

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:** Неделько Никита Дмитриевич  
(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 КР
6. Подготовка к защите КР

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

**Руководитель** 02.10.2024 М. В. Раскатова  
(дата) (Фамилия и инициалы)

**Студент** 04.10.2024 Н.Д. Неделько  
(дата) (Фамилия и инициалы)

<b>ЗАДАНИЕ .....</b>	<b>3</b>
<b>ГЛАВА 1. Постановка задачи .....</b>	<b>4</b>
<b>ГЛАВА 2. Разработка программы .....</b>	<b>8</b>
<b>2.1 Диаграммы UML .....</b>	<b>8</b>
<b>2.2 Описание классов .....</b>	<b>26</b>
<b>2.2.1 Базовый класс Person .....</b>	<b>27</b>
<b>2.2.2 Класс SalaryDatabase.....</b>	<b>28</b>
<b>2.2.3 Класс Director .....</b>	<b>29</b>
<b>2.2.4 Класс Accountant.....</b>	<b>31</b>
<b>2.2.5 Класс Secretary .....</b>	<b>33</b>
<b>2.2.6 Класс SecurityGuard .....</b>	<b>35</b>
<b>2.2.7 Класс Driver .....</b>	<b>37</b>
<b>2.2.8 Класс Company .....</b>	<b>38</b>
<b>2.2.9 Класс Menu.....</b>	<b>39</b>
<b>2.3 Описание пользовательского интерфейса .....</b>	<b>42</b>
<b>ГЛАВА 3. Реализация и тестирование программы .....</b>	<b>50</b>
<b>3.1 Листинг программы .....</b>	<b>50</b>
<b>3.2 Тестирование программы.....</b>	<b>51</b>
<b>ГЛАВА 4. Вывод .....</b>	<b>66</b>
<b>ПРИЛОЖЕНИЯ .....</b>	<b>67</b>
<b>Приложение 1 .....</b>	<b>67</b>
<b>Приложение 2 .....</b>	<b>69</b>
<b>Приложение 3 .....</b>	<b>70</b>
<b>Приложение 4 .....</b>	<b>76</b>
<b>Приложение 5 .....</b>	<b>82</b>
<b>Приложение 6 .....</b>	<b>87</b>
<b>Приложение 7 .....</b>	<b>91</b>
<b>Приложение 8 .....</b>	<b>96</b>

## **ЗАДАНИЕ**

Общее задание представлено в Приложение 1.

Иерархия наследования будет включать в себя следующие классы – Человек, Директор, Бухгалтер, Секретарь, Охранник, Водитель.

## ГЛАВА 1. Постановка задачи

Цель работы — разработать программу на основе объектно-ориентированного подхода, реализующую классы для управления и учета различных типов пользователей. Программа должна быть построена на принципах ООП и включать функционал для взаимодействия с данными, обработки событий и выполнения задач, характерных для каждого пользователя.

Задача состоит в следующем:

1. Разработать структуру классов, включающую роли: Директор, Секретарь, Бухгалтер, Водитель, Охранник и базовый класс Человек, от которого будут унаследованы общие свойства и методы.

2. Для каждого класса необходимо определить:

- свойства, отражающие характеристики объектов (например, имя, должность, заработная плата, спецсредства);
- методы, реализующие поведение объектов в соответствии с их функционалом.

3. Разработать пользовательский интерфейс, позволяющий выполнять основные функции программы:

- добавление, изменение и удаление данных;
- выполнение задач, связанных с функционалом каждой роли;
- взаимодействие с системой в удобной форме через консольный интерфейс.

4. Создать UML-диаграммы для визуального представления:

- Диаграмму классов для отображения иерархии объектов и их взаимосвязей;
- Диаграмму вариантов использования, чтобы продемонстрировать функции предметной области;
- Диаграммы последовательности и кооперативные для описания взаимодействия объектов.

5. Реализовать программу на языке C++ с использованием ключевых принципов объектно-ориентированного программирования (инкапсуляция, наследование, полиморфизм).

6. Провести тестирование программы, проверив корректность работы основных функций для каждого класса и сценариев взаимодействия.

7. Составить отчет с результатами тестирования, выявленными проблемами (если есть) и их решениями.

Итоговая реализация должна быть компактной, удобной для пользователя и соответствовать поставленным требованиям.

Описание предметной области:

Разрабатывается информационная система для управления тюрьмой, которая будет автоматизировать процесс учета заключенных, их перемещения, а также управление персоналом и различными административными задачами.

При поступлении новых заключенных директор издает приказ об осуществлении перевозки с указанием всех необходимой информации (кто перевозит, откуда, куда и сколько). Затем водитель, получив задание, отправляется выполнять перевозку. В системе фиксируются все данные о перевозке: дату, маршрут и ответственного сотрудника. Это позволяет синхронизировать информацию с другими подразделениями, например, с охраной, которая будет контролировать процесс прибытия заключенных в учреждение.

Охранники обеспечивают порядок и безопасность на территории учреждения. В системе они регистрируют свои смены и управляют статусами объектов проверки, фиксируя выполненные проверки помещений и территорий. В системе охранники могут менять данные о спецсредствах, которые они используют, фиксируя переход с одного средства на другое в зависимости от ситуации.

После поступления новых заключенных секретарь регистрирует их в системе. Это включает в себя внесение личных данных, таких как фамилия, имя, дата рождения, идентификационные данные, а также информация о

совершенных преступлениях, сроках наказания и условиях содержания. Все эти данные используются для назначения задач персоналу, а также для учета и планирования мероприятий в рамках тюремного режима.

Секретарь выполняет важную роль в организации административных процессов. Он планирует встречи и мероприятия (ужины, походы в душ, прогулки и собрания сотрудников). В системе секретарь создает события, указывая дату, время, место и список заключенных для участия. Также он может выводить списки сотрудников, включая данные об охранниках и водителях, а также информацию о работниках, подчиненных конкретному директору.

Исполнительный директор отвечает за стратегическое управление учреждением, утверждает бюджет, который напрямую влияет на количество сотрудников и их зарплаты. В условиях ограниченного бюджета директор может ограничить найм новых работников. Он также решает вопросы найма и увольнения персонала, рассматривает кандидатуры, утверждает назначения и добавление сотрудников в систему. После утверждения новый сотрудник добавляется в систему, и его зарплата списывается с бюджета отдела. Увольнение сотрудника также проходит через согласование с директором, который принимает решение об освобождении от должности и возвращении средств в бюджет.

Бухгалтер занимается расчетом заработной платы сотрудников, учитывая их ставки, премии и может изменять коэффициент заработной платы определенной должности. Он также ведет учет всех финансовых операций, связанных с содержанием учреждения, и составляет отчеты для исполнительного директора.

Функции программы:

Разрабатываемая информационная система обеспечивает выполнение следующих функций, реализующих основные задачи программы:

Функции базового класса "Человек":

- хранение и предоставление основных данных о человеке (имя, фамилия, дата рождения);

- управление общими действиями, такими как вывод информации о пользователе и обновление его данных.

Функции класса "Директор":

- утверждение бюджета подразделения, что влияет на количество сотрудников и их зарплаты;
- найм новых сотрудников с учетом доступного бюджета;
- увольнение сотрудников и перерасчет бюджета после увольнения;
- вывод информации обо всех сотрудниках.

Функции класса "Секретарь":

- планирование и создание мероприятий;
- ввод информации о назначенных встречах и их участниках;
- управление списками сотрудников, включая фильтрацию по категориям (например, вывод всех охранников или водителей).

Функции класса "Бухгалтер":

- расчет заработной платы сотрудников, включая ставки и премии;
- изменение коэффициента зарплаты для определенной должности;
- вывод списка сотрудников с их текущей ставкой и заработной платой.

Функции класса "Охранник":

- проверка объектов с возможностью установки статусов проверки;
- управление личным спецсредством.

Функции класса "Водитель":

- создание заданий на перевозку заключенных;
- управление водительскими правами (добавление или удаление);
- управление транспортным средством (изменение с учетом прав).

Пользовательский интерфейс:

- добавление, редактирование и удаление данных о сотрудниках, задачах;
- удобная навигация по разделам системы;
- взаимодействие с системой через консольный интерфейс, поддерживающий основные команды и сценарии работы.

## **ГЛАВА 2. Разработка программы**

Разработка программы включает создание ее архитектуры, описание функционала и технической реализации. В данной главе представлены ключевые аспекты проектирования, такие как UML-диаграммы, описание классов и пользовательского интерфейса, которые позволят понять структуру и функционал программы.

### **2.1 Диаграммы UML**

Для проектирования программы были разработаны следующие UML-диаграммы:

- диаграмма вариантов использования: описывает основные функции программы с точки зрения предметной области;
- диаграмма последовательности: демонстрирует порядок выполнения операций между объектами;
- кооперативные диаграммы: отображают взаимодействие объектов для выполнения конкретных задач;
- диаграммы деятельности: представление рабочих процессов поэтапных действий и действий с поддержкой выбора, итерации и параллелизма.

Диаграмма вариантов использования помогает понять, кто имеет доступ к каким функциям, ключевые роли и взаимодействие пользователей с программой (Рис. 2.1).



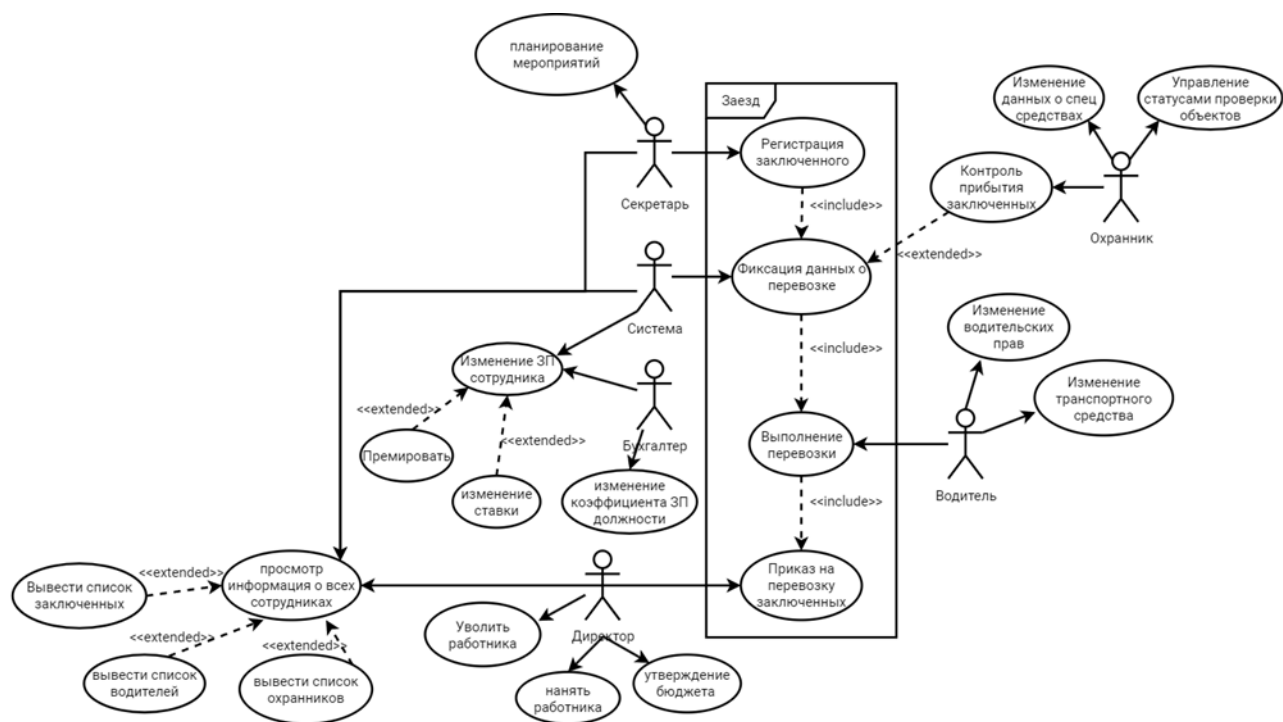


Рис. 2.1. Диаграмма вариантов использования

Диаграммы деятельности для некоторых вариантов использования представлены ниже (Рис. 2.2 – Рис. 2.6).

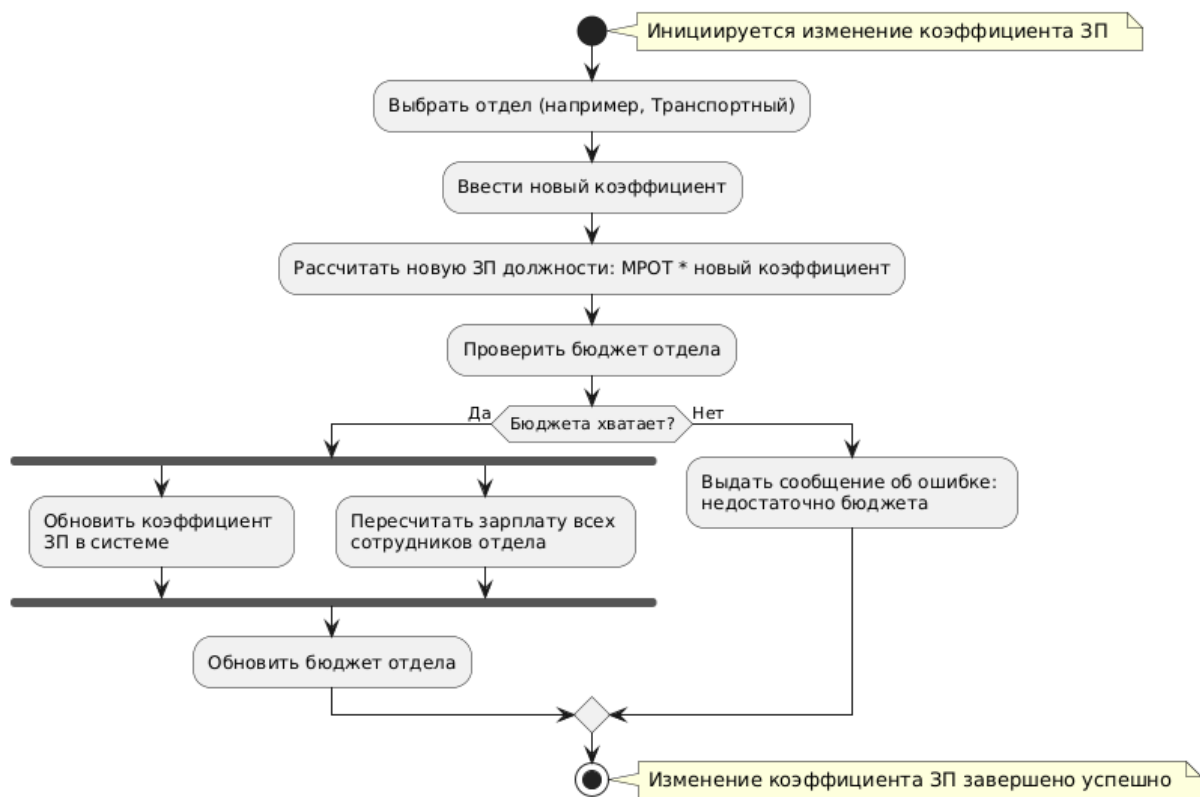


Рис. 2.2. Диаграмма деятельности для варианта использования "Изменение коэффициента ЗП"

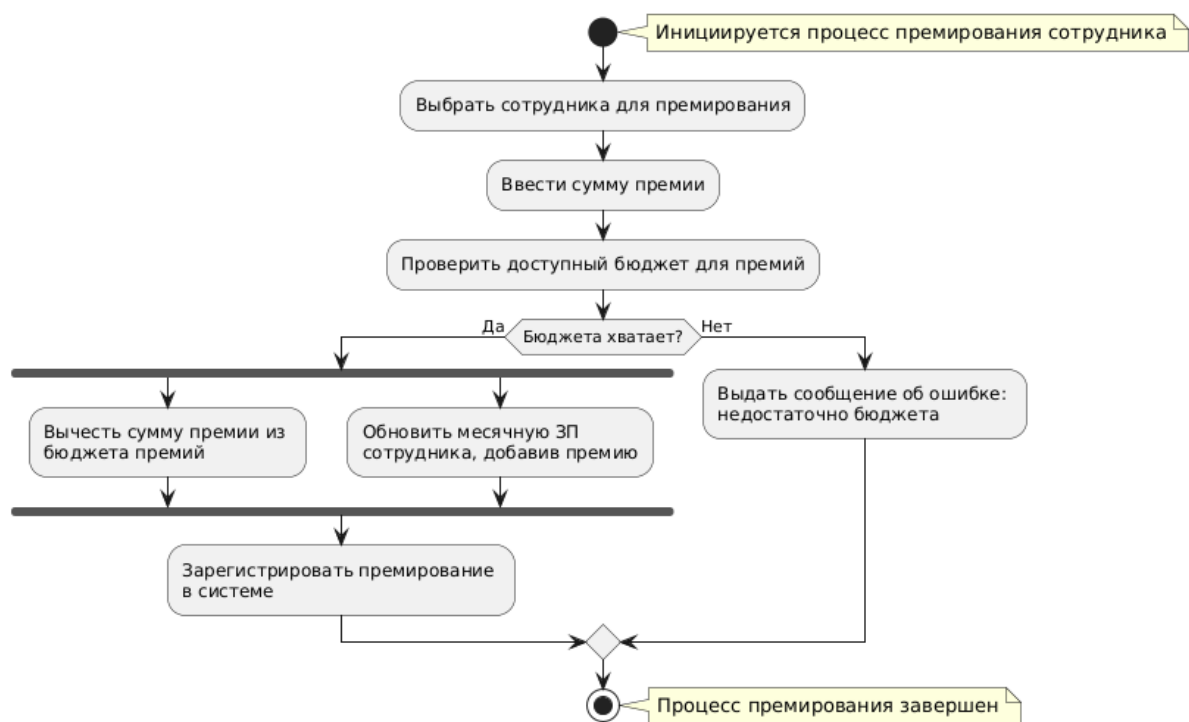


Рис. 2.3. Диаграмма деятельности для варианта использования "Премировать сотрудника"

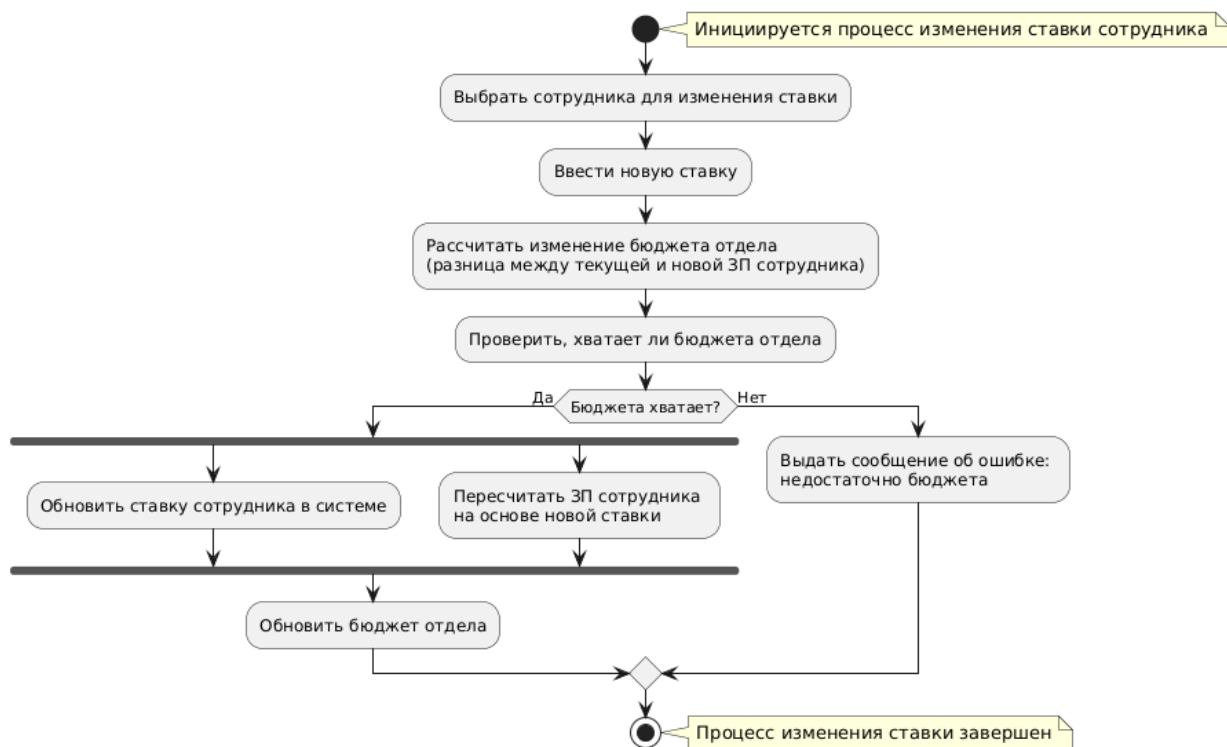


Рис. 2.4. Диаграмма деятельности для варианта использования "Изменение ставки сотрудника"

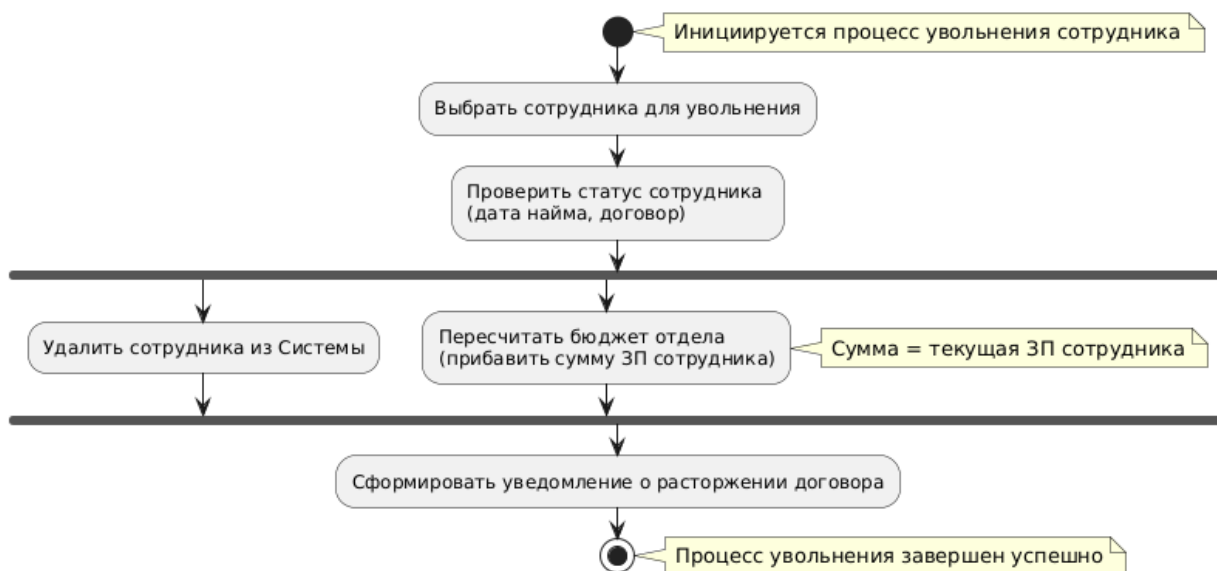


Рис. 2.5. Диаграмма деятельности для варианта использования "Увольнение рабочего"

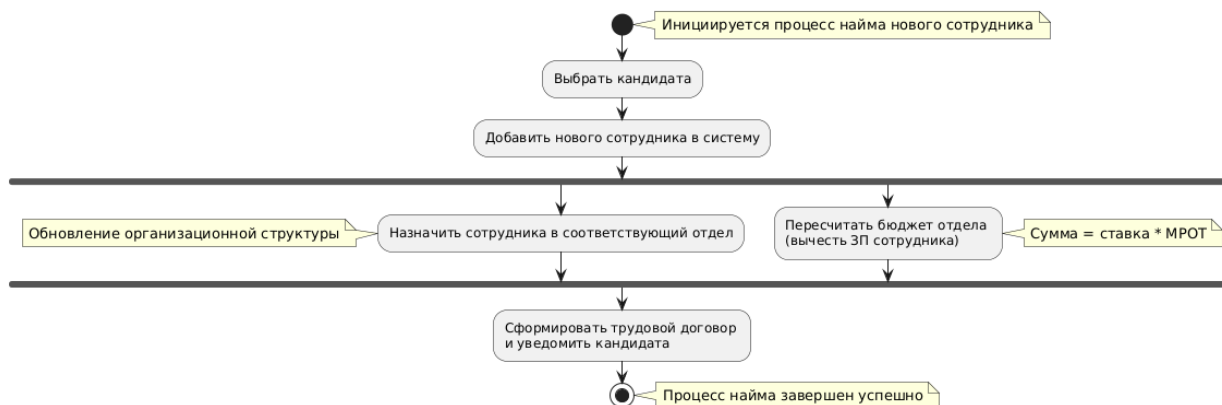


Рис. 2.6. Диаграмма деятельности для варианта использования "Нанять сотрудника"

Диаграммы деятельности описывают поток управления целевой системой, такой как исследование сложных бизнес-правил и операций, а также описание прецедентов и бизнес-процессов.

