3);
}

```

```

        default:
            cout << "Некорректный выбор.\n";
        }
    } while (action != 3);
}

// меню действия для водителей
void driverActions(Director& director) {
    if (director.getDrivers().empty()) {
        cout << "Нет нанятых водителей.\n";
        return;
    }

    cout << "Выберите водителя:\n";
    for (size_t i = 0; i < director.getDrivers().size(); ++i) {
        Driver* driver = dynamic_cast<Driver*>(director.getDrivers()[i]); // Приводим к
        типу Driver*
        if (driver) { // Проверяем, успешно ли произошло приведение
            cout << i + 1 << ". " << driver->getName() << " " << driver->getSurname() <<
            "\n";
        }
        else {
            cout << "Ошибка: Не удалось привести тип к Driver для индекса " << i + 1 <<
            ".\n";
        }
    }

    int driverChoice;
    getInput(driverChoice);
    system("cls");
    if (driverChoice > 0 && driverChoice <= director.getDrivers().size()) {
        Driver* driverPtr = dynamic_cast<Driver*>(director.getDrivers()[driverChoice -
        1]); // Приведение к Driver*

        if (driverPtr) { // Проверяем успешность приведения
            Driver& driver = *driverPtr; // Разыменовываем указатель, чтобы получить
            ссылку на Driver

            int action;
            do {

```

```

        cout << "\nМеню действий для водителя " << driver.getName() << " " <<
driver.getSurname() << ":\n";
        cout << "1. Добавить категорию прав\n";
        cout << "2. Убрать категорию прав\n";
        cout << "3. Добавить транспортное средство\n";
        cout << "4. Убрать транспортное средство\n";
        cout << "5. Вернуться в меню директора\n";
        cout << "Выберите действие: ";
        getInput(action);
        system("cls");

        switch (action) {
        case 1: {
            string license;
            cout << "Переключите раскладку на английский.\n";
            cout << "Введите категорию прав: ";
            cin >> license;
            cout << "Верните раскладку на русский.\n";
            try {
                driver.addLicense(license);
                cout << "Категория прав добавлена.\n";
            }
            catch (const runtime_error& e) {
                cout << e.what() << endl;
            }
            break;
        }
        case 2: {
            if (driver.getLicenses().empty()) {
                cout << "Нет категорий прав для удаления.\n";
            }
            else {
                cout << "Выберите категорию для удаления:\n";
                for (size_t i = 0; i < driver.getLicenses().size(); ++i) {
                    cout << i + 1 << ". " << driver.getLicenses()[i] << "\n";
                }
                int licenseChoice;
                getInput(licenseChoice);
                if (licenseChoice > 0 && licenseChoice <=
driver.getLicenses().size()) {

```



```

        driver.removeLicense(licenseChoice - 1);
        cout << "Категория прав удалена.\n";
    }
    else {
        cout << "Некорректный выбор.\n";
    }
}
break;
}
case 3: {
    string vehicle;
    cout << "Введите название транспортного средства (с большой буквы):
";

    getAlphaString(vehicle);
    driver.addVehicle(vehicle);
    cout << "Транспортное средство добавлено.\n";
    break;
}
case 4: {
    if (driver.getVehicles().empty()) {
        cout << "Нет транспортных средств для удаления.\n";
    }
    else {
        cout << "Выберите транспортное средство для удаления:\n";
        for (size_t i = 0; i < driver.getVehicles().size(); ++i) {
            cout << i + 1 << ". " << driver.getVehicles()[i] << "\n";
        }
        int vehicleChoice;
        getInput(vehicleChoice);
        if (vehicleChoice > 0 && vehicleChoice <=
driver.getVehicles().size()) {
            driver.removeVehicle(vehicleChoice - 1);
            cout << "Транспортное средство удалено.\n";
        }
        else {
            cout << "Некорректный выбор.\n";
        }
    }
    break;
}
}

```

```

        case 5:
            cout << "Возврат в меню директора.\n";
            break;
        default:
            cout << "Некорректный выбор.\n";
        }
    } while (action != 5);
}
}
else {
    cout << "Некорректный выбор.\n";
}
}