Далее будут приведены диаграммы последовательности и кооперативные диаграммы UML для каждого варианта использования (Рис. 2.7 - Рис. 2.26).

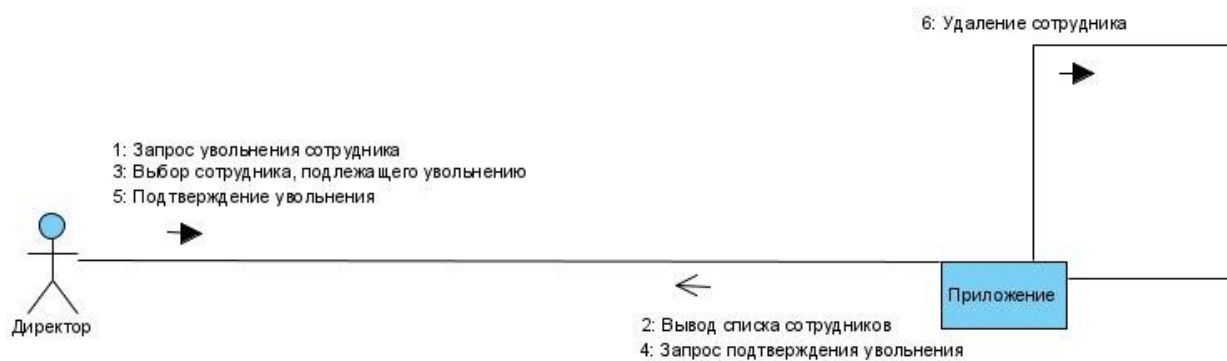


Рис. 2.7. Кооперативная диаграмма для варианта использования "Уволить работника"

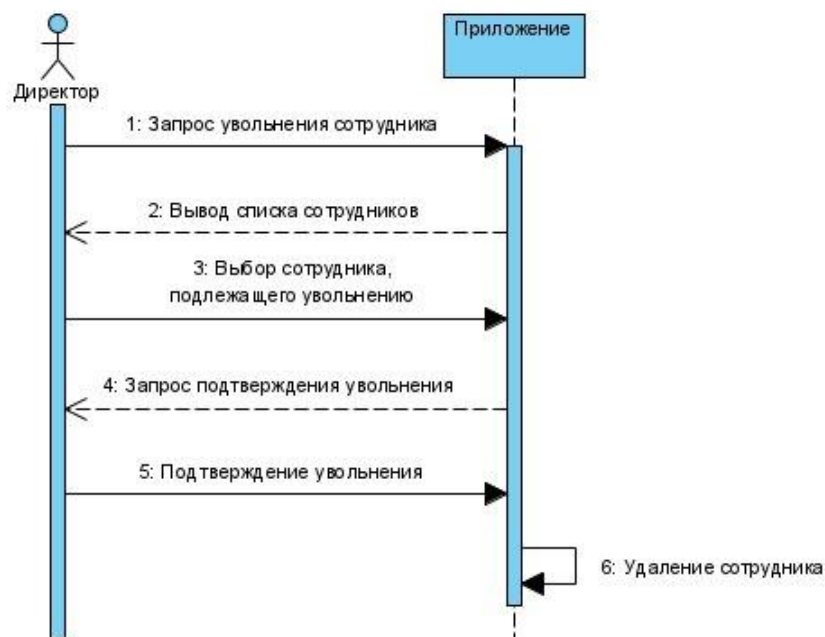


Рис. 2.8 Диаграмма последовательности для варианта использования "Уволить работника"



Рис. 2.9. Диаграмма последовательности для варианта использования "Изменение спец средств"

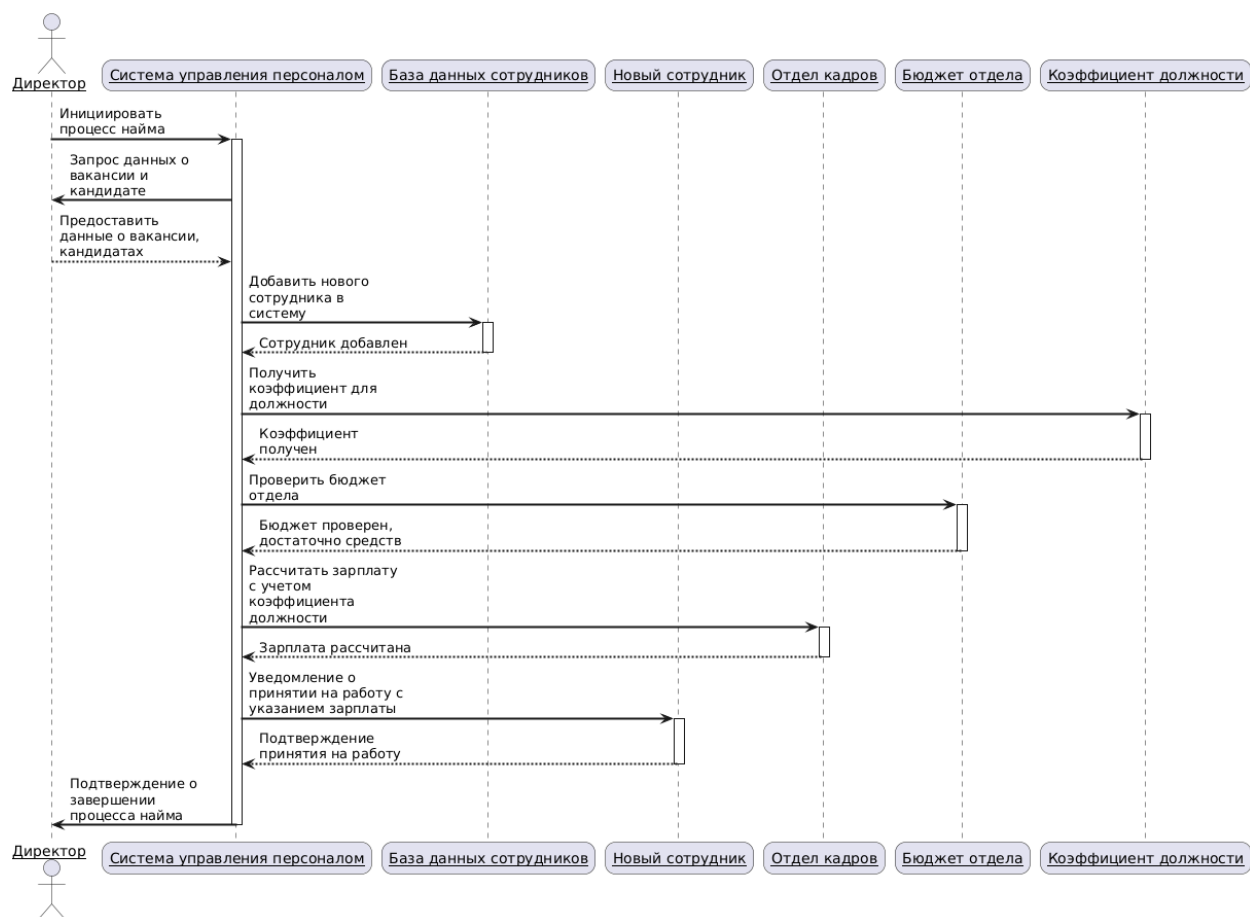


Рис. 2.10. Диаграмма последовательности для варианта использования "Нанять сотрудника"

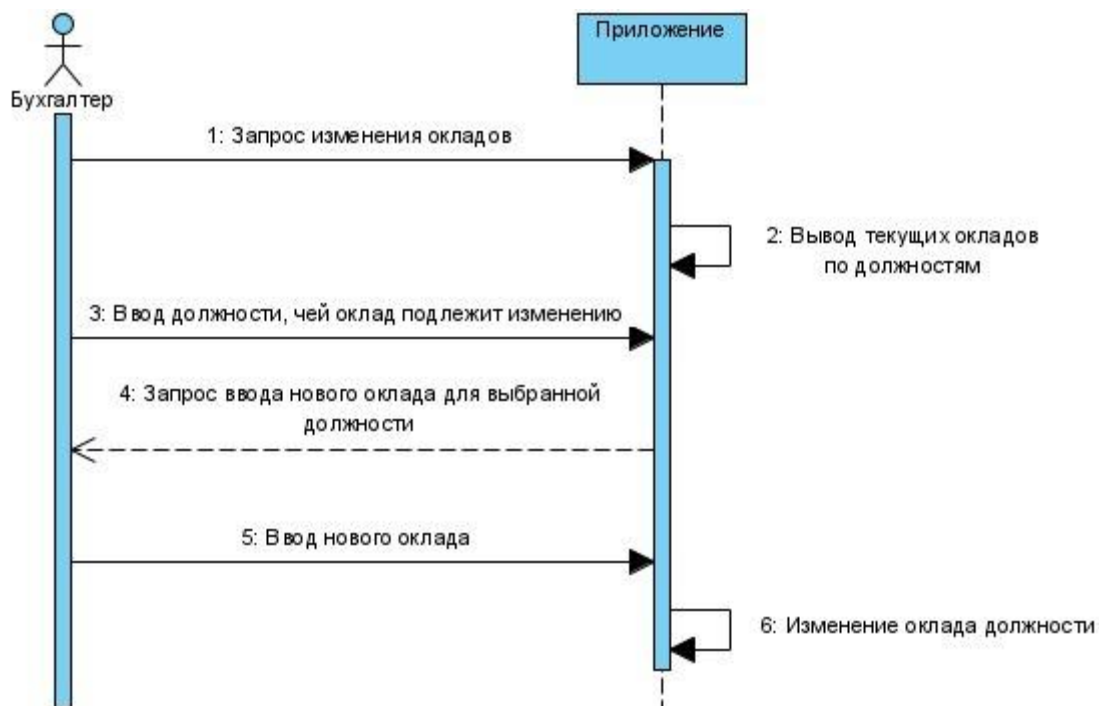


Рис. 2.11. Диаграмма последовательности для варианта использования "Изменить ставку"

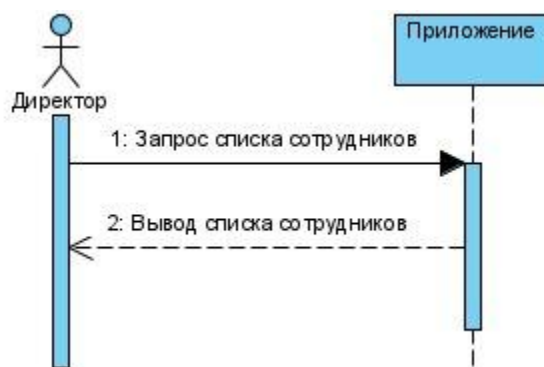


Рис. 2.12. Диаграмма последовательности для варианта использования "Вывести информацию о всех сотрудниках"

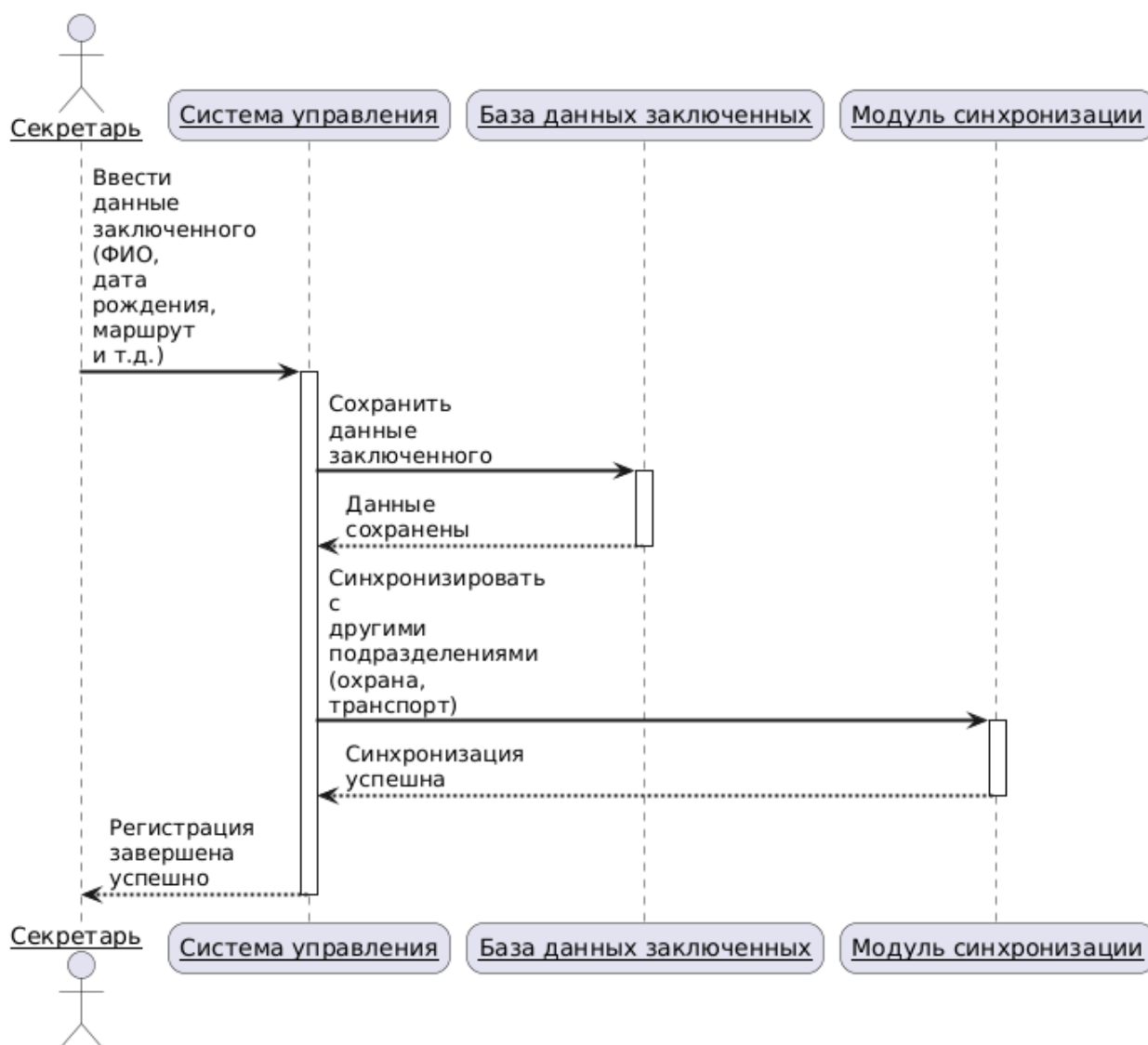


Рис. 2.13. Диаграмма последовательности для варианта использования "Регистрация заключенного"

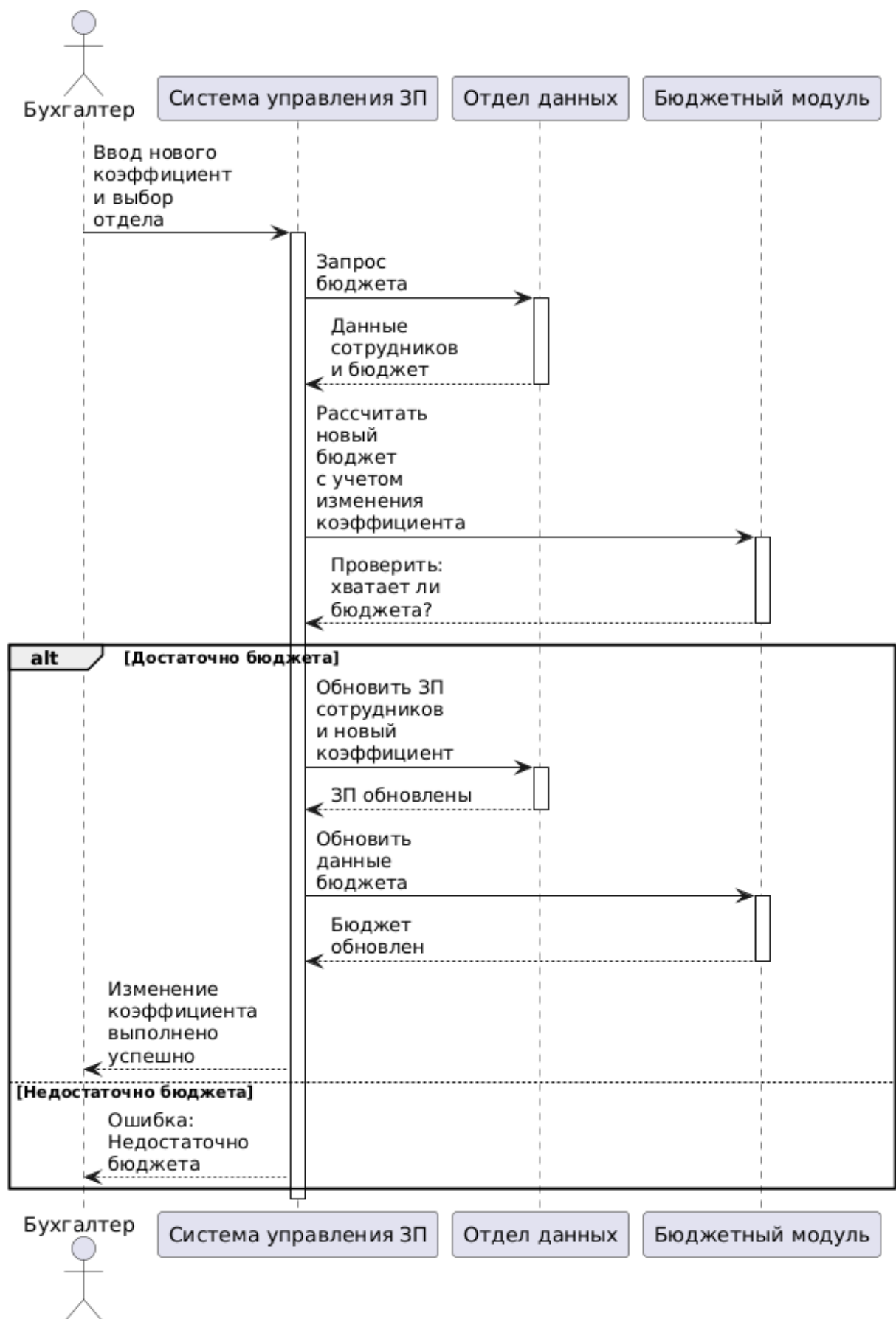


Рис. 2.14. Диаграмма последовательности для варианта использования "Изменение коэффициента ЗП должности"

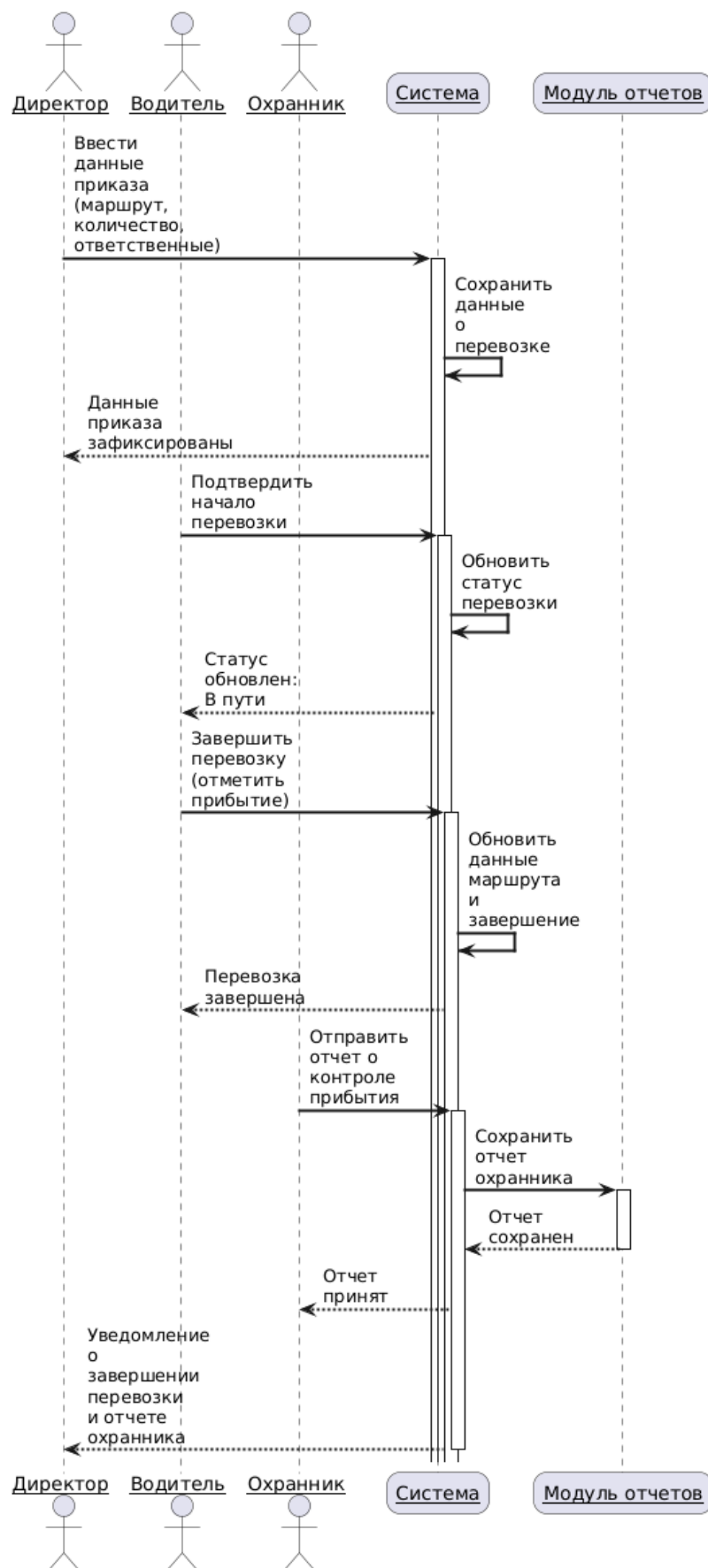


Рис. 2.15. Диаграмма последовательности для раздела заезд "Перевоз заключенных"



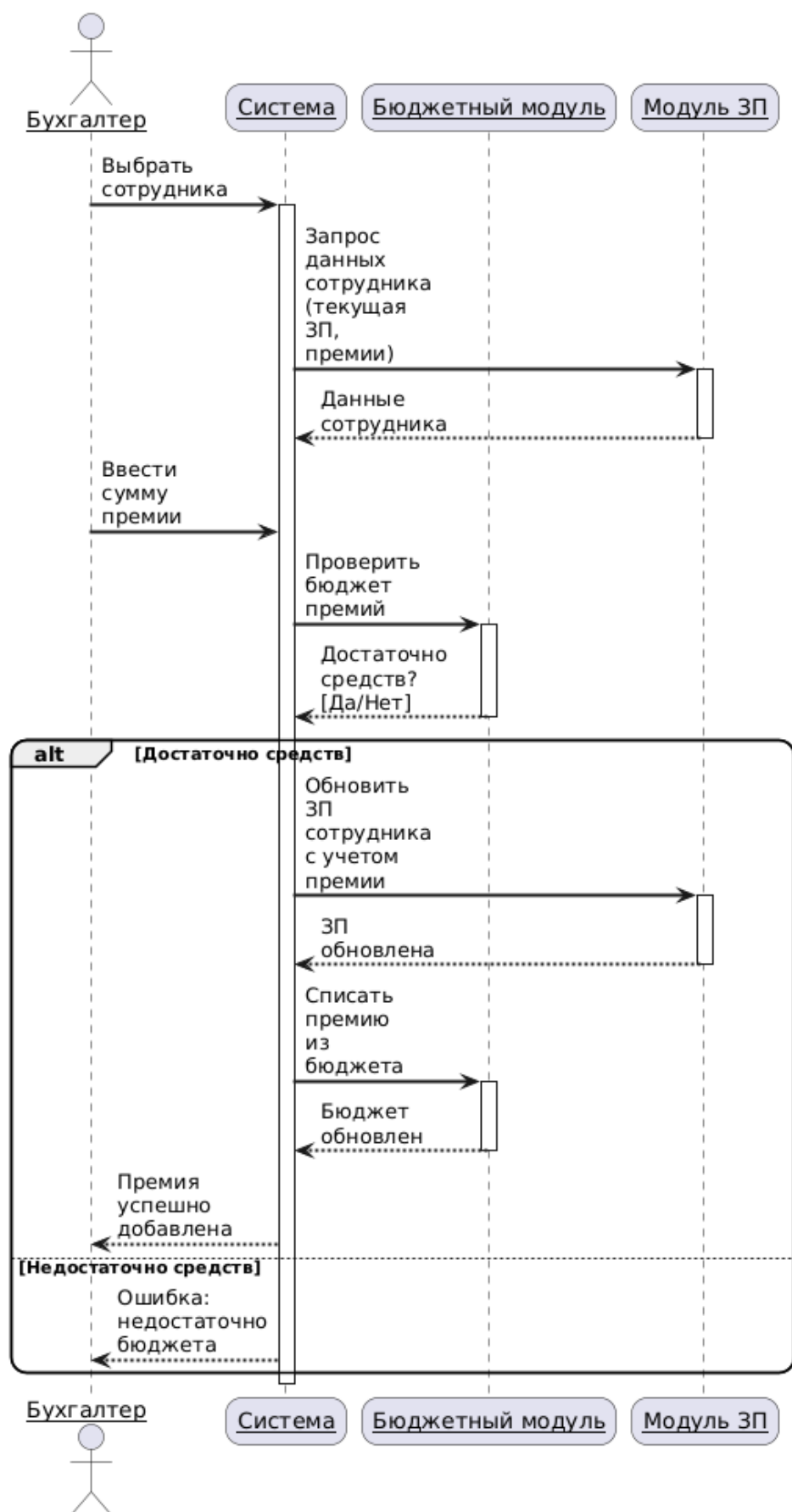


Рис. 2.16. Диаграмма последовательности для варианта использования "Премировать"

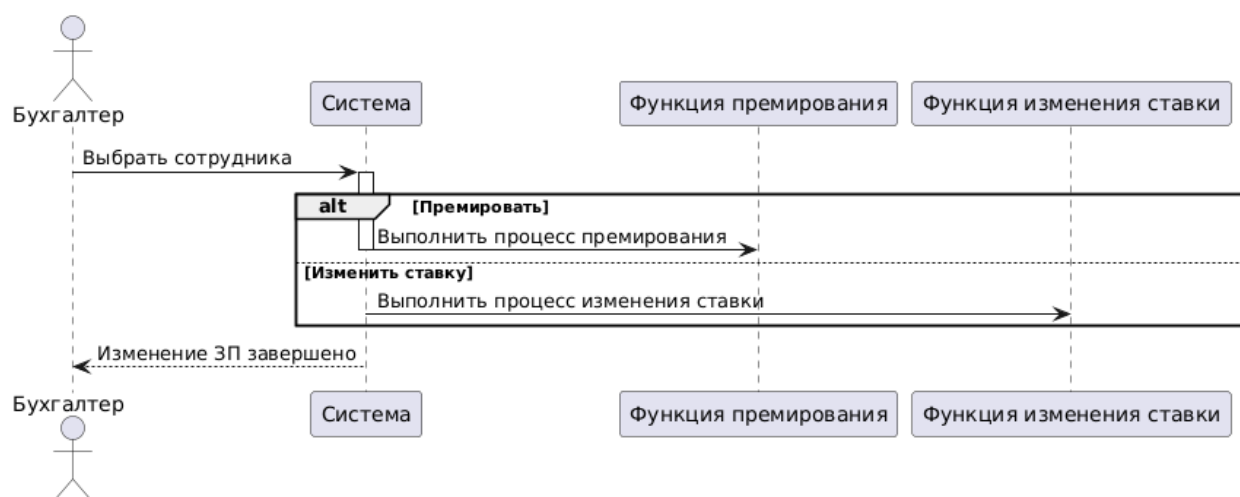


Рис. 2.17. Диаграмма последовательности для варианта использования "Изменение ЗП сотрудника"

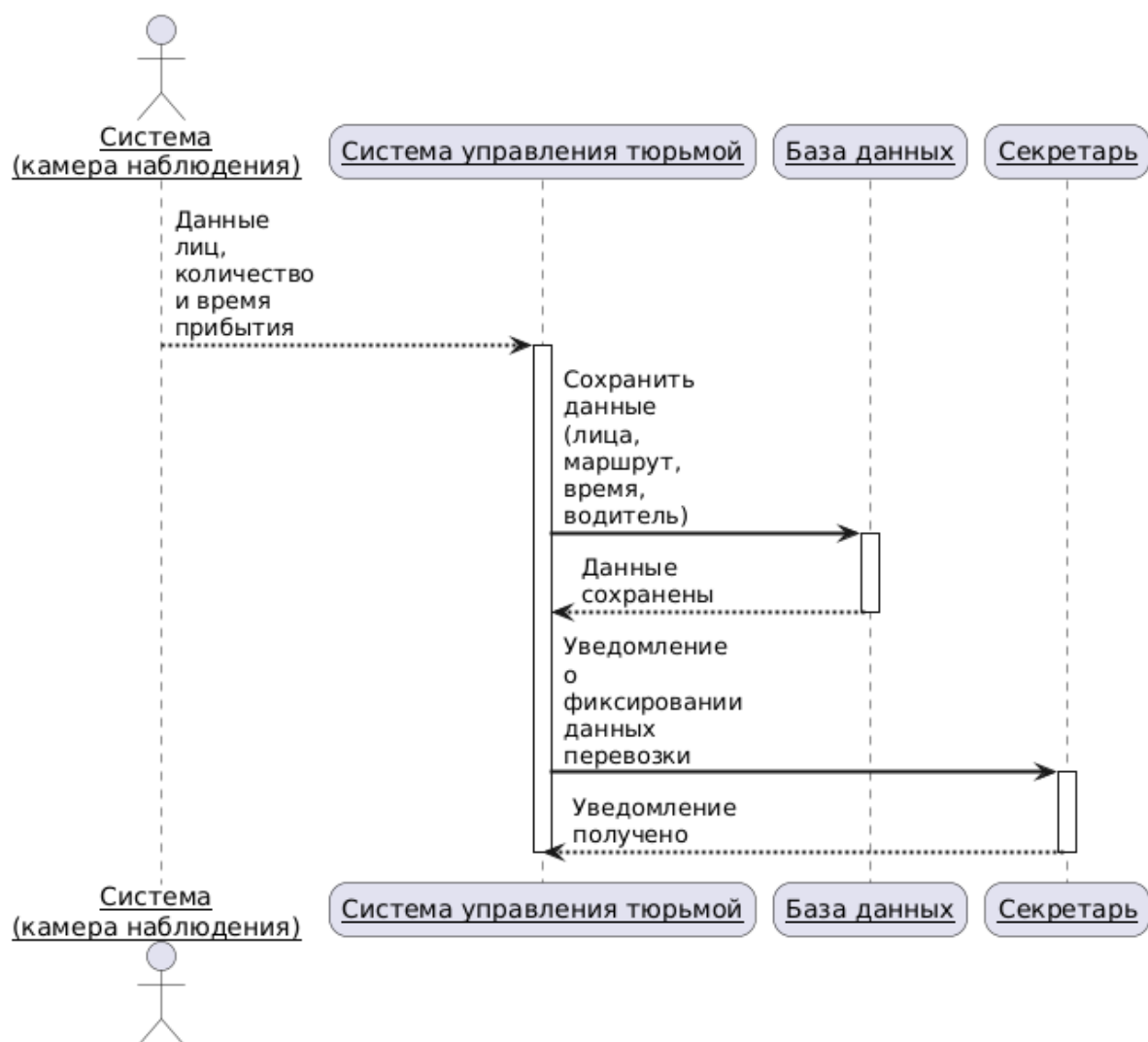


Рис. 2.18. Диаграмма последовательности для варианта использования "Фиксация данных о перевозке"

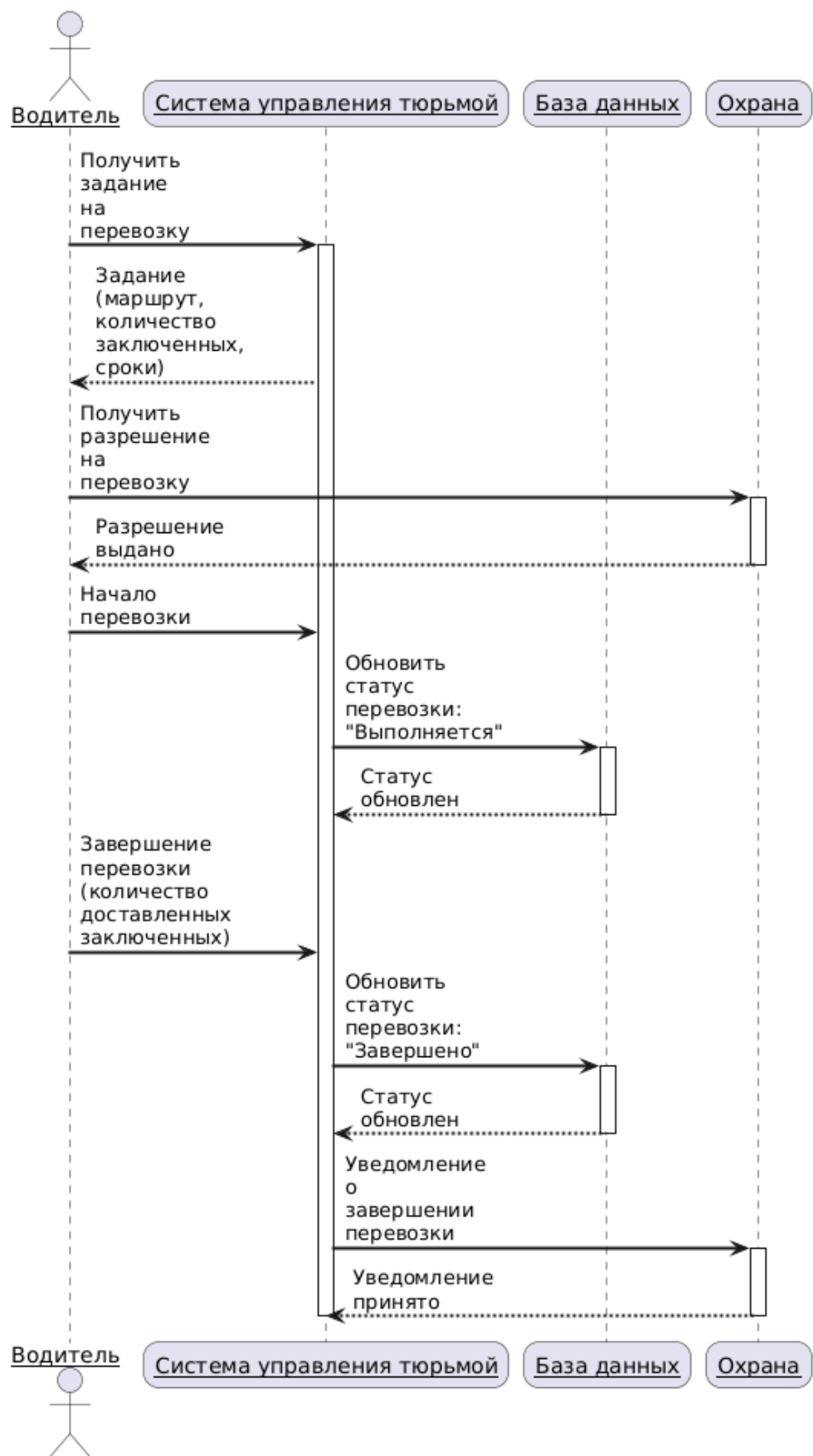


Рис. 2.19. Диаграмма последовательности для варианта использования "Выполнение перевозки"



Рис. 2.20. Диаграмма последовательности для варианта использования "Изменение водительских прав"

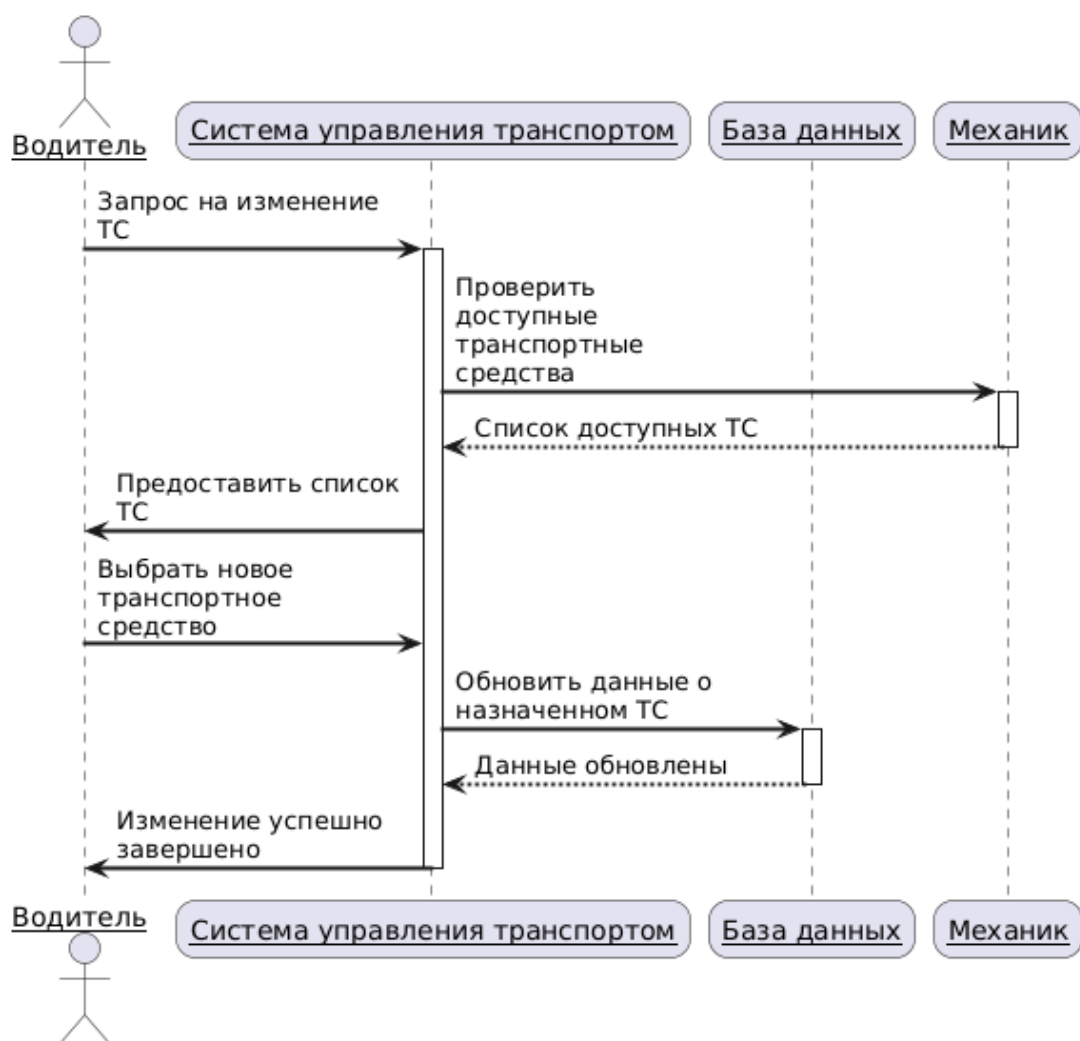


Рис. 2.21. Диаграмма последовательности для варианта использования "Изменение транспортного средства"

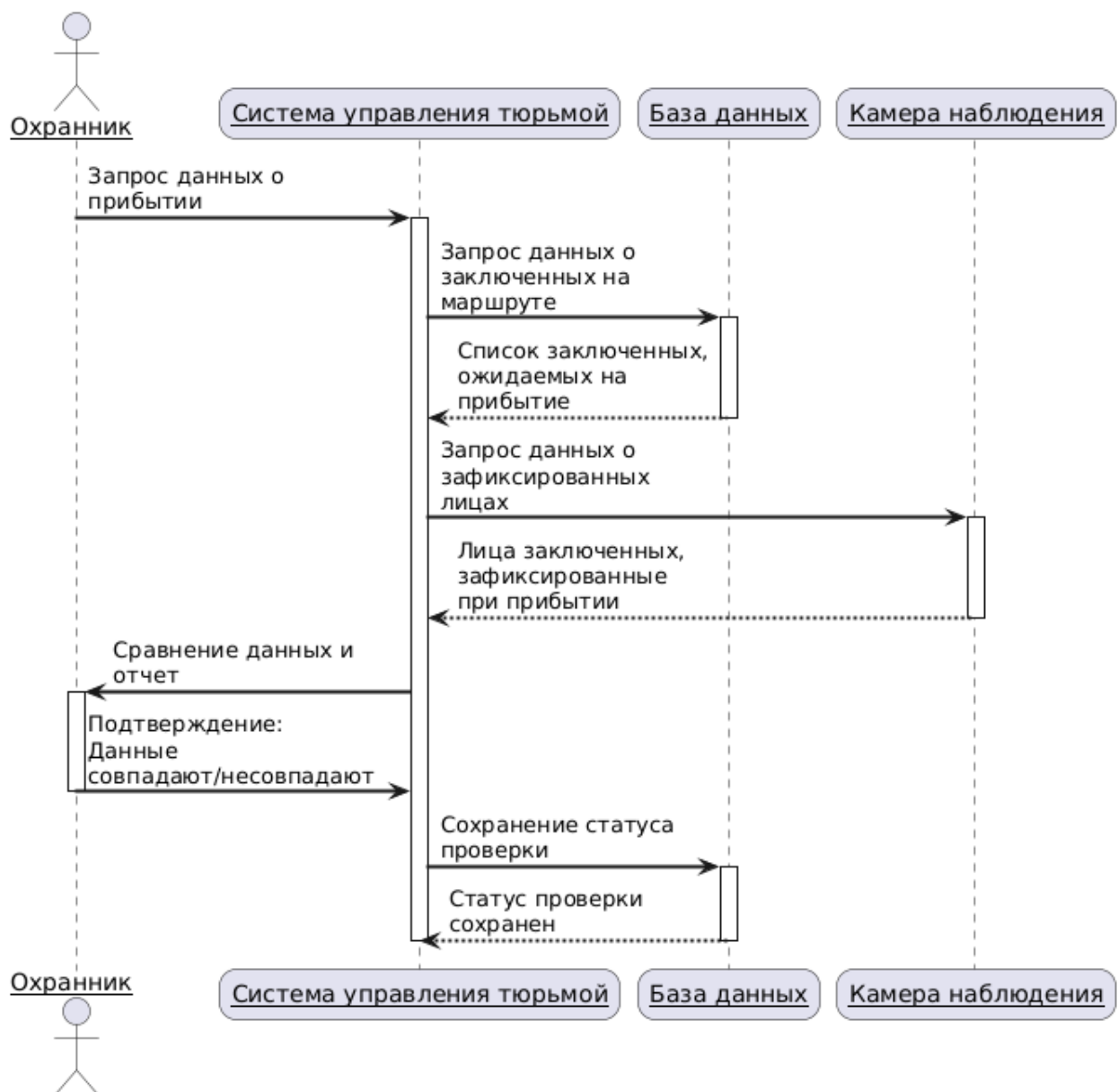


Рис. 2.22. Диаграмма последовательности для варианта использования "Контроль прибытия заключенных"