// меню для действий для электриков
void guardActions(Director& director) {
    if (director.getElectricians().empty()) {
        cout << "Нет нанятых электриков.\n";
        return;
    }

    cout << "Выберите электрика:\n";
    for (size_t i = 0; i < director.getElectricians().size(); ++i) {
        Electrician* electrician =
dynamic_cast<Electrician*>(director.getElectricians()[i]); // Приведение к типу
Electrician*
        if (electrician) { // Проверка на успешность приведения
            cout << i + 1 << ". " << electrician->getName() << " " << electrician-
>getSurname() << "\n";
        }
        else {
            cout << "Ошибка: Не удалось привести тип к Electrician для индекса " << i +
1 << ".\n";
        }
    }

    int guardChoice;
    getInput(guardChoice);
    system("cls");
}

```

```

        if (guardChoice > 0 && guardChoice <= director.getElectricians().size()) {
            Electrician*                                guardPtr                                =
dynamic_cast<Electrician*>(director.getElectricians()[guardChoice - 1]); // Приведение к
Electrician*

            if (guardPtr) { // Проверяем успешность приведения
                Electrician& Electrician = *guardPtr; // Разыменовываем указатель, чтобы
получить ссылку на Electrician

                int action;
                string equipment;
                bool disappeared = false;
                do {
                    cout << "\nМеню действий для электрика " << Electrician.getName() << " "
<< Electrician.getSurname() << ":\n";
                    cout << "1. Поменять оборудование\n";
                    cout << "2. Совершить починку проводки\n";
                    cout << "3. Вернуться в меню директора\n";
                    cout << "Выберите действие: ";
                    getInput(action);
                    system("cls");

                    switch (action) {
                        case 1:
                            cout << "Введите новое оборудование (с большой буквы): ";
                            getAlphaString(equipment);
                            Electrician.setEquipment(equipment);
                            cout << "Оборудование изменено.\n";
                            break;
                        case 2: {
                            disappeared = Electrician.fix_wiring();
                            if (disappeared) { director.fireElectrician(Electrician.getName());
}

                            break;
                        }
                        case 3:
                            cout << "Возврат в меню директора.\n";
                            break;
                        default:
                            cout << "Некорректный выбор. Попробуйте снова.\n";

```

```

        }
    } while (action != 3 && !disappeared);
}
else {
    cout << "Некорректный выбор.\n";
}
}
}

// меню управления для директора
void directorMenu(Director& director) {
    int choice;
    do {
        cout << "\nМеню директора " << director.getName() << ":\n";
        cout << "1. Показать информацию о директоре\n";
        cout << "2. Показать подчинённых\n";
        cout << "3. Нанять сотрудника\n";
        cout << "4. Уволить сотрудника\n";
        cout << "5. Действия для водителя\n";
        cout << "6. Действия для электрика\n";
        cout << "7. Действия для секретаря\n";
        cout << "8. Действия для бухгалтера\n";
        cout << "9. Вернуться в main меню\n";
        cout << "Выберите действие: ";
        getInput(choice);
        system("cls");

        switch (choice) {
            case 1:
                director.displayInfo();
                break;
            case 2:
                if (director.getAccountant() == nullptr && director.getSecretary() == nullptr
&&
                    director.getElectricians().empty() && director.getDrivers().empty()) {
                    cout << "Нет подчинённых.\n";
                }
            else {
                director.displaySubordinates();
            }
        }
    }
}

```

```

        break;
    case 3: {
        int hireChoice;
        cout << "\nКого вы хотите нанять?\n";
        cout << "1. Электрика\n";
        cout << "2. Водителя\n";
        cout << "3. Бухгалтера\n";
        cout << "4. Секретаря\n";
        cout << "Выберите должность: ";
        getInput(hireChoice);

        if (hireChoice == 1) {
            string name, surname, patronymic, equipment;
            double rate;
            cout << "Введите имя электрика: ";
            getAlphaString(name);
            cout << "Введите фамилию электрика: ";
            getAlphaString(surname);
            cout << "Введите отчество электрика: ";
            getAlphaString(patronymic);
            do {
                cout << "Введите ставку электрика (от 0 до 1): ";
                getInput(rate);
                if (rate < 0 || rate > 1) {
                    cout << "Ошибка: ставка должна быть в диапазоне от 0 до 1.\n";
                }
            } while (rate < 0.1 || rate > 1);
            cout << "Введите оборудование электрика: ";
            getAlphaString(equipment);
            director.hireElectrician(new Electrician(name, surname, patronymic,
rate, equipment));
        }
        else if (hireChoice == 2) {
            string name, surname, patronymic;
            double rate;
            cout << "Введите имя водителя: ";
            getAlphaString(name);
            cout << "Введите фамилию водителя: ";
            getAlphaString(surname);
            cout << "Введите отчество водителя: ";

```