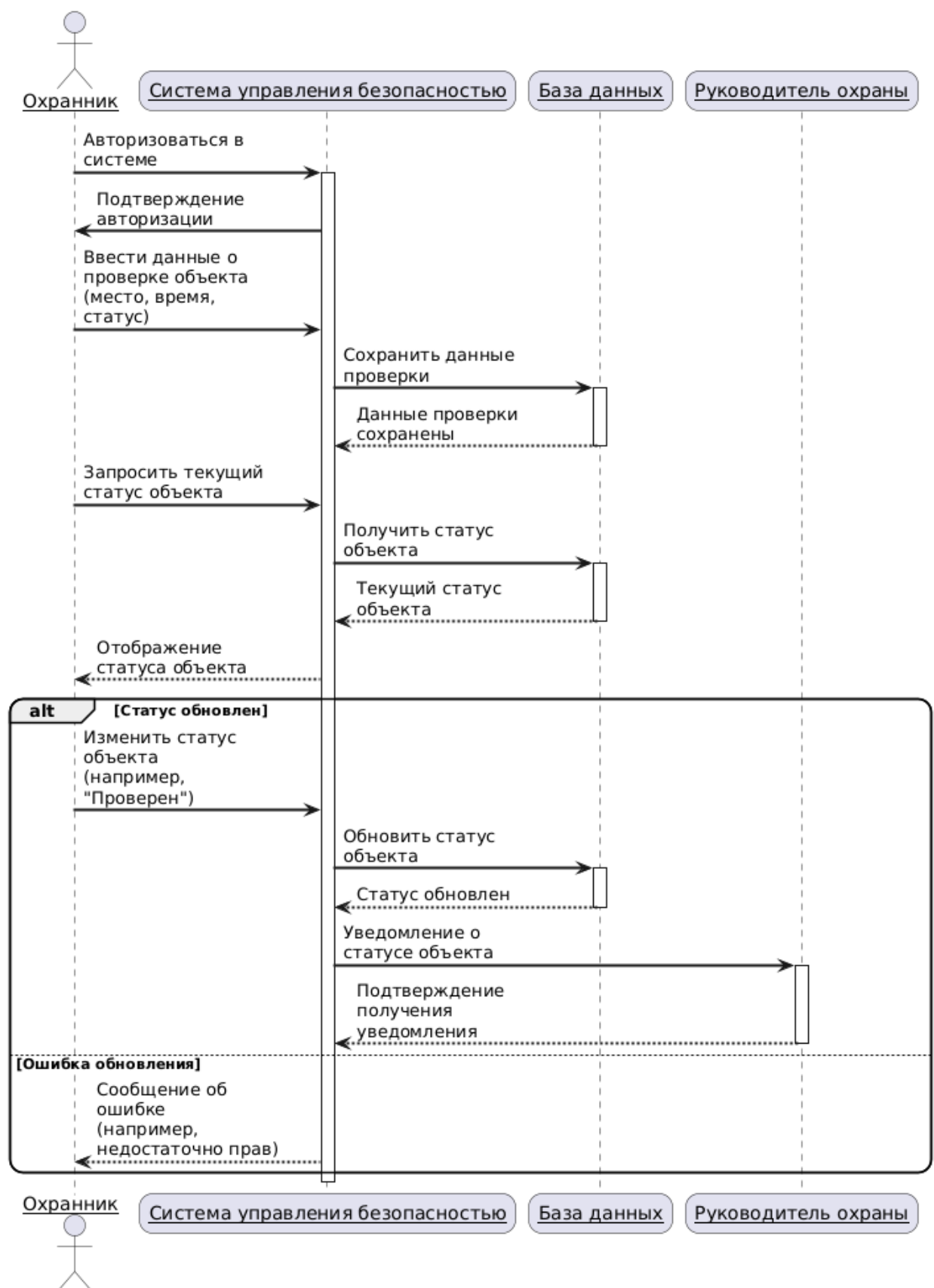


Рис. 2.23. Диаграмма последовательности для варианта использования "Изменение статуса проверки объектов"

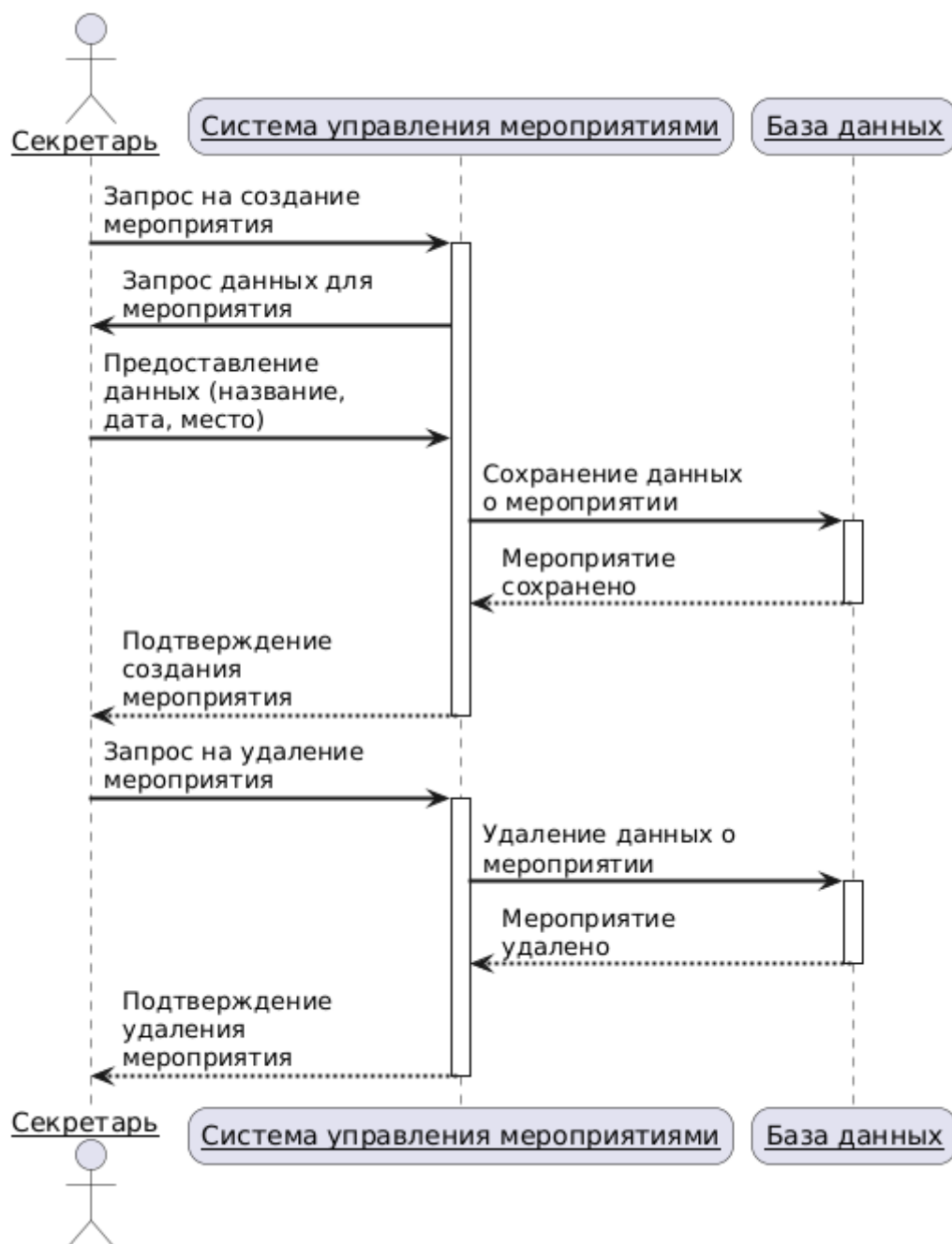


Рис. 2.24. Диаграмма последовательности для варианта использования "Создание мероприятий"



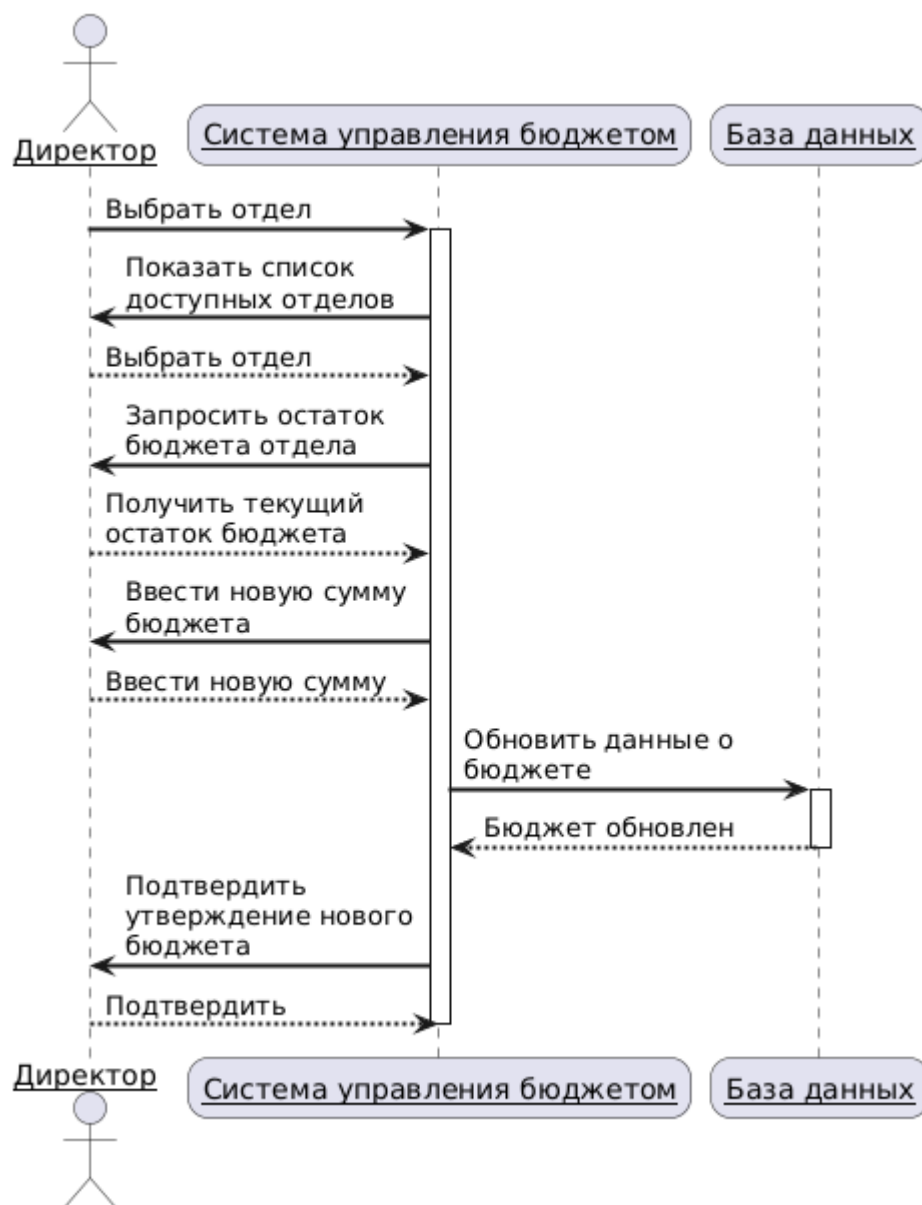


Рис. 2.25. Диаграмма последовательности для варианта использования "Утверждение бюджета отдела"

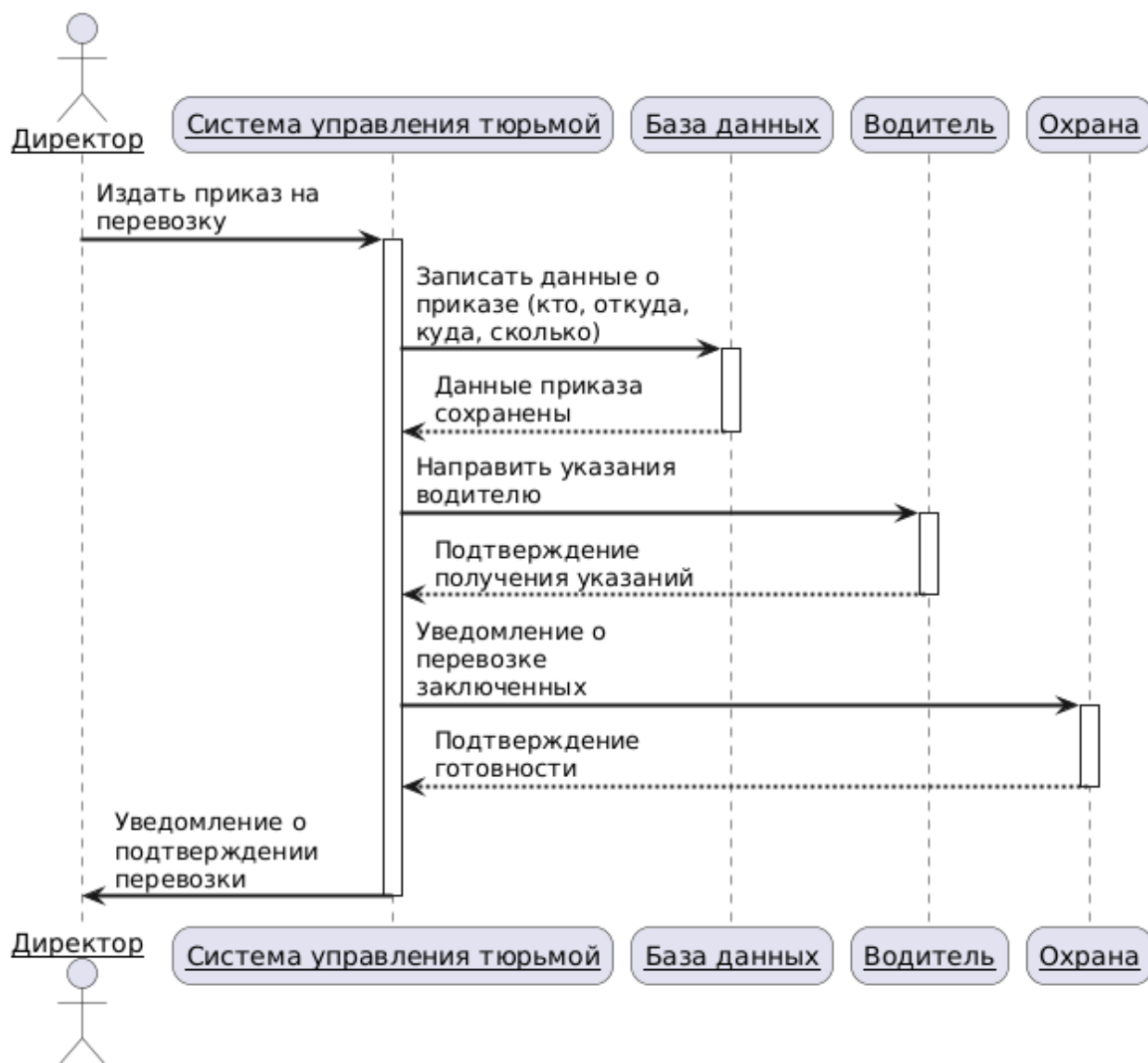


Рис. 2.26. Диаграмма последовательности для варианта использования "Издать приказ на перевозку"

## 2.2 Описание классов

В программе реализованы классы, каждый из которых отвечает за конкретный функционал. Все классы наследуются от базового класса Человек.

Ниже на диаграмме классов представлены все классы и их функции, реализованные в программе (Рис. 2.27).

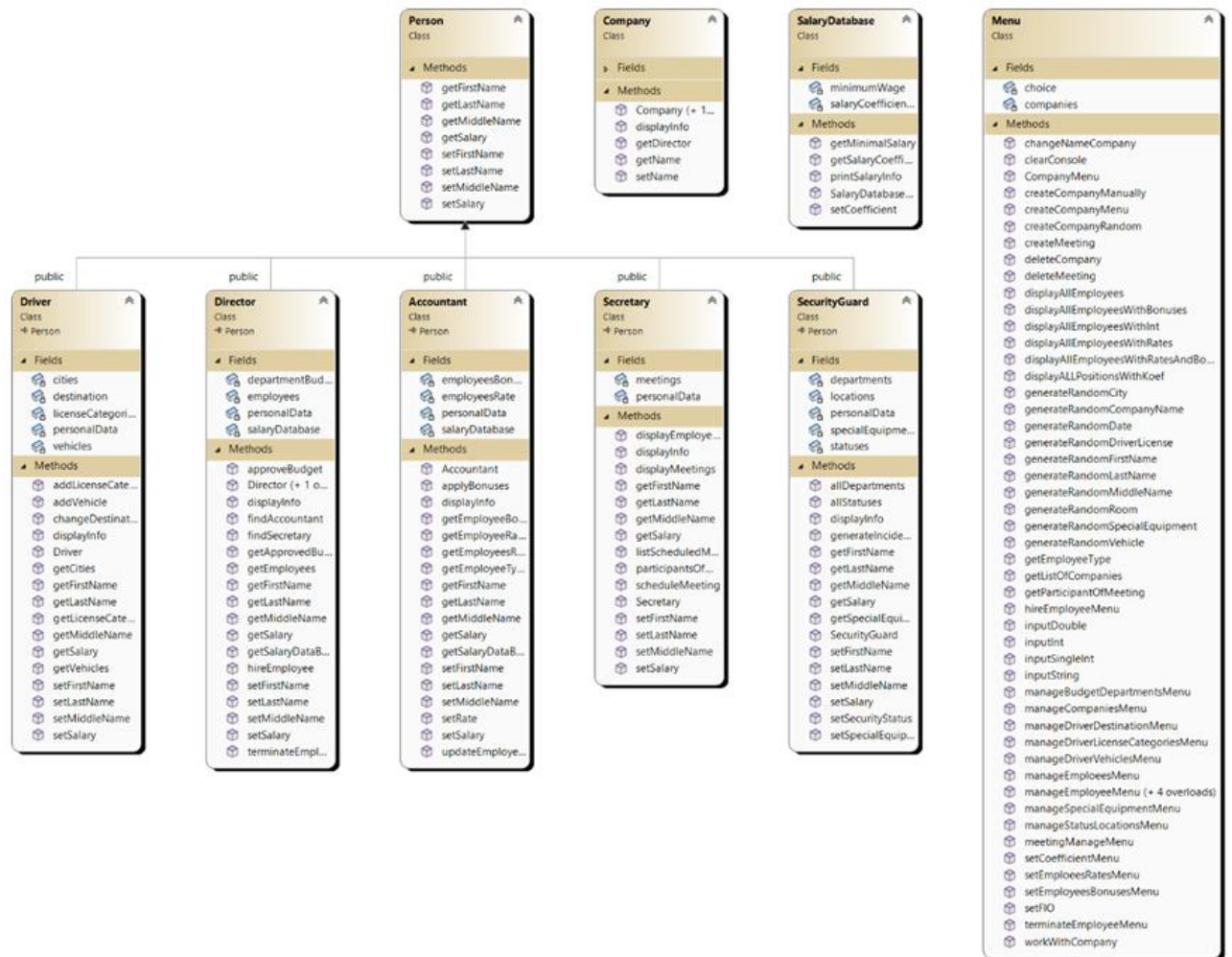


Рис. 2.27. Диаграмма классов

## 2.2.1 Базовый класс Person

Назначение: класс Person является абстрактным базовым классом или интерфейсом, от которого наследуются все остальные классы. Он содержит основные методы, общие для всех типов сотрудников.

Поля: в классе Person нет конкретных полей, так как это абстрактный класс, который определяет интерфейс для работы с общими характеристиками. Поля (например, имя, фамилия, зарплата) должны быть реализованы в классах-наследниках.

Методы. Класс содержит следующие чисто виртуальные методы, которые должны быть переопределены в классах-наследниках:

- `void setFirstName(const string &firstName)` — устанавливает имя;
- `string getFirstName() const` — возвращает имя;
- `void setLastName(const string &lastName)` — устанавливает фамилию;
- `string getLastName() const` — возвращает фамилию;
- `void setMiddleName(const string &middleName)` — устанавливает отчество;
- `string getMiddleName() const` — возвращает отчество;
- `void setSalary(const double salary)` — устанавливает зарплату;
- `double getSalary() const` — возвращает зарплату.

Конструкторы и деструкторы: класс `Person` не имеет явного конструктора, так как это абстрактный класс. Тем не менее, у класса есть виртуальный деструктор, чтобы гарантировать корректное удаление объектов наследников:

```
virtual ~Person() = default;
```

### 2.2.2 Класс `SalaryDatabase`

Назначение: класс `SalaryDatabase` предназначен для управления минимальной заработной платой и коэффициентами, используемыми для расчета зарплат сотрудников на различных должностях.

Поля:

- `map<Position, double> salaryCoefficients` — ассоциативный контейнер, хранящий коэффициенты для каждой должности;
- `double minimumWage` — минимальный размер оплаты труда (МРОТ), используемый для расчета базовых зарплат.

Методы:

- `void setCoefficient(const Position &position, const double &newCoefficient, Director &director)` — изменяет коэффициент зарплаты для указанной должности;

- `double getMinimalSalary(Position position) const` — возвращает итоговую зарплату для указанной должности, рассчитанную на основе минимальной зарплаты и соответствующего коэффициента;
- `void printSalaryInfo() const` — выводит информацию о минимальных зарплатах для каждой должности;
- `map<Position, double> &getSalaryCoefficients()` — возвращает ссылку на список коэффициентов для каждой должности.

Конструкторы и деструкторы:

- `SalaryDatabase()` — конструктор по умолчанию. Устанавливает минимальную зарплату в размере 19 242 и инициализирует коэффициенты для должностей:
  - для должности `Position::Driver` коэффициент равен 1.5;
  - для должности `Position::Director` коэффициент равен 3.5;
  - для должности `Position::SecurityGuard` коэффициент равен 1.2;
  - для должности `Position::Secretary` коэффициент равен 1.8;
  - для должности `Position::Accountant` коэффициент равен 2.0.
- `SalaryDatabase(const double &minWage)` — параметризованный конструктор. Устанавливает минимальную зарплату в переданное значение и инициализирует коэффициенты для должностей аналогично конструктору по умолчанию.

### 2.2.3 Класс Director

Назначение: класс `Director` наследуется от базового класса `Person` и представляет собой объект, который обладает полномочиями управлять сотрудниками, распределять бюджеты по отделам и работать с базой данных зарплат.

Поля:

- `PersonalData personalData` — структура для хранения личных данных директора (имя, отчество, фамилия, зарплата);

- `SalaryDatabase salaryDatabase` — база данных зарплат для управления коэффициентами и минимальной зарплатой;

- `map<Person *, Position> employees` — ассоциативный контейнер, сотрудников и их должностей;

- `map<Department, double> departmentBudgets` — ассоциативный контейнер, отделов и утвержденного бюджета для каждого из них.

Методы:

- Геттеры и сеттеры для полей `Director`:

- `void setFirstName(const string &firstName)` — задает имя директора;

- `string getFirstName() const` — возвращает имя директора;

- `void setLastName(const string &lastName)` — задает фамилию директора;

- `string getLastName() const` — возвращает фамилию директора;

- `void setMiddleName(const string &middleName)` — задает отчество директора;

- `string getMiddleName() const` — возвращает отчество директора;

- `void setSalary(const double salary)` — задает зарплату директора;

- `double getSalary() const` — возвращает зарплату директора.

- Управление сотрудниками:

- `map<Person *, Position> getEmployees() const` — возвращает список сотрудников, находящихся в подчинении директора;

- `void hireEmployee(Person *employee, const Position &position)` — нанимает сотрудника на указанную должность;

- `void terminateEmployee(Person *employee)` — увольняет сотрудника.

- Управление бюджетами:

- `void approveBudget(const Department &department, double budget)` — утверждает бюджет для указанного отдела;

- `map<Department, double> &getApprovedBudget()` — возвращает утвержденный бюджет всех отделов.

- Работа с базой данных зарплат:
    - SalaryDatabase &getSalaryDataBase() — возвращает ссылку на базу данных зарплат;
  - Поиск сотрудников:
    - Person \*findAccountant() const — ищет бухгалтера среди сотрудников;
    - Person \*findSecretary() const — ищет секретаря среди сотрудников;
  - Прочие методы:
    - void displayInfo() — выводит полную информацию о директоре;
- Конструкторы и деструкторы:
- Director() — конструктор по умолчанию. Устанавливает стандартные значения для полей:
    - имя, отчество и фамилия устанавливаются как "NULL";
    - зарплата рассчитывается на основе минимальной зарплаты для должности Director;
      - Department::Bonuses — 20 000;
      - Department::Finance — 200 000;
      - Department::Management — 200 000;
      - Department::Security — 200 000;
      - Department::Transportation — 200 000;
      - Department::Executive — 100 000 минус зарплата директора.
  - Director(const PersonalData &personalData, SalaryDatabase &salaryDatabase) — параметризованный конструктор. Устанавливает личные данные директора и зарплату на основе минимальной зарплаты для должности Director. Инициализация бюджетов аналогична конструктору по умолчанию.

#### 2.2.4 Класс Accountant

Назначение: класс Accountant представляет бухгалтера, который отвечает за учет ставок, бонусов и зарплат сотрудников на основе данных из базы зарплат.

Бухгалтер наследуется от базового класса `Person` и управляет персональными данными сотрудников, а также взаимодействует с базой данных зарплат.

Поля:

- `PersonalData personalData` — объект, содержащий персональные данные сотрудника, такие как имя, отчество, фамилия, и зарплата;
- `SalaryDatabase salaryDatabase` — объект базы данных зарплат, предоставляющий информацию о минимальной зарплате и коэффициентах для расчета зарплат;
- `map<Person *, double> employeesRate` — контейнер для хранения ставок каждого сотрудника;
- `map<Person *, double> employeesBonuses` — контейнер для хранения премий каждого сотрудника.

Методы:

- `void setFirstName(const string &firstName)` — устанавливает имя сотрудника;
- `string getFirstName() const` — возвращает имя сотрудника;
- `void setLastName(const string &lastName)` — устанавливает фамилию сотрудника;
- `string getLastName() const` — возвращает фамилию сотрудника;
- `void setMiddleName(const string &middleName)` — устанавливает отчество сотрудника;
- `string getMiddleName() const` — возвращает отчество сотрудника;
- `void setSalary(double salary)` — устанавливает зарплату сотрудника;
- `double getSalary() const` — возвращает зарплату сотрудника;
- `void applyBonuses(Person *employee, const double bonus, Director &director)` — изменяет премию сотрудника (требуется согласие директора);
- `void setRate(Person *employee, double rate, Director &director)` — изменяет ставку сотрудника (требуется согласие директора);



- `void updateEmployeesRate(Director &director)` — обновляет ставки для всех сотрудников в системе;
- `double getEmployeeRate(Person *employee) const` — возвращает ставку для конкретного сотрудника;
- `double getEmployeeBonus(Person *employee) const` — возвращает премию для конкретного сотрудника;
- `SalaryDatabase &getSalaryDataBase()` — возвращает ссылку на объект базы данных зарплат;
- `Position getEmployeeType(Person *employee)` — возвращает тип должности сотрудника на основе его данных;
- `map<Person *, double> getEmployeesRates() const` — возвращает все ставки сотрудников;
- `void displayInfo()` — выводит полную информацию о бухгалтере, включая персональные данные и информацию о его действиях с сотрудниками.

Конструкторы и деструкторы:

`Accountant(const PersonalData &personalData, SalaryDatabase &salaryDatabase)` — конструктор, инициализирующий объект бухгалтерии с персональными данными сотрудника и базой данных зарплат.

### 2.2.5 Класс Secretary

Назначение: класс `Secretary` используется для управления расписанием встреч, а также для работы с данными о сотрудниках, такими как планирование мероприятий, вывод списков сотрудников и встреч. Он наследуется от базового класса `Person`.

Поля:

- `PersonalData personalData` — структура, содержащая персональные данные секретаря (имя, фамилия, отчество, зарплата);
- `vector<Meeting> meetings` — контейнер, хранящий список встреч, где каждая встреча содержит информацию о времени, месте и участниках.

Методы:

- `void setFirstName(const string &firstName) override` — устанавливает имя секретаря;
- `string getFirstName() const override` — возвращает имя секретаря;
- `void setLastName(const string &lastName) override` — устанавливает фамилию секретаря;
- `string getLastName() const override` — возвращает фамилию секретаря;
- `void setMiddleName(const string &middleName) override` — устанавливает отчество секретаря;
- `string getMiddleName() const override` — возвращает отчество секретаря;
- `void setSalary(double salary) override` — устанавливает зарплату секретаря;
- `double getSalary() const override` — возвращает зарплату секретаря.
- `void scheduleMeeting(vector<Person *> employees, const string &place, const Date &date)` — создает новую встречу, добавляя ее в список встреч;
- `vector<Meeting> &listScheduledMeetings()` — возвращает ссылку на список всех запланированных встреч;
- `void displayMeetings() const` — выводит на экран список всех запланированных встреч;
- `void displayEmployeeListInTable(const map<Person *, Position> &employees) const` — выводит список сотрудников с указанием их должностей в табличной форме;
- `void participantsOfMeeting(Meeting meeting) const` — выводит информацию об участниках указанной встречи;
- `void displayInfo() const` — выводит всю информацию о классе, включая данные секретаря, список встреч и другие детали.

Конструкторы и деструкторы:

`Secretary(const PersonalData &personalData)` — конструктор, принимающий объект структуры `PersonalData`. Устанавливает имя, фамилию и отчество секретаря с использованием геттеров и сеттеров.

### 2.2.6 Класс `SecurityGuard`

Назначение: класс `SecurityGuard` предназначен для представления сотрудников службы безопасности, выполняющих задачи проверки безопасности на объектах, использования специального оборудования и составления отчетов о выявленных инцидентах.

Поля:

- `PersonalData personalData` — структура, содержащая персональные данные сотрудника (имя, фамилия, отчество, зарплата);
- `map<string, string> locations` — карта, где ключ представляет объект (например, "Склад", "Производство"), а значение — текущий статус проверки объекта;
- `Equipment specialEquipment` — тип используемого специального оборудования (например, дубинка, шокер, фонарь);
- `vector<string> departments` — список всех объектов для проверки ("Склад", "Производство", "Администрация", "Лаборатория")
- `vector<string> statuses` — список возможных статусов проверки объекта ("Всё в порядке", "Нужна проверка", "Нарушение безопасности", "Неизвестный статус").

Методы:

- `void setFirstName(const string &firstName)` — устанавливает имя охранника;
- `string getFirstName() const` — возвращает имя охранника;
- `void setLastName(const string &lastName)` — устанавливает фамилию охранника;

- `string getLastName() const` — возвращает фамилию охранника;
- `void setMiddleName(const string &middleName)` — устанавливает отчество охранника;
- `string getMiddleName() const` — возвращает отчество охранника;
- `void setSalary(double salary)` — устанавливает заработную плату охранника;
- `double getSalary() const` — возвращает заработную плату охранника;
- `void setSecurityStatus(string &department, string &status)` — изменяет статус безопасности для указанного объекта;
- `void generateIncidentReport() const` — генерирует отчет о безопасности с информацией о текущем статусе каждого объекта;
- `void setSpecialEquipment(const Equipment equipment)` — устанавливает используемое специальное оборудование;
- `Equipment getSpecialEquipment() const` — возвращает текущее специальное оборудование охранника;
- `vector<string> allDepartments()` — возвращает список всех объектов для проверки;
- `vector<string> allStatuses()` — возвращает список возможных статусов проверки объектов;
- `void displayInfo() const` — отображает всю информацию о классе, включая данные охранника, текущие статусы проверок и используемое оборудование.

Конструкторы и деструкторы:

`SecurityGuard(const PersonalData &personalData, const Equipment &equipment)` — параметризованный конструктор, инициализирующий личные данные сотрудника, используемое специальное оборудование (по умолчанию — фонарь) и статусы проверок всех объектов как "Нужна проверка".

### 2.2.7 Класс Driver

Назначение: класс Driver предназначен для представления данных о водителе, таких как личная информация, категории водительских прав, транспортные средства, а также маршрутные назначения.

Поля:

- `PersonalData personalData` — структура, содержащая личные данные водителя (имя, фамилия, отчество, зарплата);
- `string destination` — текущее место назначения водителя;
- `vector<DriverLicense> licenseCategories` — массив категорий водительских прав (например, А, В, С);
- `vector<VehicleType> vehicles` — массив транспортных средств, которыми управляет водитель;
- `vector<string> cities` — список городов, доступных для назначения маршрута.

Методы:

- `void setFirstName(const string &firstName)` — устанавливает имя водителя;
- `string getFirstName() const` — возвращает имя водителя;
- `void setLastName(const string &lastName)` — устанавливает фамилию водителя;
- `string getLastName() const` — возвращает фамилию водителя;
- `void setMiddleName(const string &middleName)` — устанавливает отчество водителя;
- `string getMiddleName() const` — возвращает отчество водителя;
- `void setSalary(double salary)` — устанавливает зарплату водителя;
- `double getSalary() const` — возвращает зарплату водителя;
- `void changeDestination(const string &newDestination)` — изменяет место назначения;

- `const vector<string> getCities()` — возвращает список доступных городов для назначения маршрута;
- `void addLicenseCategory(const DriverLicense &category)` — добавляет новую категорию водительских прав;
- `vector<DriverLicense> &getLicenseCategories()` — возвращает список категорий водительских прав;
- `void addVehicle(const VehicleType &vehicle)` — добавляет новое транспортное средство;
- `vector<VehicleType> &getVehicles()` — возвращает список транспортных средств водителя;
- `void displayInfo() const` — отображает всю информацию о водителе.

Конструкторы и деструкторы:

`Driver(const PersonalData &personalData, DriverLicense licenseCategories, VehicleType vehicles)` — параметризованный конструктор. Устанавливает личные данные водителя, добавляет категорию прав и транспортное средство, инициализирует место назначения как "Не задано" и список доступных городов.

### 2.2.8 Класс Company

Назначение: класс Company используется для управления информацией о компании и директоре, связанном с ней.

Поля:

- `string name` — название компании;
- `Director director` — директор, связанный с данной компанией.

Методы:

- `Director &getDirector()` — возвращает ссылку на объект класса Director, связанный с данной компанией;
- `void displayInfo() const` — выводит информацию о компании и директоре;
- `void setName(string name)` — устанавливает новое название компании;

- `string getName() const` — возвращает текущее название компании.

Конструкторы и деструкторы:

- `Company()` — конструктор по умолчанию, инициализирует название компании значением "NULL";
- `Company(const string &name, Director director)` — параметризованный конструктор, задающий название компании и привязывающий к ней директора.

### 2.2.9 Класс Menu

Назначение: Класс Menu используется для управления взаимодействием пользователя с системой. Реализует функционал отображения и выполнения различных меню и действий, связанных с управлением компаниями, сотрудниками, финансами и другими элементами системы.

Поля:

- `int choice` — выбор пользователя в текущем меню;
- `vector<Company> companies` — список всех созданных компаний.

Методы:

- `Position getEmployeeType(Person *employee);` — определяет должность сотрудника на основе указателя на объект `Person`;
- `void displayAllEmployees(Director &director, vector<Person *> &employees);` — выводит в табличном виде информацию обо всех сотрудниках (должность, ФИО, зарплата);
- `void displayAllEmployeesWithInt(Director &director, vector<int> &indexes);` — выводит в табличном виде информацию о выбранных сотрудниках (должность, ФИО, зарплата);
- `void displayAllEmployeesWithRatesAndBonuses(Director &director, Accountant &accountant);` — выводит в табличном виде информацию о сотрудниках с учетом ставок и премий;
- `void displayALLPositionsWithKoef(Accountant &accountant);` — выводит информацию о должностях, их коэффициентах и зарплатах;

- `vector<Company> getListOfCompanies() const;` — возвращает список всех компаний;
- `bool createCompanyManually(Company &company);` — создает компанию вручную;
- `bool changeNameCompany(Company &company);` — изменяет название компании;
- `bool deleteCompany(Company &companyToRemove);` — удаляет компанию из списка;
- `bool setFIO(Person &person);` — изменяет ФИО сотрудника;
- `bool hireEmployeeMenu(Director &director);` — позволяет директору нанять нового сотрудника с выбором должности и проверкой бюджета отдела;
- `bool terminateEmployeeMenu(Director &director);` — позволяет директору уволить сотрудника с перерасчетом бюджета отдела;
- `bool createMeeting(Secretary &secretary, Director &director);` — создает встречу с указанием времени, названия, участников и места;
- `bool getParticipantOfMeeting(Secretary &secretary);` — выводит информацию обо всех встречах;
- `bool deleteMeeting(Secretary &secretary);` — удаляет выбранную встречу;
- `string generateRandomFirstName() const;` — генерирует случайное имя;
- `string generateRandomMiddleName() const;` — генерирует случайную фамилию;
- `string generateRandomLastName() const;` — генерирует случайное отчество;
- `string generateRandomCompanyName() const;` — генерирует случайное название компании;
- `Company createCompanyRandom();` — создает компанию с случайными параметрами.
- `bool inputSingleInt(int &digit, const int start, const int end);` — ввод положительного числа в указанном диапазоне;



- `bool inputString(string &str, const int &minLen = 2, const int &maxLen = 15, bool firstBigLetter = 1, bool allowNumbers = 0);` — ввод строки с проверкой длины, формата и допустимых символов;
- `bool inputInt(int &digit, const int start, const int end);` — ввод целого числа с учетом заданных ограничений;
- `bool inputDouble(double &number, const double &min = __DBL_MIN__ / 10, const double &max = __DBL_MAX__ / 10, const string &key = "NULL", const bool minus = 0, const bool point = 1, int decimalLimit = 2);` — ввод числа с плавающей запятой;
- `void clearConsole();` — очищает консоль и выводит сообщение "нажмите ESC, чтобы выйти";
- `bool createCompanyMenu();` — отображает меню создания компании;
- `bool CompanyMenu();` — меню для выбора между созданием и управлением компаниями;
- `bool manageCompaniesMenu();` — меню выбора компании для управления;
- `bool workWithCompany(Company &company);` — управление конкретной компанией;
- `bool manageEmployeesMenu(Director &director);` — меню управления сотрудниками компании;
- `bool manageBudgetDepartmentsMenu(Director &director);` — меню управления бюджетом отделов;
- `bool meetingManageMenu(Secretary &secretary, Director &director);` — меню управления встречами.

Конструкторы и деструкторы: в классе `Menu` отсутствуют явно объявленные конструкторы и деструкторы, так как используется стандартный конструктор по умолчанию.

## 2.3 Описание пользовательского интерфейса

Интерфейс программы разработан в виде консольного меню, которое позволяет пользователю последовательно выполнять действия, начиная с создания компании и заканчивая управлением её сотрудниками. Взаимодействие с программой организовано таким образом, чтобы обеспечить интуитивно понятный и логичный пользовательский опыт.

Приложение консольное, следовательно взаимодействие с ним осуществляется в виде диалога с пользователем.

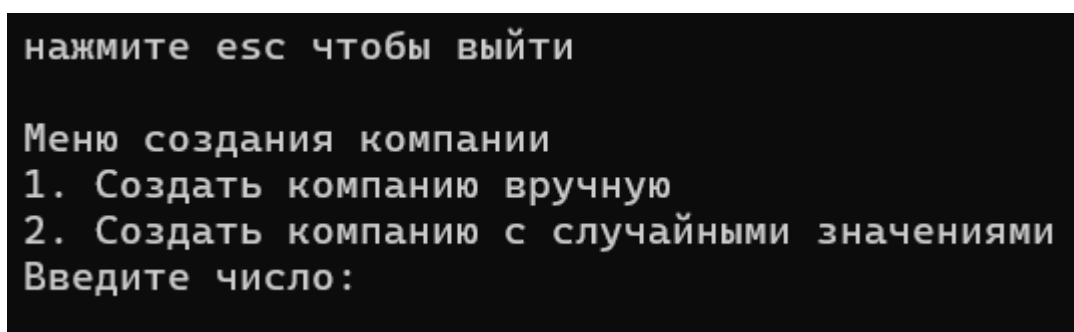
Для всего меню актуально правило – Пользователь может вернуться на шаг назад путём нажатия клавиши "esc".

### Начало работы

При запуске программы пользователь попадает в стартовое меню (Рис. 2.28), где ему предлагается выбрать один из режимов работы:

- создание компании вручную;
- создание компании со случайными значениями.

Интерфейс программы направляет пользователя через последовательность экранов, обеспечивая пошаговое выполнение действий.



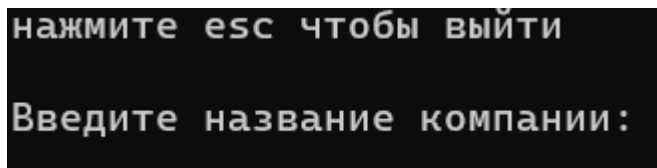
*Рис. 2.28. Стартовое меню программы*

### Режим создания компании вручную

Если пользователь выбирает пункт "Создать компанию вручную", система предлагает ввести следующие данные:

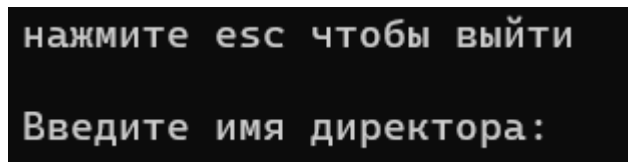
- название компании (Рис. 2.29);
- имя директора (Рис. 2.30);

- фамилию директора (Рис. 2.31);
- отчество директора (Рис. 2.32);



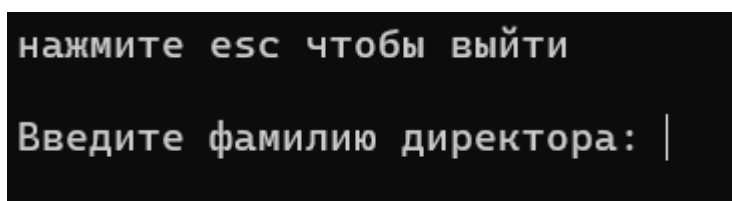
нажмите есc чтобы выйти  
Введите название компании:

*Рис. 2.29. Ввод названия компании*



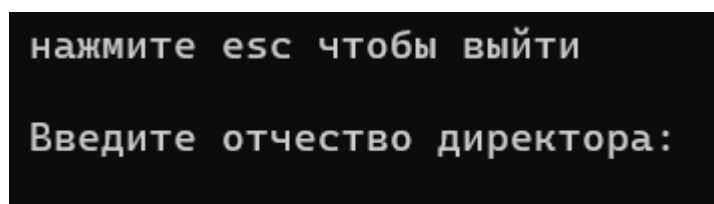
нажмите есc чтобы выйти  
Введите имя директора:

*Рис. 2.30. Ввод имя директора*



нажмите есc чтобы выйти  
Введите фамилию директора: |

*Рис. 2.31. Ввод фамилии директора*



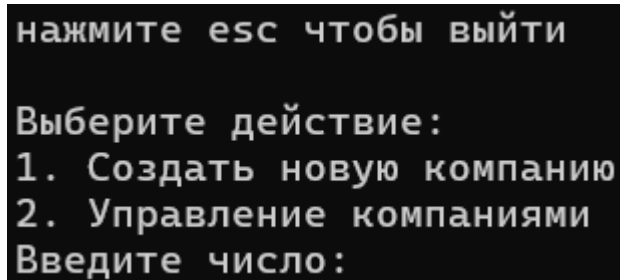
нажмите есc чтобы выйти  
Введите отчество директора:

*Рис. 2.32. Ввод отчества директора*

#### Режим случайного создание компании

Если пользователь выбирает пункт "Создать компанию с случайными значениями", система автоматически генерирует параметры компании (название, данные директора, базовую зарплату).

После того как пользователь выбрал режим случайного создание компании или ручного заполнения компании, он попадает в меню управления компаниями (Рис. 2.33).

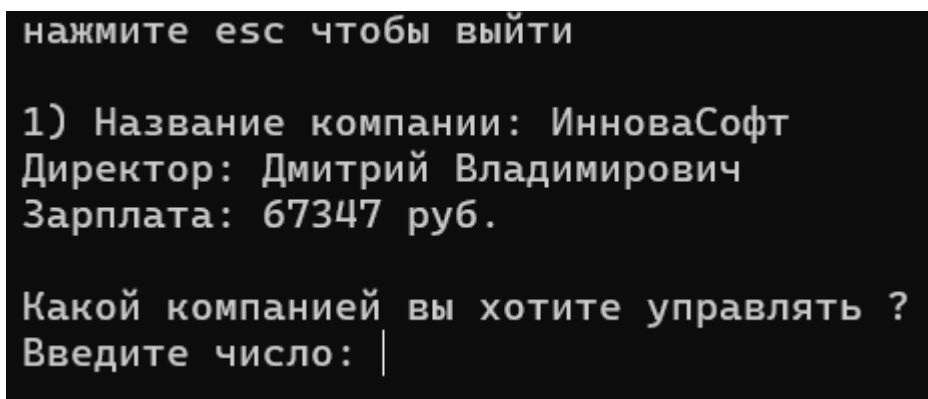


```
нажмите esc чтобы выйти  
Выберите действие:  
1. Создать новую компанию  
2. Управление компаниями  
Введите число:
```

*Рис. 2.33. Меню выбора создать или управлять компаниями*

### Управление компаниями

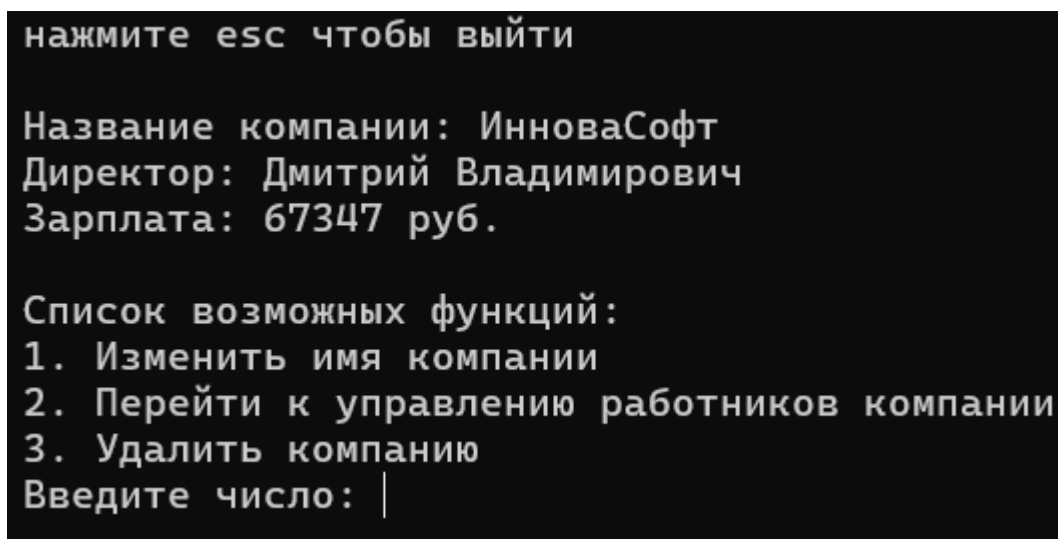
При выборе пункта "Управление компаниями" программа отображает список всех созданных компаний (Рис. 2.34). Пользователь выбирает одну из них для дальнейшей работы.



```
нажмите esc чтобы выйти  
  
1) Название компании: ИнноваСофт  
Директор: Дмитрий Владимирович  
Зарплата: 67347 руб.  
  
Какой компанией вы хотите управлять ?  
Введите число: |
```

*Рис. 2.34. Меню выбора компании для управления*

После выбора пользователем компании, появится новое окно со всеми возможными функциями взаимодействия с выбранной компанией (Рис. 2.35).



```
нажмите esc чтобы выйти  
  
Название компании: ИнноваСофт  
Директор: Дмитрий Владимирович  
Зарплата: 67347 руб.  
  
Список возможных функций:  
1. Изменить имя компании  
2. Перейти к управлению работников компании  
3. Удалить компанию  
Введите число: |
```

*Рис. 2.35. Меню всех действий с компанией*

При выборе первого варианта, появится окно с изменением имени компании (Рис. 2.29)

Если пользователь решит удалить компанию, то появится окно с демонстрацией успешного выполнения действия (Рис. 2.36).