```

getAlphaString(patronymic);
do {
    cout << "Введите ставку водителя (от 0 до 1): ";
    getInput(rate);
    if (rate < 0 || rate > 1) {
        cout << "Ошибка: ставка должна быть в диапазоне от 0 до 1.\n";
    }
} while (rate < 0.1 || rate > 1);

Driver* newDriver = new Driver(name, surname, patronymic, rate);

// ввод категорий прав
char addMore;
do {
    string license;
    cout << "Введите категорию прав (или '0' для завершения ввода): ";
    cin >> license;
    if (license == "0") break;
    try {
        newDriver->addLicense(license); // Используем указатель
    }
    catch (const runtime_error& e) {
        cout << e.what() << endl;
    }
    cout << "Хотите добавить еще категорию прав? (1 - да/0 - нет): ";
    cin >> addMore;
} while (addMore == '1');

// ввод транспортных средств
do {
    string vehicle;
    cout << "Введите название транспортного средства (или '0' для
завершения ввода): ";
    getAlphaString(vehicle);
    if (vehicle == "0") break;
    try {
        newDriver->addVehicle(vehicle); // Используем указатель
    }
    catch (const runtime_error& e) {
        cout << e.what() << endl;
    }
}

```

```

    }
    cout << "Хотите добавить еще транспортное средство? (1 - да/0 - нет): ";

    cin >> addMore;
} while (addMore == '1');

director.hireDriver(newDriver);
}
else if (hireChoice == 3) {
    if (director.getAccountant() == nullptr) {
        string name, surname, patronymic;
        double rate;
        cout << "Введите имя бухгалтера: ";
        getAlphaString(name);
        cout << "Введите фамилию бухгалтера: ";
        getAlphaString(surname);
        cout << "Введите отчество бухгалтера: ";
        getAlphaString(patronymic);
        do {
            cout << "Введите ставку бухгалтера (от 0 до 1): ";
            getInput(rate);
            if (rate < 0 || rate > 1) {
                cout << "Ошибка: ставка должна быть в диапазоне от 0 до 1.\n";
            }
        } while (rate < 0.1 || rate > 1);
        director.hireAccountant(new Accountant(name, surname, patronymic,
rate));
    }
    else {
        cout << "Бухгалтер уже нанят.\n";
    }
}
else if (hireChoice == 4) {
    if (director.getSecretary() == nullptr) {
        string name, surname, patronymic;
        double rate;
        cout << "Введите имя секретаря: ";
        getAlphaString(name);
        cout << "Введите фамилию секретаря: ";
        getAlphaString(surname);

```

```

        cout << "Введите отчество секретаря: ";
        getAlphaString(patronymic);
        do {
            cout << "Введите ставку секретаря (от 0 до 1): ";
            getInput(rate);
            if (rate < 0 || rate > 1) {
                cout << "Ошибка: ставка должна быть в диапазоне от 0 до 1.\n";
            }
        } while (rate < 0.1 || rate > 1);
        director.hireSecretary(new Secretary(name, surname, patronymic,
rate));

    }
    else {
        cout << "Секретарь уже нанят.\n";
    }
}
break;
}
case 4: {
    if (director.getAccountant() == nullptr && director.getSecretary() == nullptr
&&
        director.getElectricians().empty() && director.getDrivers().empty()) {
        cout << "Некого увольнять.\n";
    }
    else {
        Accountant* accountant = director.getAccountant();
        double driverBaseSalary;
        double guardBaseSalary;
        if (accountant) {
            driverBaseSalary = accountant->getBaseSalary("Водитель");
            guardBaseSalary = accountant->getBaseSalary("Электрик");
        }
        int fireChoice;
        cout << "\nКого вы хотите уволить?\n";
        cout << "1. Электрика\n";
        cout << "2. Водителя\n";
        cout << "3. Бухгалтера\n";
        cout << "4. Секретаря\n";
        cout << "Выберите должность: ";
        getInput(fireChoice);

```