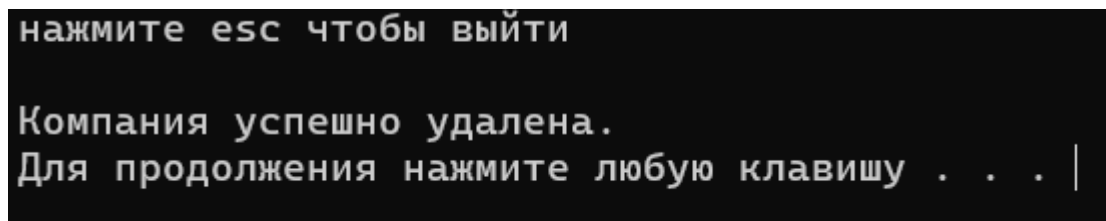


Рис. 2.36. Вывод сообщения об успешном выполнении удаления компании

### Управление работниками компании

При выборе пункта "Перейти к управлению работниками компании", программа отобразит список всех сотрудников компании в виде таблицы, включая такие данные, как должность, ФИО и заработная плата. Пользователь может выбрать любого работника для дальнейшего взаимодействия.

Пример отображения списка сотрудников ниже (Рис. 2.37).

нажмите esc чтобы выйти

Сотрудники данной компании:

№	Должность	Фамилия	Имя	Отчество	Зарплата
0	Директор	Васильев	Андрей	Викторович	67347.00 руб.
1	Бухгалтер	Алексеев	Дмитрий	Алексеевич	38484.00 руб.
2	Секретарь	Волков	Алексей	Сергеевич	34635.60 руб.
3	Водитель	Алексеев	Александр	Анатольевич	28863.00 руб.
4	Водитель	Попов	Дмитрий	Григорьевич	28863.00 руб.
5	Охранник	Попов	Никита	Иванович	23090.40 руб.
6	Охранник	Кузнецов	Дмитрий	Никитич	23090.40 руб.

Выберите сотрудника: |

Рис. 2.37. Меню выбора сотрудника

Меню функций для директора (Рис. 2.38)

нажмите esc чтобы выйти

№	Должность	Фамилия	Имя	Отчество	Зарплата
0	Директор	Егоров	Андрей	Петрович	67347.00 руб.

Бюджет отделов:

Отдел транспорта	142274.00 руб.
Отдел управление	165364.40 руб.
Отдел безопасности	153819.20 руб.
Финансовый отдел	161516.00 руб.
Исполнит. отдел	32653.00 руб.
Бюджет на премии	20000.00 руб.

Список возможных функций:

1. Изменить ФИО
2. Нанять работника
3. Уволить работника
4. Вывести всех работников
5. Управление бюджетом отделов

Введите число: |

Рис. 2.38. Меню действий директора

#### 1. Нанять работника

После выбора этой функции появится окно с выбором должности нового сотрудника (Рис. 2.39.), затем программа предложит ввести имя, фамилию и отчество сотрудника.

нажмите esc чтобы выйти

Выберите тип работника для найма:

1. Водитель
2. Секретарь
3. Охранник
4. Бухгалтер

Введите число: |

Рис. 2.39. Меню выбора должности работника

#### 2. Уволить работника

После выбора этой функции появится список всех сотрудников данного директора в виде таблицы, включая такие данные, как должность, ФИО и заработная плата (Рис. 2.40.)

нажмите есc чтобы выйти

Список сотрудников:

№	Должность	Фамилия	Имя	Отчество	Зарплата
1	Бухгалтер	Алексеев	Дмитрий	Алексеевич	38484.00 руб.
2	Секретарь	Волков	Алексей	Сергеевич	34635.60 руб.
3	Водитель	Алексеев	Александр	Анатольевич	28863.00 руб.
4	Водитель	Попов	Дмитрий	Григорьевич	28863.00 руб.
5	Охранник	Попов	Никита	Иванович	23090.40 руб.
6	Охранник	Кузнецов	Дмитрий	Никитич	23090.40 руб.

Введите номер сотрудника, которого хотите уволить: |

Рис. 2.40. Меню выбора сотрудника компании

### 3. Вывести всех работников

После выбора этой функции появится список всех сотрудников данного директора в виде таблицы, включая такие данные, как должность, ФИО и заработная плата (Рис. 2.37.)

### 4. Управление бюджетом отделов

После выбора этой функции появится список всех отделов и их текущий бюджет (Рис. 2.41).

нажмите есc чтобы выйти

Бюджет отделов:

1. Отдел транспорта	(Бюджет: 142274.00 руб.)
2. Отдел управление	(Бюджет: 200000.00 руб.)
3. Отдел безопасности	(Бюджет: 153819.20 руб.)
4. Финансовый отдел	(Бюджет: 161516.00 руб.)
5. Исполнит. отдел	(Бюджет: 32653.00 руб.)
6. Бюджет на премии	(Бюджет: 20000.00 руб.)

Введите число: |

Рис. 2.41. Меню выбора отдела

## Меню функций для Секретаря (Рис. 2.42)

нажмите esc чтобы выйти

№	Должность	Фамилия	Имя	Отчество	Зарплата
0	Секретарь	Степанов	Алексей	Анатольевич	34635.60 руб.

Список возможных функций:  
1. Изменить ФИО  
2. Вывести сотрудников, данного директора  
3. Вывести данные об охранниках  
4. Вывести данные о водителях  
5. Управление встречами  
Введите число: |

Рис. 2.42. Меню выбора функций у секретаря

### 2.1 Вывести сотрудников, данного директора

После выбора этой функции появится список всех сотрудников данного директора в виде таблицы, включая такие данные, как должность, ФИО и заработная плата (Рис. 2.37)

### 2.2 Вывести данные о водителях

## Меню функций для Бухгалтера (Рис. 2.43)

нажмите esc чтобы выйти

№	Должность	Фамилия	Имя	Отчество	Зарплата	Ставка	Премия
0	Бухгалтер	Волков	Андрей	Николаевич	38484.00 руб.	1.00	0.00 руб.

Список возможных функций:  
1. Изменить ФИО  
2. Изменить ставку сотруднику  
3. Назначить премиальные сотруднику  
4. Изменить коэффициент ЗП должности  
5. Вывести информацию о всех работниках  
Введите число: |

Рис. 2.43. Меню выбора функций у бухгалтера

## Меню функций для Охранника (Рис. 2.44)



нажмите esc чтобы выйти

№	Должность	Фамилия	Имя	Отчество	Зарплата
0	Охранник	Павлов	Дмитрий	Дмитриевич	23090.40 руб.

Спец средство: Фонарь  
 Список возможных функций:  
 1. Изменить ФИО  
 2. Управление спец. средством  
 3. Управление статусом объектов  
 Введите число: |

Рис. 2.44. Меню выбора функций у охранника

### Функции для Водителя (Рис. 2.45)

нажмите esc чтобы выйти

№	Должность	Фамилия	Имя	Отчество	Зарплата
0	Водитель	Михайлов	Дмитрий	Дмитриевич	28863.00 руб.

Имеющиеся категории прав : Категория А  
 Имеющиеся транспортные средства : Мотоцикл  
 Выехал в город – Не задано  
 Список возможных функций:  
 1. Изменить ФИО  
 2. Управление водительскими правами  
 3. Управление транспортом  
 4. Отправить водителя в путешествие  
 Введите число: |

Рис. 2.45. Меню выбора функций у водителя

## **ГЛАВА 3. Реализация и тестирование программы**

В данной главе представлена реализация программы для управления тюрьмой, включая основные компоненты программы, их взаимосвязи и особенности реализации. Программа разработана на языке программирования C++ с применением объектно-ориентированного подхода, что обеспечивает модульность, читаемость и легкость в сопровождении кода.

В первой части главы приведен листинг программы, включающий описание классов, методов и ключевых алгоритмов. В частности, описаны базовый класс Человек, а также дочерние классы для ролей директора, секретаря, бухгалтера, водителя и охранника. Реализация программы охватывает функции управления персоналом, учета заключенных, планирования задач и других процессов.

Во второй части главы рассматривается процесс тестирования программы. Приведены сценарии тестирования, результаты выполнения программы на различных наборах данных, а также анализ корректности работы ключевых функций.

### **3.1 Листинг программы**

В данном разделе представлена реализация программы на языке C++ с использованием объектно-ориентированного подхода. Программа была разработана на основе структуры классов, описанной в предыдущих разделах, и реализует функционал, соответствующий требованиям задачи.

В рамках реализации были определены следующие классы:

- Человек (базовый класс) — описывает общие свойства и поведение пользователей системы.
- Директор, Секретарь, Бухгалтер, Водитель, Охранник — дочерние классы, реализующие специфический функционал для каждой роли.

Листинг основного файла программы (Приложение 2 - Приложение 8)

Код, реализующий базовый класс Человек и его основные свойства (Приложение 2).

Код, реализующий базовый класс Директор и его основные свойства (Приложение 3).

Код, реализующий базовый класс Бухгалтер и его основные свойства (Приложение 4).

Код, реализующий базовый класс Секретарь и его основные свойства (Приложение 5).

Код, реализующий базовый класс Охранник и его основные свойства (Приложение 6).

Код, реализующий базовый класс Водитель и его основные свойства (Приложение 7).

### **3.2 Тестирование программы**

В данном разделе представлена проверка работоспособности программы, её функций и сценариев взаимодействия с пользователем. Тестирование проводилось для проверки корректности выполнения всех действий, предусмотренных функционалом системы, а также предотвращения некорректных действий пользователя.

В программе есть ограничения, которые всегда работают, не зависимо в каком окне сейчас пользователь:

- у пользователя нет возможности в меню выбрать недоступное для него действия, так как программа допускает ввод только корректного ввода, путем отключения всех лишних клавиш на клавиатуре;
- нельзя ввести строку длиннее, чем 15 символов, так как после 15 символа отключается любой ввод с клавиатуры;
- нельзя продолжить выполнение программы, если ввести строку меньше двух символов. В этом случае клавиша “enter” не будет работать;

- если программа запрашивает ввод числа и пользователь попытается ввести не допустимое значение, то программа на этапе ввода будет ограничивать пользователя в вводе числа, что предотвращает ввод некорректного значения.

#### Сценарий №1. Запуск программы и выбор метода создания компании

После запуска программы пользователю предоставляется возможность выбрать один из двух вариантов инициализации:

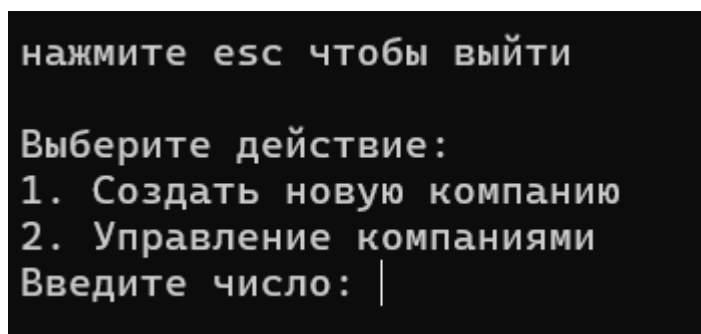
##### 1. Случайное создание компании.

Количество созданных компаний ограничено количеством 9 штук.

В этом режиме создается компания со случайно сгенерированным названием, а также автоматически добавляются: директор, секретарь, бухгалтер, два охранника, два водителя.

Программа корректно создает указанные объекты и заполняет их данными.

Корректный результат выполнения данной команды (Рис. 3.1.).



*Рис. 3.1. Результат создания компании через автоматическое заполнение*

Можем заметить, что никаких ошибок не появилось, и программа продолжила свою работу.

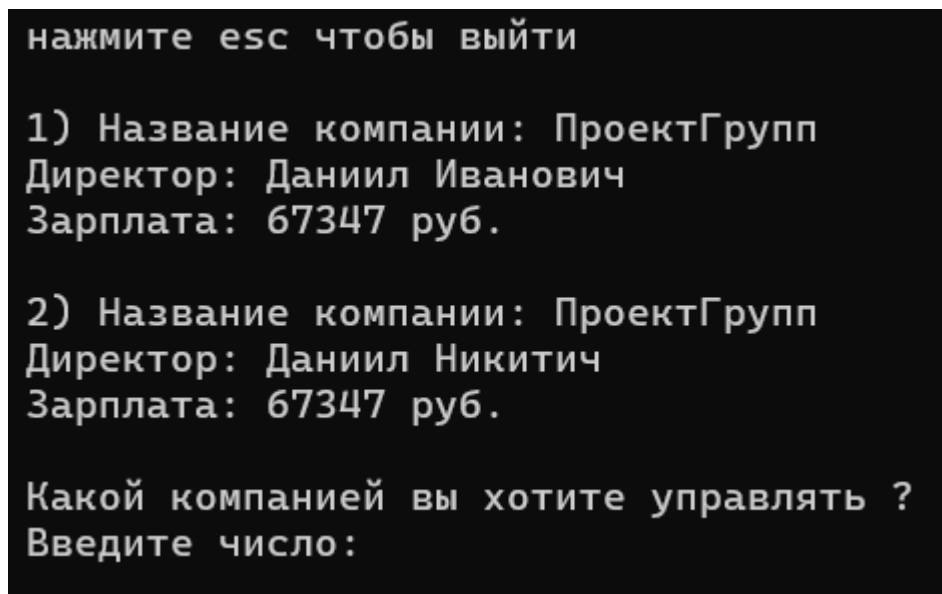
##### 2. Ручное заполнение данных.

В этом режиме пользователь вручную вводит название компании и ФИО директора. Программа проверяет введенные данные и не позволяет продолжить работу, если они некорректны (например, если оставлены пустые поля или длина поля меньше чем 2 символа, тогда кнопка enter не будет работать). Также есть возможность ввода двойных фамилий/имен/отчеств через символ “-”. Этот сценарий также отрабатывается корректно, с учетом постоянных

ограничений описанных выше, обеспечивая полное управление вводом данных. Корректный результат выполнения данной команды (Рис. 3.1).

#### Сценарий №2. Выбор компании для управления

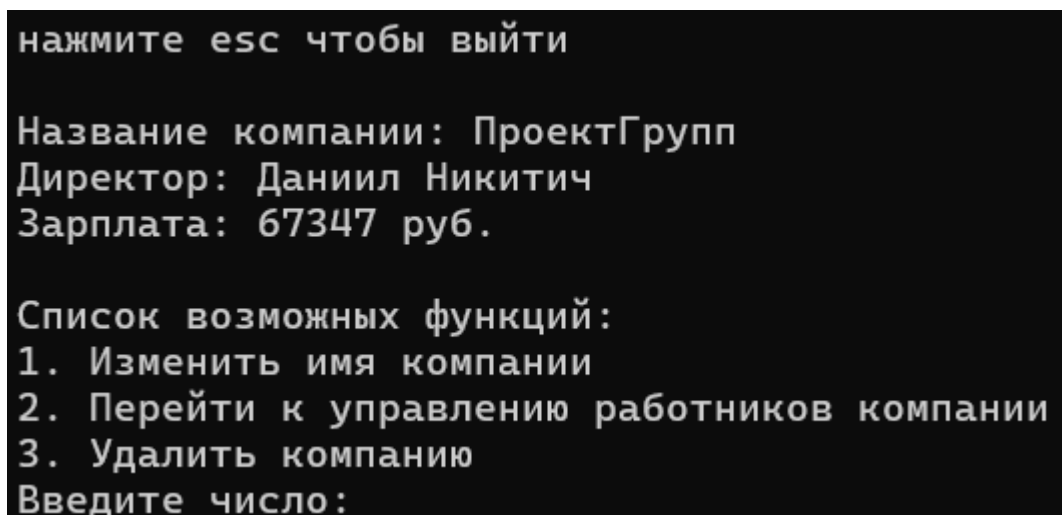
После создания компании пользователь может перейти в раздел с выбором компании для дальнейшей работы с ними (Рис. 3.2).



*Рис. 3.2. Окно выбора компании*

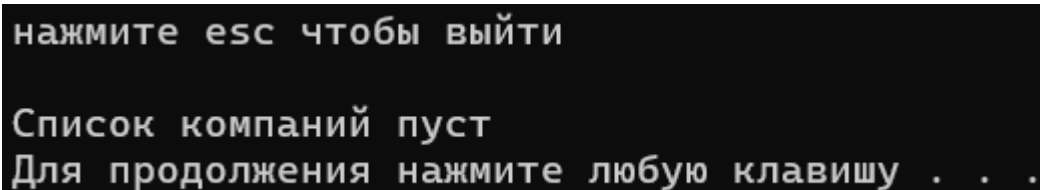
В данном окне пользователь не может выбрать номер компании, которой нет, так как программа не будет считывать номер не существующей компании.

Результат выбора второй компании для управления (Рис. 3.3).



*Рис. 3.3. Меню управления компанией*

В случае если пользователь не создал компанию и попытается перейти в раздел “2. Управления компаниями”, то сработает исключение (Рис. 3.4).



нажмите esc чтобы выйти  
Список компаний пуст  
Для продолжения нажмите любую клавишу . . .

*Рис. 3.4. Ошибка при выводе списка компаний*

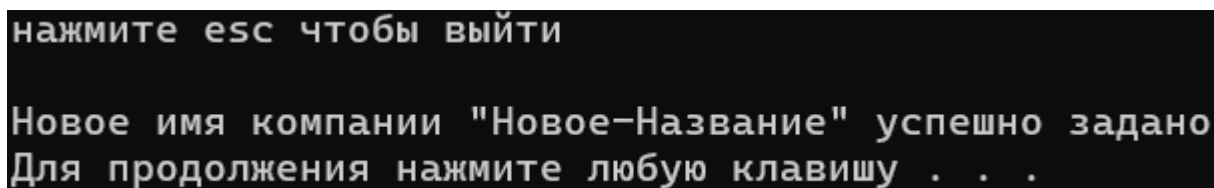
### Сценарий №3. Управление компанией

В это окне пользователь может выбрать следующие действия: изменения имени компании, перейти к управлению работниками компании, удалить компанию (Рис. 3.3).

#### 1. Изменения имени компании

При выборе этого действия применяются постоянные ограничения, описанные выше.

Результат работы данной операции (Рис 3.5).

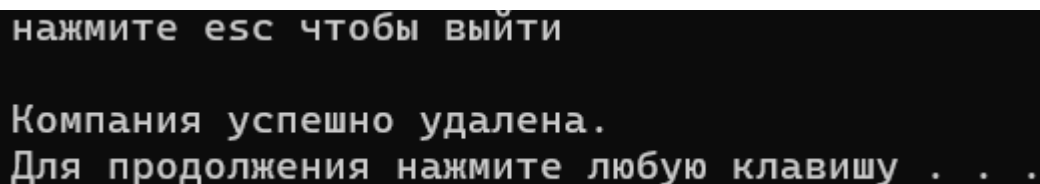


нажмите esc чтобы выйти  
Новое имя компании "Новое-Название" успешно задано  
Для продолжения нажмите любую клавишу . . .

*Рис 3.5. Сообщение об успешном выполнении операции “изменить название компании”*

#### 2. Удалить компанию

При выборе данной операции пользователем, будет выведено сообщение об успешном удалении компании (Рис. 3.6).



нажмите esc чтобы выйти  
Компания успешно удалена.  
Для продолжения нажмите любую клавишу . . .

*Рис. 3.6. Сообщение об успешном выполнении операции “удалить компанию”*

#### 3. Перейти к управлению работников компании

Эта операция выводит на экран список всех работников данной компании, чтобы пользователь мог выбрать любого и посмотреть его функции (Рис. 3.7).

нажмите есч чтобы выйти

Сотрудники данной компании:

№	Должность	Фамилия	Имя	Отчество	Зарплата
0	Директор	Михайлов	Андрей	Станиславович	67347.00 руб.
1	Бухгалтер	Смирнов	Никита	Евгеньевич	38484.00 руб.
2	Секретарь	Лебедев	Даниил	Дмитриевич	34635.60 руб.
3	Водитель	Федоров	Даниил	Викторович	28863.00 руб.
4	Водитель	Васильев	Андрей	Николаевич	28863.00 руб.
5	Охранник	Лебедев	Никита	Тимофеевич	23090.40 руб.
6	Охранник	Лебедев	Алексей	Михайлович	23090.40 руб.

Выберите сотрудника:

Рис. 3.7. Список всех работников компании

#### Сценарий №4. Управление директором

Если пользователь для управления выбрал директора, то у него появляется следующий выбор действий (Рис. 3.8).

нажмите есч чтобы выйти

№	Должность	Фамилия	Имя	Отчество	Зарплата
0	Директор	Михайлов	Андрей	Станиславович	67347.00 руб.

Бюджет отделов:

Отдел транспорта	142274.00 руб.
Отдел управление	165364.40 руб.
Отдел безопасности	153819.20 руб.
Финансовый отдел	161516.00 руб.
Исполнит. отдел	32653.00 руб.
Бюджет на премии	20000.00 руб.

Список возможных функций:

1. Изменить ФИО
2. Нанять работника
3. Уволить работника
4. Вывести всех работников
5. Управление бюджетом отделов

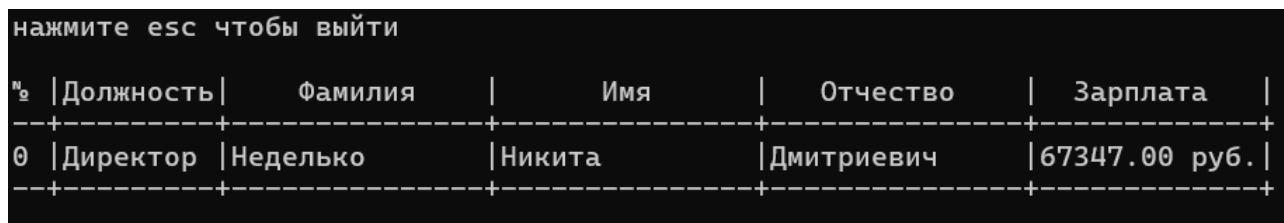
Введите число: |

Рис. 3.8. Меню управления директором

## 1. Изменить ФИО

При выборе этого действия применяются постоянные ограничения, описанные выше.

Результат работы данной операции (Рис. 3.9).

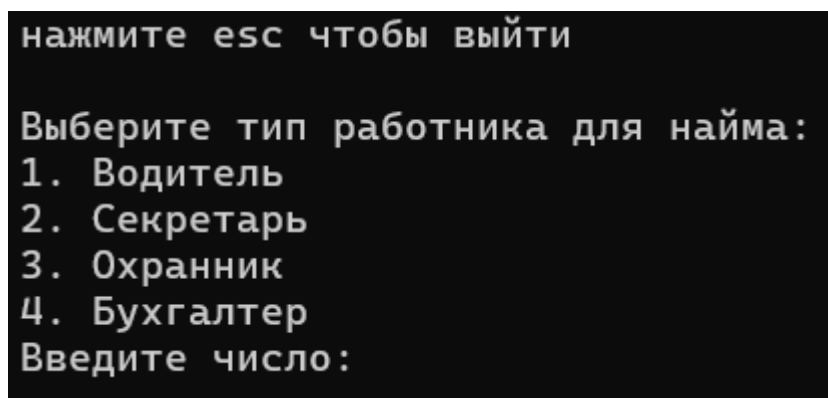


№	Должность	Фамилия	Имя	Отчество	Зарплата
0	Директор	Неделько	Никита	Дмитриевич	67347.00 руб.

Рис. 3.9. Результат выполнения изменения ФИО директора

## 2. Нанять работника

При отработке данного действия программа попросит пользователя ввести тип сотрудника для найма (Рис. 3.10).

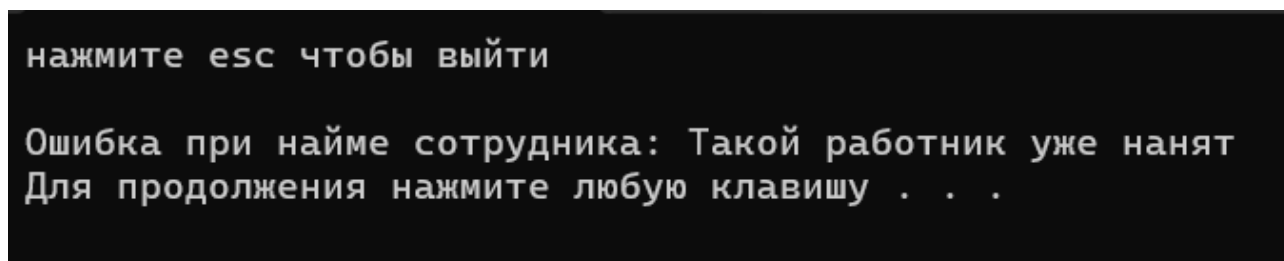


```
нажмите есч чтобы выйти

Выберите тип работника для найма:
1. Водитель
2. Секретарь
3. Охранник
4. Бухгалтер
Введите число:
```

Рис. 3.10. Меню выбора типа сотрудника для найма

Если в компании уже есть сотрудник типа Секретарь или Бухгалтер, и пользователь попытается нанять еще одного сотрудника того же типа, то появится ошибка (Рис. 3.11).



```
нажмите есч чтобы выйти

Ошибка при найме сотрудника: Такой работник уже нанят
Для продолжения нажмите любую клавишу . . .
```

Рис. 3.11. Ошибка при найме секретаря или бухгалтера

Корректной выполнение операции добавления нового сотрудника (Рис. 3.12).





нажмите esc чтобы выйти

Водитель Королев нанят на работу. С зарплатой 28863.00  
Для продолжения нажмите любую клавишу . . .

*Рис. 3.12. Ошибка при найме секретаря или бухгалтера*

### 3. Уволить работника

При выборе данного действия у пользователя нет возможности ввести не корректные данные, которые могут привести к выбросу исключения в программе, так как отрабатывают постоянные ограничения программы.

Корректное выполнение данной команды (Рис. 3.13).



нажмите esc чтобы выйти

Сотрудник Волков уволен. Бюджет отдела Отдел безопасности восстановлен на 23090.40  
Для продолжения нажмите любую клавишу . . .

*Рис. 3.13. Корректное выполнение “уволить сотрудника”*

Если в компании нет сотрудников, кроме директора, то у программы отработает исключение (Рис. 3.14).



нажмите esc чтобы выйти

Нет сотрудников для увольнения.

Для продолжения нажмите любую клавишу . . .

*Рис. 3.14. Ошибка при попытке увольнения рабочего*

### 4. Вывести всех работников

Если в компании есть работники, кроме директора, то будет выведена таблица со всеми сотрудниками (Рис. 3.15).

нажмите esc чтобы выйти

№	Должность	Фамилия	Имя	Отчество	Зарплата
1	Бухгалтер	Федоров	Александр	Иванович	38484.00 руб.
2	Секретарь	Павлов	Даниил	Владимирович	34635.60 руб.
3	Водитель	Попов	Алексей	Александрович	28863.00 руб.
4	Водитель	Новиков	Никита	Петрович	28863.00 руб.
5	Охранник	Кузнецов	Александр	Дмитриевич	23090.40 руб.
6	Охранник	Новиков	Никита	Дмитриевич	23090.40 руб.

Для продолжения нажмите любую клавишу . . .

Рис. 3.15. Вывод всех сотрудников данного директора

Если в компании нет сотрудников, кроме директора, то у программы отработает исключение (Рис. 3.14).

#### 5. Управление бюджетом отделов

У пользователя есть возможность поменять бюджет любого отдела на сумму от 0 до 1000000.

Пример корректного выполнения программы (Рис. 3.16).

нажмите esc чтобы выйти

Прошлый бюджет Финансовый отдел = 161516.00 руб.  
 Нынешний бюджет Финансовый отдел успешно изменен на 73423.00 руб.  
 Для продолжения нажмите любую клавишу . . .

Рис. 3.16. Корректное изменение бюджета отдела

Пользователь не может ввести сумму меньше или больше данного диапазона, так как отработывают постоянные ограничения, описанные выше.

#### Сценарий №5. Управление бухгалтером

##### 1. Изменить ФИО

При выборе этого действия применяются постоянные ограничения, описанные выше.

Результат работы данной операции (Рис. 3.17).

нажмите esc чтобы выйти

%	Должность	Фамилия	Имя	Отчество	Зарплата	Ставка	Премия
0	Бухгалтер	Неделько	Никита	Дмитриевич	38484.00 руб.	1.00	0.00 руб.

Рис. 3.17. Корректное изменение ФИО бухгалтера

## 2. Изменить ставку сотруднику

После того как пользователь выбрал операцию “Изменить ставку сотруднику”, и ввел номер сотрудника, у пользователя появляется окно с вводом ставки. Доступные значения в диапазоне от 0.1 до 1.

Если пользователь попытается ввести ставку равную меньше, чем 0.1, то сработает исключение и появится окно ошибки (Рис. 3.18).

**Ставка не может быть ниже 0.1.**

Рис. 3.18. Ошибка при изменении ставки сотрудника

Корректное выполнение данной операции (Рис. 3.19).

нажмите esc чтобы выйти

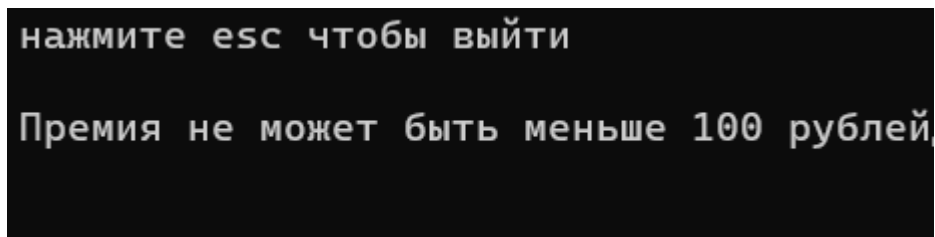
Ставка сотрудника Лебедев изменена на 0.80. Зарплата теперь 27708.48.  
Для продолжения нажмите любую клавишу . . .

Рис. 3.19. Корректное выполнение изменение ставки

## 3. Назначить премиальные сотруднику

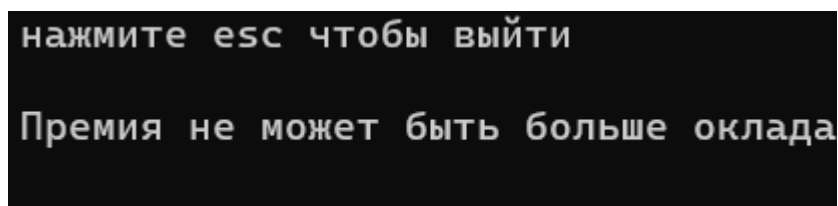
При запуске данной операции, пользователю необходимо выбрать сотрудника, а затем ввести сумму его премиальных. Значение премиальных в диапазоне от 100 до ЗП данного сотрудника.

Если пользователь попытается ввести сумму премиальных меньше 100 рублей, то сработает исключение (Рис. 3.20).



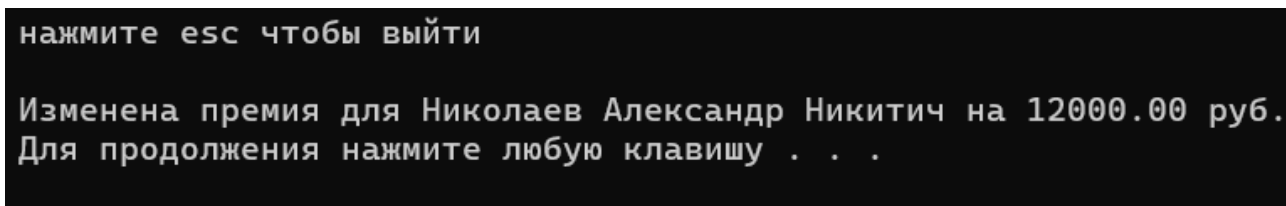
*Рис. 3.20. Выброс исключения при попытке ввода премиальных меньше 100 рублей*

Если пользователь попытается ввести сумму премиальных меньше 100 рублей, то сработает исключение (Рис. 3.21).



*Рис. 3.21. Выброс исключения при попытке ввода премиальных больше ЗП сотрудника*

Корректное выполнение данной операции (Рис. 3.22).

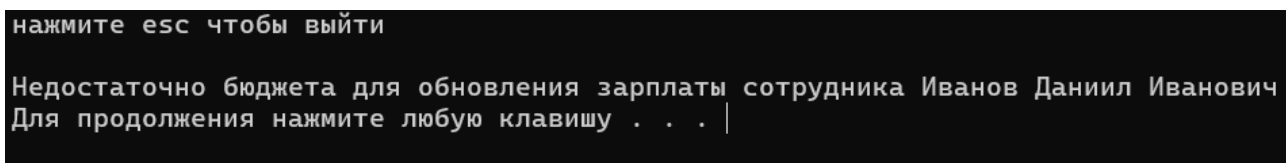


*Рис. 3.22. Корректное выполнение операции назначения премиальных*

#### 4. Изменить коэффициент ЗП должности

При запуске данной операции, пользователю необходимо выбрать тип должности, а затем ввести коэффициент, которые изменит базовую ЗП для данной должности. Значение коэффициента в диапазоне от 1 до 5.

Если пользователь попытается ввести коэффициент, для которого не хватит бюджета, то сработает исключение, и программа выведет данные сотрудника, для которого не хватило бюджета (Рис. 3.23).



*Рис. 3.23. Выброс исключения для изменения коэффициента*

нажмите esc чтобы выйти

Коэффициент для должности Бухгалтер успешно изменен на 3.00.  
Для продолжения нажмите любую клавишу . . . |

Рис. 3.24. Корректное выполнение операции “изменение коэффициента должности”

Корректное выполнение данной операции (Рис. 3.24).

#### 5. Вывести информацию обо всех работниках

Корректное выполнение данной операции (Рис. 3.25).

нажмите esc чтобы выйти									
№	Должность	Фамилия	Имя	Отчество	Зарплата	Ставка	Премии		
0	Директор	Васильев	Александр	Сергеевич	67347.00 руб.	1.00	0.00	руб.	
1	Бухгалтер	Васильев	Даниил	Алексиевич	38484.00 руб.	1.00	0.00	руб.	
2	Секретарь	Степанов	Андрей	Юрьевич	34635.60 руб.	1.00	0.00	руб.	
3	Водитель	Кузнецов	Алексей	Сергеевич	28863.00 руб.	1.00	0.00	руб.	
4	Водитель	Николаев	Дмитрий	Анатолевич	28863.00 руб.	1.00	0.00	руб.	
5	Охранник	Федоров	Даниил	Николаевич	23090.40 руб.	1.00	0.00	руб.	
6	Охранник	Иванов	Алексей	Николаевич	23090.40 руб.	1.00	0.00	руб.	
Для продолжения нажмите любую клавишу . . .									

Рис. 3.25. Корректное выполнение операции “Вывести информацию о всех работниках”

#### Сценарий №6. Управление секретарем

##### 1. Изменить ФИО

При выборе этого действия применяются постоянные ограничения, описанные выше.

Результат работы данной операции (Рис. 3.26).

нажмите esc чтобы выйти									
№	Должность	Фамилия	Имя	Отчество	Зарплата				
0	Секретарь	Неделько	Никита	Дмитриевич	34635.60 руб.				

Рис. 3.26. Корректное выполнение операции “Изменения ФИО” у секретаря

##### 2. Вывести сотрудников, данного директора

Корректное выполнение данной операции (Рис. 3.27).

нажмите есч чтобы выйти

№	Должность	Фамилия	Имя	Отчество	Зарплата
1	Бухгалтер	Васильев	Даниил	Алексиевич	38484.00 руб.
2	Секретарь	Неделько	Никита	Дмитриевич	34635.60 руб.
3	Водитель	Кузнецов	Алексей	Сергеевич	28863.00 руб.
4	Водитель	Николаев	Дмитрий	Анатолевич	28863.00 руб.
5	Охранник	Федоров	Даниил	Николаевич	23090.40 руб.
6	Охранник	Иванов	Алексей	Николаевич	23090.40 руб.

Для продолжения нажмите любую клавишу . . .

Рис. 3.27. Корректное выполнение операции “Вывести сотрудников, данного директора”

### 3. Вывести данные об охранниках

Корректное выполнение данной операции (Рис. 3.28).

нажмите есч чтобы выйти

№	Должность	Фамилия	Имя	Отчество	Зарплата
1	Охранник	Федоров	Даниил	Николаевич	23090.40 руб.
2	Охранник	Иванов	Алексей	Николаевич	23090.40 руб.

Для продолжения нажмите любую клавишу . . .

Рис. 3.28. Корректное выполнение операции “Вывести охранников, данного директора”

### 4. Вывести данные о водителях

Корректное выполнение данной операции (Рис. 3.29).

нажмите есч чтобы выйти

№	Должность	Фамилия	Имя	Отчество	Зарплата
1	Водитель	Кузнецов	Алексей	Сергеевич	28863.00 руб.
2	Водитель	Николаев	Дмитрий	Анатолевич	28863.00 руб.

Для продолжения нажмите любую клавишу . . .

Рис. 3.29. Корректное выполнение операции “Вывести водителей, данного директора”

## Сценарий №7. Управление водителем

### 1. Изменить ФИО

При выборе этого действия применяются постоянные ограничения, описанные выше.

Результат работы данной операции (Рис. 3.30).

нажмите есч чтобы выйти

№	Должность	Фамилия	Имя	Отчество	Зарплата
0	Водитель	Неделько	Никита	Дмитриевич	28863.00 руб.

Рис. 3.30. Корректное выполнение операции “Изменить ФИО” для водителя

## 2. Управление водительскими правами

При выборе данного действия, пользователю необходимо выбрать, какие права добавить (Рис. 3.31).

нажмите есч чтобы выйти

Категория Категория Е добавлена.  
Для продолжения нажмите любую клавишу . . . |

Рис. 3.31. Добавление водительских прав

Если пользователь повторно выберет категорию прав, которые уже есть, то произойдёт удаление этой категории прав (Рис. 3.32).

нажмите есч чтобы выйти

Категория Категория Е удалена.  
Для продолжения нажмите любую клавишу . . . |

Рис. 3.32. Удаление водительских прав

## 3. Управление транспортом

У пользователя нет возможности добавить транспорт, на который нет прав.

Пример корректного добавления транспортного средства (Рис. 3.33).

нажмите есч чтобы выйти

Транспортное средство Грузовик добавлено.  
Для продолжения нажмите любую клавишу . . . |

Рис. 3.33. Корректное добавление транспортного средства

#### 4. Отправить водителя в путешествие

Пример корректного добавления места назначения для водителя (Рис. 3.34).

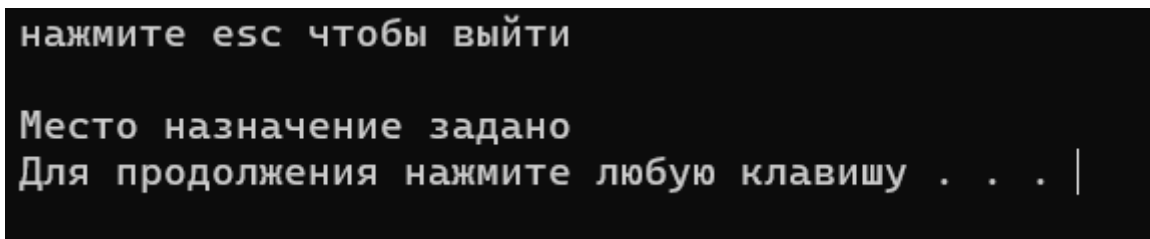


Рис. 3.34. Корректное добавление места назначения

#### Сценарий №8. Управление охранником

##### 1. Изменить ФИО

При выборе этого действия применяются постоянные ограничения, описанные выше.

Результат работы данной операции изображен (Рис. 3.35).

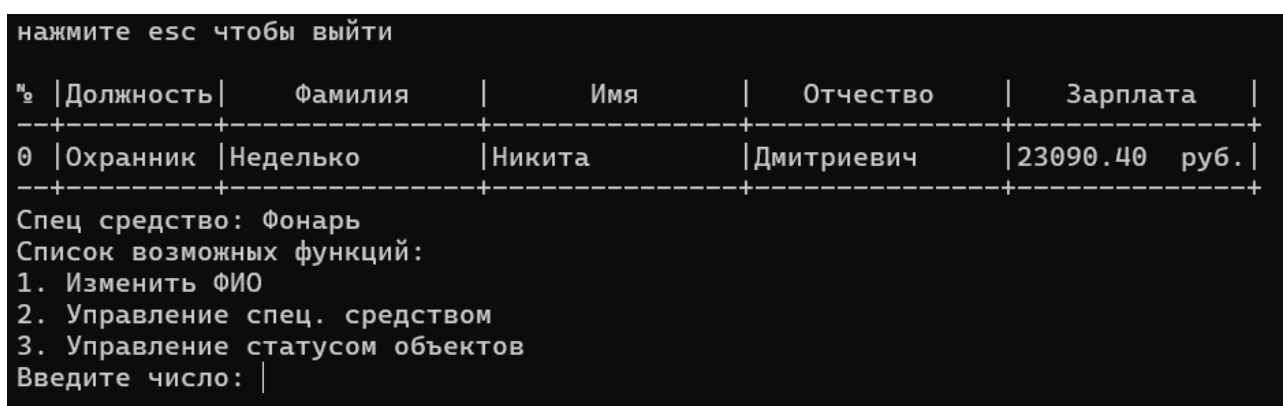


Рис. 3.35. Корректное выполнение операции “Изменить ФИО” для охранника

##### 2. Управление спец. средством

При выборе этого действия применяются постоянные ограничения, описанные выше.

Результат работы данной операции (Рис. 3.36).



```
нажмите esc чтобы выйти

№ | Должность |      Фамилия
--+-+-----+-----
0 | Охранник | Неделько
--+-+-----+-----
Спец средство: Дубинка
```

*Рис. 3.36. Корректное выполнение операции “добавление спец средств”*

## **ГЛАВА 4. Вывод**

В ходе выполнения курсовой работы была разработана информационная система для автоматизации управления компанией, включающей ключевые роли: директора, секретаря, бухгалтера, охранников и водителей. Программа успешно реализует функциональные возможности, отвечающие требованиям предметной области, включая управление персоналом, планирование задач и защиту от некорректного ввода данных пользователем.

Основные результаты работы:

- разработана программная структура, включающая классы для каждого типа сотрудников, с четко определенными методами и функционалом;
- реализован удобный пользовательский интерфейс, предоставляющий два режима создания компании: автоматическое чтение данных из файла и ручное заполнение;
- внедрены механизмы предотвращения ошибок ввода, что повысило надежность работы программы;
- проведено тестирование программы, подтвердившее её корректную работу во всех предусмотренных сценариях.

Программа отвечает поставленным требованиям, позволяет эффективно управлять компанией и предотвращает человеческие ошибки на этапе ввода данных. Таким образом, цели курсовой работы достигнуты, а задачи успешно выполнены.

## ПРИЛОЖЕНИЯ

### Приложение 1

Общее задание на курсовую работу ООАиП:

Реализовать иерархию классов в соответствии с вариантом. У всех вариантов в качестве базового класса выступает класс “Человек”. Построить диаграмму классов.

В случае недопустимых значений полей выбрасываются исключения.

Подготовить тестовый набор данных.

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

Бухгалтер или секретарь должен быть один.

Вариант задания - 15: 0 1 2 4 5

0. 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.

1. 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.

## 2. 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 “Водитель”, то реализовать метод, который принимает массив водителей и выводит их в таблицу.

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

## 4. Класс Охранник:

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

Класс должен содержать поле, которое хранит в себе название спец. средства (дубинка, шокер и т.д.)