```

        for (size_t i = 0; i < director.getElectricians().size();
++i) {
            auto emp = director.getElectricians()[i]; // Получаем
сотрудника
            Electrician* electrician =
dynamic_cast<Electrician*>(emp); // Приведение типа

            if (electrician) { // Проверяем, что приведение прошло
успешно
                cout << "| " << setw(20) << electrician->getName()
<< " | "
                << setw(20) << electrician->getSurname() << "
|\n";
            }
        }

        cout << "+-----+-----+\n";
    }

    int guardIndex;
    cout << "Выберите электрика для увольнения (номер): ";
    getInput(guardIndex);
    if (guardIndex > 0 && guardIndex <=
director.getElectricians().size()) {

director.fireElectrician(director.getElectricians()[guardIndex - 1]->getName());
        cout << "Электрик уволен.\n";
    }
    else {
        cout << "Некорректный номер.\n";
    }
}

else if (fireChoice == 2) {
    if (director.getDrivers().empty()) {
        cout << "Нет нанятых водителей.\n";
    }
    else {
        if (accountant) {

```

```

cout << "+-----+-----+---
-----+\n";
cout << " |      Имя      |      Фамилия      |
Зарплата      |\n";
cout << "+-----+-----+---
-----+\n";

for (size_t i = 0; i < director.getDrivers().size(); ++i) {
    auto emp = director.getDrivers()[i]; // Получаем
сотрудника

    Driver* driver = dynamic_cast<Driver*>(emp); //
Приведение типа

    if (driver) { // Проверяем, что приведение прошло успешно
        double salary = driver-
>calculateSalary(driverBaseSalary);
        cout << " | " << setw(20) << driver->getName() << " | "
"
        << setw(20) << driver->getSurname() << " | "
        << setw(20) << salary << " |\n";
    }
}

cout << "+-----+-----+---
-----+\n";
}
else {
    cout << "+-----+-----+-----+\n";
    cout << " |      Имя      |      Фамилия      |\n";
    cout << "+-----+-----+-----+\n";

    for (size_t i = 0; i < director.getDrivers().size(); ++i) {
        auto emp = director.getDrivers()[i]; // Получаем
сотрудника

        Driver* driver = dynamic_cast<Driver*>(emp); //
Приведение типа

        if (driver) { // Проверяем, что приведение прошло успешно
            cout << " | " << setw(20) << driver->getName() << " | "
"

```

```

        << setw(20) << driver->getSurname() << " |\n";
    }
}

    cout << "+-----+-----+\n";
}

    int driverIndex;
    cout << "Выберите водителя для увольнения (номер): ";
    getInput(driverIndex);
    if (driverIndex > 0 && driverIndex <=
director.getDrivers().size()) {
        director.fireDriver(director.getDrivers()[driverIndex - 1]-
>getName());

        cout << "Водитель уволен.\n";
    }
    else {
        cout << "Некорректный номер.\n";
    }
}

}

else if (fireChoice == 3) {
    if (director.getAccountant() == nullptr) {
        cout << "Бухгалтер не назначен.\n";
    }
    else {
        director.fireAccountant();
        cout << "Бухгалтер уволен.\n";
    }
}

else if (fireChoice == 4) {
    if (director.getSecretary() == nullptr) {
        cout << "Секретарь не назначен.\n";
    }
    else {
        director.fireSecretary();
        cout << "Секретарь уволен.\n";
    }
}

else {

```

```

        cout << "Некорректный выбор.\n";
    }
}
break;
}

case 5: {
    driverActions(director);
    break;
}
case 6: {
    guardActions(director);
    break;
}
case 7: {
    secretaryActions(director);
    break;
}
case 8: {
    accountantActions(director);
    break;
}
case 9:
    cout << "Возврат в main меню.\n";
    break;
default:
    cout << "Некорректный выбор. Попробуйте снова.\n";
    break;
}
} while (choice != 9);
}

// удаление директора
void removeDirector(vector<Director>& directors, bool& testDataLoaded) {
    if (directors.empty()) {
        cout << "Нет созданных директоров для удаления.\n";
        return;
    }

    cout << "Выберите директора для удаления:\n";

```

```

    for (size_t i = 0; i < directors.size(); ++i) {
        cout << i + 1 << ". " << directors[i].getName() << " " << directors[i].getSurname()
<< "\n";
    }

    int dir_choice;
    getInput(dir_choice);
    system("cls");
    string dirName = directors[dir_choice - 1].getName();
    if (dir_choice > 0 && dir_choice <= directors.size()) {
        // Если удаляемый директор является тестовым, сбрасываем флаг testDataLoaded
        if (directors[dir_choice - 1].getIsTestDirector()) {
            testDataLoaded = false;
        }
        directors.erase(directors.begin() + dir_choice - 1);
        cout << "Директор " << dirName << " удален.\n";
    }
    else {
        cout << "Некорректный выбор.\n";
    }
}

// добавление с файла
void loadDirectorsFromFile(vector<Director>& directors) {
    ifstream inFile("example.txt");
    if (!inFile.is_open()) {
        cout << "Не удалось открыть файл для чтения.\n";
        return;
    }

    string line;
    Director* currentDirector = nullptr;

    while (getline(inFile, line)) {
        istringstream iss(line);
        string position, name, surname, patronymic;
        double rate;

        getline(iss, position, ',');
        getline(iss, name, ',');

```

```

getline(iss, surname, ',');
getline(iss, patronymic, ',');

if (position == "Директор") {
    // Создаем нового директора
    directors.emplace_back(name, surname, patronymic, true);
    currentDirector = &directors.back();
}
else if (currentDirector != nullptr) {
    // Читаем ставку для сотрудников (кроме директора)
    iss >> rate;

    // Добавляем сотрудника к текущему директору в зависимости от роли
    if (position == "Бухгалтер") {
        currentDirector->hireAccountant(new Accountant(name, surname,
patronymic, rate));
    }
    else if (position == "Секретарь") {
        currentDirector->hireSecretary(new Secretary(name, surname, patronymic,
rate));
    }
    else if (position == "Водитель") {
        currentDirector->hireDriver(new Driver(name, surname, patronymic,
rate));
    }
    else if (position == "Электрик") {
        currentDirector->hireElectrician(new Electrician(name, surname,
patronymic, rate, "Без инструмента"));
    }
}

inFile.close();
cout << "Данные по умолчанию загружены из файла example.txt.\n";
}
// main меню программы
void mainMenu() {
    vector<Director> directors;
    int choice;
    bool testDataLoaded = false;

```

```

if (!testDataLoaded) { // Проверка, загружены ли тестовые данные
    loadDirectorsFromFile(directors);
    testDataLoaded = true;
}
else {
    cout << "Данные по умолчанию уже были загружены.\n";
}

do {
    cout << "\nMain меню:\n";
    cout << "1. Добавить нового директора\n";
    cout << "2. Выбрать директора для управления компанией\n";
    cout << "3. Уволить директора\n";
    cout << "4. Выход\n";
    cout << "Выберите действие: ";
    getInput(choice);
    system("cls");

    switch (choice) {
    case 1: {
        string name, surname, patronymic;
        cout << "Введите имя директора: ";
        getAlphaString(name);
        cout << "Введите фамилию директора: ";
        getAlphaString(surname);
        cout << "Введите отчество директора: ";
        getAlphaString(patronymic);

        directors.push_back(Director(name, surname, patronymic));
        break;
    }
    case 2: {
        if (directors.empty()) {
            cout << "Нет созданных директоров.\n";
        }
        else {
            cout << "Выберите директора:\n";
            for (size_t i = 0; i < directors.size(); ++i) {

```



```

        cout << i + 1 << ". " << directors[i].getName() << " " <<
directors[i].getSurname() << "\n";
    }
    int dir_choice;
    getInput(dir_choice);
    if (dir_choice > 0 && dir_choice <= directors.size()) {
        directorMenu(directors[dir_choice - 1]);
    }
    else {
        cout << "Некорректный выбор.\n";
    }
}
break;
}
case 3:
    removeDirector(directors, testDataLoaded);
    break;
case 4:
    cout << "Выход из программы.\n";
    break;
default:
    cout << "Некорректный выбор. Попробуйте снова.\n";
    break;
}
} while (choice != 4);
}

int main() {
    setlocale(LC_ALL, "RU");
    system("chcp 1251");
    system("cls");

    mainMenu();
    return 0;
}

```