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

## 5. 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.

### Листинг (код) класса Person:

```
class Person // базовый класс Человек
{
public:
    // свойства
    virtual void setFirstName(const string &firstName) = 0;    // Изменить имя
    virtual string getFirstName() const = 0;                    // Получить имя
    virtual void setLastName(const string &lastName) = 0;       // Изменить отчество
    virtual string getLastName() const = 0;                     // Получить отчество
    virtual void setMiddleName(const string &middleName) = 0;   // Изменить фамилию
    virtual string getMiddleName() const = 0;                   // Получить фамилию
    virtual void setSalary(const double salary) = 0;            // Изменить ЗП
    virtual double getSalary() const = 0;                       // Получить ЗП

    virtual ~Person() = default;
};
```

## Листинг (код) класса Director:

```
// класс Директор, наследуемый от Person
class Director : public Person
{
private:
    PersonalData personalData;

    SalaryDatabase salaryDatabase;           // Ссылка на базу зарплат
    map<Person *, Position> employees;        // Человек -> Должность
    map<Department, double> departmentBudgets; // Название отдела -> бюджет

public:
    Director()
    {
        setFirstName("NULL");
        setMiddleName("NULL");
        setLastName("NULL");
        setSalary(salaryDatabase.getMinimalSalary(Position::Director));
        approveBudget(Department::Bonuses, 20000);
        approveBudget(Department::Finance, 200000);
        approveBudget(Department::Management, 200000);
        approveBudget(Department::Security, 200000);
        approveBudget(Department::Transportation, 200000);
        approveBudget(Department::Executive, 100000 - getSalary());
    }

    Director(const PersonalData &personalData, SalaryDatabase &salaryDatabase) :
    salaryDatabase(salaryDatabase)
    {
        setFirstName(personalData.firstName);
        setMiddleName(personalData.middleName);
        setLastName(personalData.lastName);
        setSalary(salaryDatabase.getMinimalSalary(Position::Director));
        approveBudget(Department::Bonuses, 20000);
        approveBudget(Department::Finance, 200000);
        approveBudget(Department::Management, 200000);
        approveBudget(Department::Security, 200000);
        approveBudget(Department::Transportation, 200000);
        approveBudget(Department::Executive, 100000 - getSalary());
    }
}
```

```

}

// Геттеры и сеттеры для полей Director
void setFirstName(const string &firstName) override
{
    personalData.firstName = firstName;
}

string getFirstName() const override
{
    return personalData.firstName;
}

void setLastName(const string &lastName) override
{
    personalData.lastName = lastName;
}

string getLastName() const override
{
    return personalData.lastName;
}

void setMiddleName(const string &middleName) override
{
    personalData.middleName = middleName;
}

string getMiddleName() const override
{
    return personalData.middleName;
}

void setSalary(const double salary) override
{
    personalData.salary = salary;
}

double getSalary() const override
{
    return personalData.salary;
}

```

```

    }

    map<Person *, Position> getEmployees() const;           // список
сотрудников в подчинении -
    void approveBudget(const Department &department, double budget); // задать бюджет
на отдел -
    map<Department, double> &getApprovedBudget();           // получить бюджет
отдела -
    void hireEmployee(Person *employee, const Position &position); // нанять
работника -
    void terminateEmployee(Person *employee);               // уволить
работника -
    SalaryDatabase &getSalaryDataBase();                   // получение базы
данных зарплат -
    void displayInfo();                                     // вся информация
о классе -
    Person *findAccountant() const;                         // находим
бухгалтера среди работников -
    Person *findSecretary() const;                         // находим
секретаря среди работников -
};

map<Person *, Position> Director::getEmployees() const
{
    return employees;
}

void Director::approveBudget(const Department &department, double budget)
{
    departmentBudgets[department] = budget;
}

map<Department, double> &Director::getApprovedBudget()
{
    return departmentBudgets;
}

void Director::hireEmployee(Person *employee, const Position &position)
{
    double baseSalary = salaryDatabase.getMinimalSalary(position);
    double currentBudget = departmentBudgets[Department(int(position))];

```



```

// Проверяем бюджет: нужен бюджет хотя бы на минимальную ставку 0.1
if (currentBudget < baseSalary * 0.1)
    throw invalid_argument("Недостаточно бюджета для найма на минимальную ставку");

// установка ставки
double maxRate = currentBudget / baseSalary;
if (maxRate > 1.0)
    maxRate = 1.0;
if (maxRate < 0.1)
    throw invalid_argument("Бюджета недостаточно даже для минимальной ставки.");

employee->setSalary(baseSalary * maxRate);

// установка начальной ставки и обновление бюджета
employees[employee] = position;
departmentBudgets[Department(int(position))] -= baseSalary * maxRate;

cout << toStringEnum(position) << ' ' << employee->getMiddleName() << " нанят на
работу. "
    << "С зарплатой " << employee->getSalary() << endl;
}

void Director::terminateEmployee(Person *employee)
{
    // Проверяем, что сотрудник существует в списке
    auto it = employees.find(employee);
    if (it == employees.end())
        throw invalid_argument("Сотрудник не найден.");

    // Получаем позицию и зарплату сотрудника
    Position position = it->second;
    double salary = employee->getSalary();

    // Возвращаем зарплату в бюджет отдела
    departmentBudgets[Department(int(position))] += salary;

    cout << "Сотрудник " << employee->getMiddleName() << " уволен. Бюджет отдела " <<
toStringEnum(Department(int(position))) << " восстановлен на " << salary << "\n";

    // Удаляем сотрудника из списка

```

```

        employees.erase(it);
        delete employee;
    }

SalaryDatabase &Director::getSalaryDataBase()
{
    return salaryDatabase;
}

void Director::displayInfo()
{
    cout << left << setw(3) << "№ |"
        << setw(10) << "Должность|"
        << setw(45) << "    Фамилия    |    Имя    |    Отчество    |"
        << "    Зарплата    |" << endl;
    cout << string(2, '-') << '+' << string(9, '-') << '+' << string(15, '-') << '+' <<
    string(15, '-') << '+' << string(15, '-') << '+' << string(13, '-') << '+' << endl;

    cout << left << setw(2) << 0
        << setw(10) << "|" + toStringEnum(Position::Director)
        << setw(16) << "|" + getMiddleName()
        << setw(16) << "|" + getFirstName()
        << setw(16) << "|" + getLastName()
        << "|" << fixed << setprecision(2) << getSalary() << " руб.|" << endl;
    cout << string(2, '-') << '+' << string(9, '-') << '+' << string(15, '-') << '+' <<
    string(15, '-') << '+' << string(15, '-') << '+' << string(13, '-') << '+' << endl;

    cout << "\nБюджет отделов:\n";
    for (auto &department : getApprovedBudget())
    {
        cout << setw(20) << toStringEnum(department.first) // Название отдела
            << setw(7) << department.second                // Бюджет
            << " руб." << endl;
    }
}

Person *Director::findAccountant() const
{
    for (const auto &employee : employees)
        if (employee.second == Position::Accountant)
            return employee.first; // Возвращаем указатель на бухгалтера
}

```

```

        return nullptr;                // Бухгалтер не найден
    }

    Person *Director::findSecretary() const
    {
        for (const auto &employee : employees)
            if (employee.second == Position::Secretary)
                return employee.first; // Возвращаем указатель на секретаря
        return nullptr;                // секретарь не найден
    }

```

### Листинг (код) класса Accountant:

```
// класс Бухгалтер, наследуемый от Person
class Accountant : public Person
{
private:
    PersonalData personalData;

    SalaryDatabase salaryDatabase;          // Ссылка на базу зарплат
    map<Person *, double> employeesRate;    // Хранение ставки по каждому сотруднику
    map<Person *, double> employeesBonuses; // Хранение премиальных по каждому
    сотруднику

public:
    Accountant(const PersonalData &personalData, SalaryDatabase &salaryDatabase) :
    salaryDatabase(salaryDatabase)
    {
        setFirstName(personalData.firstName);
        setMiddleName(personalData.middleName);
        setLastName(personalData.lastName);
        setSalary(salaryDatabase.getMinimalSalary(Position::Accountant));
    }

    // Геттеры и сеттеры для полей Accountant
    void setFirstName(const string &firstName) override
    {
        personalData.firstName = firstName;
    }

    string getFirstName() const override
    {
        return personalData.firstName;
    }

    void setLastName(const string &lastName) override
    {
        personalData.lastName = lastName;
    }

    string getLastName() const override
```

```

{
    return personalData.lastName;
}

void setMiddleName(const string &middleName) override
{
    personalData.middleName = middleName;
}

string getMiddleName() const override
{
    return personalData.middleName;
}

void setSalary(double salary) override
{
    personalData.salary = salary;
}

double getSalary() const override
{
    return personalData.salary;
}

void applyBonuses(Person *employee, const double bonus, Director &director); //
изменение премии сотрудников
void setRate(Person *employee, double rate, Director &director); //
изменение ставки сотрудников
void updateEmployeesRate(Director &director); //
добавление ставок сотрудников в массив ставок
double getEmployeeRate(Person *employee) const; //
получение ставки сотрудника
double getEmployeeBonus(Person *employee) const; //
получение премии сотрудника
SalaryDatabase &getSalaryDataBase(); //
выводим базу данных зарплат
Position getEmployeeType(Person *employee); //
определяем что за работник относительно Person *
map<Person *, double> getEmployeesRates() const; // все
ставки по каждому сотруднику

```

```

        void displayInfo();
        информация о классе
    };

void Accountant::applyBonuses(Person *employee, double bonus, Director &director)
{
    if (bonus < 100 && bonus > 0)
        throw invalid_argument("Премия не может быть меньше 100 рублей");
    if (bonus > employee->getSalary())
        throw invalid_argument("Премия не может быть больше оклада");

    double currentBudgetBonuses = director.getApprovedBudget()[Department::Bonuses] +
    getEmployeeBonus(employee);

    if (bonus > currentBudgetBonuses)
        throw invalid_argument("Недостаточно бюджета для выдачи премий!");

    director.approveBudget(Department::Bonuses, currentBudgetBonuses - bonus); //
    изменение бюджета бонусов
    employeesBonuses[employee] = bonus;
    cout << "Изменена премия для " << employee->getMiddleName() << ' ' << employee->
    >getFirstName() << " " << employee->getLastName() << " на " << fixed << setprecision(2)
    << employeesBonuses[employee] << " руб." << endl;
}

void Accountant::updateEmployeesRate(Director &director)
{
    for (const auto &[employee, position] : director.getEmployees())
    {
        // Проверяем, есть ли уже ставка для сотрудника
        if (employeesRate.find(employee) != employeesRate.end())
            continue;
        setRate(employee, employee->getSalary() /
salaryDatabase.getMinimalSalary(position), director);
    }
}

void Accountant::setRate(Person *employee, double newRate, Director &director)
{
    // Проверка минимальной ставки
    if (newRate < 0.1)

```

```

        throw invalid_argument("Ставка не может быть ниже 0.1.");

    Position position = getEmployeeType(employee);

    // Получение базовой зарплаты и текущего бюджета отдела
    double baseSalary = salaryDatabase.getMinimalSalary(position);
    double currentBudget = director.getApprovedBudget()[Department(int(position))];

    // Проверка, что сотрудник существует в базе ставок
    auto it = employeesRate.find(employee);
    if (it != employeesRate.end()) // если сотрудник найден (старый)
    {
        double oldRate = it->second; // Текущая ставка сотрудника
        double oldSalary = baseSalary * oldRate;
        // Рассчитываем изменение бюджета
        double newSalary = baseSalary * newRate;
        double salaryDifference = newSalary - oldSalary;
        if (salaryDifference > 0 && currentBudget < salaryDifference)
            throw invalid_argument("Недостаточно бюджета для увеличения ставки. " +
to_string(salaryDifference - currentBudget) + " руб.\n");
        // Обновление ставки и бюджета отдела
        employeesRate[employee] = newRate;
        director.getApprovedBudget()[Department(int(position))] -= salaryDifference;
        employee->setSalary(newSalary);

        cout << "Ставка сотрудника " << employee->getMiddleName() << " изменена на " <<
newRate
            << ". Зарплата теперь " << employee->getSalary() << ".\n";
        return;
    }

    // Обновление ставки и бюджета отдела
    employeesRate[employee] = newRate;

    return;
}

double Accountant::getEmployeeRate(Person *employee) const
{
    return employeesRate.at(employee);
}

```

```

double Accountant::getEmployeeBonus(Person *employee) const
{
    auto it = employeesBonuses.find(employee);
    if (it != employeesBonuses.end())
    {
        return it->second;
    }
    return 0; // Возвращаем 0, если ключ не найден
}

map<Person *, double> Accountant::getEmployeesRates() const
{
    return employeesRate;
}

SalaryDatabase &Accountant::getSalaryDataBase()
{
    return salaryDatabase;
}

void Accountant::displayInfo()
{ // Заголовок таблицы
    cout << left << setw(3) << "№ |"
        << setw(10) << "Должность|"
        << setw(45) << "    Фамилия    |    Имя    |    Отчество    |"
        << "    Зарплата    |" << setw(6) << "Ставка|" << "    Премия    |" << endl;
    cout << string(2, '-') << '+' << string(9, '-') << '+' << string(15, '-') << '+'
        << string(15, '-') << '+' << string(15, '-') << '+' << string(14, '-')
        << '+' << string(6, '-') << '+' << string(14, '-') << '+' << endl;

    // Данные о бухгалтере
    cout << left << setw(2) << 0
        << setw(10) << "|" + toStringEnum(Position::Accountant)
        << setw(16) << "|" + getMiddleName()
        << setw(16) << "|" + getFirstName()
        << setw(16) << "|" + getLastName()
        << "|" << setw(9) << fixed << setprecision(2) << getSalary() << " руб."
        << setw(6) << fixed << setprecision(2) << getEmployeeRate(this) << "|"
        << setw(9) << fixed << setprecision(2) << getEmployeeBonus(this) << " руб."
    << endl;
}

```



```
cout << string(2, '-') << '+' << string(9, '-') << '+' << string(15, '-')  
    << '+' << string(15, '-') << '+' << string(15, '-') << '+'  
    << string(14, '-') << '+' << string(6, '-') << '+' << string(14, '-') << '+'  
<< endl;  
}
```

### Листинг (код) класса Secretary:

```
// класс Secretary, наследуемый от Person
class Secretary : public Person
{
private:
    PersonalData personalData;

    vector<Meeting> meetings; // Работник и Место_встречи -> Время_встречи

public:
    Secretary(const PersonalData &personalData)
    {
        setFirstName(personalData.firstName);
        setMiddleName(personalData.middleName);
        setLastName(personalData.lastName);
    }

    // Геттеры и сеттеры для полей Human
    void setFirstName(const string &firstName) override
    {
        personalData.firstName = firstName;
    }

    string getFirstName() const override
    {
        return personalData.firstName;
    }

    void setLastName(const string &lastName) override
    {
        personalData.lastName = lastName;
    }

    string getLastName() const override
    {
        return personalData.lastName;
    }

    void setMiddleName(const string &middleName) override
```

```

    {
        personalData.middleName = middleName;
    }

    string getMiddleName() const override
    {
        return personalData.middleName;
    }

    void setSalary(double salary) override
    {
        personalData.salary = salary;
    }

    double getSalary() const override
    {
        return personalData.salary;
    }

    // методы планирования встреч и управления данными о сотрудниках
    void scheduleMeeting(vector<Person *> employees, const string &place, const Date
&date); /// создание встречи
    vector<Meeting> &listScheduledMeetings();
    void displayMeetings() const; //
вывод всех встреч
    void displayEmployeeListInTable(const map<Person *, Position> &employees) const; //
вывод в таблицу
    void participantsOfMeeting(Meeting meeting) const; //
вывод информации одной встречи
    void displayInfo() const; //
вся информация о классе
};

void Secretary::displayEmployeeListInTable(const map<Person *, Position> &employees)
const
{
    int employeeNumber = 1;
    for (int i = 0; i <= 4; ++i)
    {
        for (const auto &pair : employees)
        {

```

```

        Person *employee = pair.first;
        string positionName = toStringEnum(pair.second);

        if (i == 1 && pair.second == Position::Accountant ||
            i == 2 && pair.second == Position::Secretary ||
            i == 3 && pair.second == Position::Driver ||
            i == 4 && pair.second == Position::SecurityGuard)
        {
            cout << fixed << setprecision(2);
            cout << left << setw(2) << employeeNumber++
                 << setw(10) << "|" + positionName
                 << setw(16) << "|" + employee->getMiddleName()
                 << setw(16) << "|" + employee->getFirstName()
                 << setw(16) << "|" + employee->getLastName()
                 << "|" << setw(9) << fixed << setprecision(2) << employee-
>getSalary() << " py6." << endl;
            cout << string(2, '-') << '+' << string(9, '-') << '+' << string(15, '-')
            << '+' << string(15, '-') << '+' << string(15, '-') << '+' << string(14, '-') << '+'
            << endl;
        }
    }
}

```

```

void Secretary::scheduleMeeting(vector<Person *> employees, const string &place, const
Date &date)
{
    // Проверка на количество встреч за текущий день
    int meetingsToday = 0;

    // Проверка на участие сотрудника в другой встрече в то же время
    for (const auto &meeting : meetings)
    {
        if (meeting.date == date)
        {
            meetingsToday++;
            if (meetingsToday == 5)
                throw("Нельзя запланировать больше 5 встреч за день\n");
            for (const auto &employee : employees)
            {

```

```

        if (find(meeting.employees.begin(), meeting.employees.end(), employee)
!= meeting.employees.end())
        {
            system("cls");
            cout << "Сотрудник " << employee->getMiddleName() << " " <<
employee->getFirstName() << " уже занят на встрече в это время.\n";
            system("pause");
            return;
        }
    }
}

// Если все условия выполнены, добавляем встречу
meetings.emplace_back(employees, place, date);
cout << "Встреча успешно запланирована.\n";
}

vector<Meeting> &Secretary::listScheduledMeetings()
{
    return meetings;
}

void Secretary::displayMeetings() const
{
    cout << "Запланированные встречи:\n\n";
    int index = 0;
    for (const auto &meeting : meetings)
    {
        cout << ++index << ") Место: " << meeting.place << ", Дата: " <<
meeting.date.toStringDate() << "\n";
        // cout << "Сотрудники: ";
        // for (const auto &emp : meeting.employees)
        // {
        //     cout << emp->getFirstName() << " " << emp->getMiddleName();
        // }
    }
}

void Secretary::participantsOfMeeting(Meeting meeting) const
{

```

```

        cout << "Место: " << meeting.place << ", Дата: " << meeting.date.toStringDate() <<
"\n";
        cout << "Сотрудники: \n";
        for (const auto &emp : meeting.employees)
        {
            cout << emp->getMiddleName() << " " << emp->getFirstName() << " " << emp-
>getLastName() << endl;
        }
        cout << "\n";
    }

void Secretary::displayInfo() const
{

    cout << left << setw(3) << "№ |"
        << setw(10) << "Должность|"
        << setw(45) << "    Фамилия    |        Имя        |    Отчество    |"
        << "    Зарплата    |" << endl;
    cout << string(2, '-') << '+' << string(9, '-') << '+' << string(15, '-') << '+' <<
string(15, '-') << '+' << string(15, '-') << '+' << string(14, '-') << '+' << endl;

    cout << left << setw(2) << 0
        << setw(10) << "|" + toStringEnum(Position::Secretary)
        << setw(16) << "|" + getMiddleName()
        << setw(16) << "|" + getFirstName()
        << setw(16) << "|" + getLastName()
        << "|" << setw(9) << fixed << setprecision(2) << getSalary() << " руб." <<
endl;
    cout << string(2, '-') << '+' << string(9, '-') << '+' << string(15, '-') << '+' <<
string(15, '-') << '+' << string(15, '-') << '+' << string(14, '-') << '+' << endl;
}

```

### Листинг (код) класса SecurityGuard:

```
// класс SecurityGuard, наследуемый от Person
class SecurityGuard : public Person
{
private:
    PersonalData personalData;

    map<string, string> locations; // Место -> Статус проверки
    Equipment specialEquipment;   // Название спецсредства (дубинка, шокер и т.д.)

    // Список всех объектов
    vector<string> departments = {"Склад", "Производство", "Администрация",
    "Лаборатория"};
    // Список возможных статусов
    vector<string> statuses = {"Всё в порядке", "Нужна проверка", "Нарушение
    безопасности", "Неизвестный статус"};

public:
    SecurityGuard(const PersonalData &personalData, const Equipment &equipment) :
    specialEquipment(equipment)
    {
        setFirstName(personalData.firstName);
        setMiddleName(personalData.middleName);
        setLastName(personalData.lastName);
        specialEquipment = Equipment::Flashlight;
        for (string dep : departments)
        {
            locations[dep] = "Нужна проверка";
        }
    }
    // Геттеры и сеттеры для полей Human
    void setFirstName(const string &firstName) override
    {
        personalData.firstName = firstName;
    }

    string getFirstName() const override
    {
        return personalData.firstName;
    }
}
```

```

    }

    void setLastName(const string &lastName) override
    {
        personalData.lastName = lastName;
    }

    string getLastName() const override
    {
        return personalData.lastName;
    }

    void setMiddleName(const string &middleName) override
    {
        personalData.middleName = middleName;
    }

    string getMiddleName() const override
    {
        return personalData.middleName;
    }

    void setSalary(double salary) override
    {
        personalData.salary = salary;
    }

    double getSalary() const override
    {
        return personalData.salary;
    }

    // методы проверки безопасности и составления отчетов об инцидентах
    void setSecurityStatus(string &department, string &status); // Метод для изменения
статуса безопасности
    void generateIncidentReport() const; // Метод для генерации
отчета о безопасности
    void setSpecialEquipment(const Equipment equipment);
    Equipment getSpecialEquipment() const;
    vector<string> allDepartments(); // все объекты
    vector<string> allStatuses(); // все статусы

```



```

        void displayInfo() const;           // вся информация о классе
};

vector<string> SecurityGuard::allDepartments()
{
    return departments;
}

vector<string> SecurityGuard::allStatuses()
{
    return statuses;
}

void SecurityGuard::setSecurityStatus(string &department, string &status)
{
    locations[department] = status;
}

void SecurityGuard::generateIncidentReport() const
{
    cout << "\nОтчет о безопасности:\n";
    cout << "Отдел\t\tСтатус\n";
    cout << "-----\n";
    int i = 0;
    for (const auto &location : locations)
    {
        cout << ++i << ") " << location.first << "\t" << location.second << endl;
    }
}

void SecurityGuard::setSpecialEquipment(const Equipment equipment)
{
    specialEquipment = equipment;
}

Equipment SecurityGuard::getSpecialEquipment() const
{
    return specialEquipment;
}

void SecurityGuard::displayInfo() const
{
    cout << left << setw(3) << "№ |"
        << setw(10) << "Должность|"
        << setw(45) << "    Фамилия    |    Имя    |    Отчество    |"
        << "    Зарплата    |" << endl;
}

```

```

    cout << string(2, '-') << '+' << string(9, '-') << '+' << string(15, '-') << '+' <<
string(15, '-') << '+' << string(15, '-') << '+' << string(14, '-') << '+' << endl;

    cout << left << setw(2) << 0
        << setw(10) << "|" + toStringEnum(Position::SecurityGuard)
        << setw(16) << "|" + getMiddleName()
        << setw(16) << "|" + getFirstName()
        << setw(16) << "|" + getLastName()
        << "|" << setw(9) << fixed << setprecision(2) << getSalary() << " py6." <<
endl;
    cout << string(2, '-') << '+' << string(9, '-') << '+' << string(15, '-') << '+' <<
string(15, '-') << '+' << string(15, '-') << '+' << string(14, '-') << '+' << endl;

    cout << "Спец средство: " << toStringEnum(getSpecialEquipment());
}

```

### Листинг (код) класса Driver:

```
// класс Driver, наследуемый от Person
class Driver : public Person
{
private:
    PersonalData personalData;

    string destination;
    vector<DriverLicense> licenseCategories; // Массив категорий прав (например, А, В,
C)
    vector<VehicleType> vehicles;           // Массив транспортных средств, которыми
управляет водитель
    vector<string> cities;

public:
    Driver(const PersonalData &personalData, DriverLicense licenseCategories,
VehicleType vehicles)
    {
        setFirstName(personalData.firstName);
        setMiddleName(personalData.middleName);
        setLastName(personalData.lastName);
        addLicenseCategory(licenseCategories);
        addVehicle(vehicles);
        destination = "Не задано";
        cities = {"Москва", "Санкт-Петербург", "Нижний Новгород", "Казань",
"Челябинск", "Омск", "Ростов-на-Дону", "Уфа", "Не задано"};
    }
    // Геттеры и сеттеры для полей Human
    void setFirstName(const string &firstName) override
    {
        personalData.firstName = firstName;
    }

    string getFirstName() const override
    {
        return personalData.firstName;
    }

    void setLastName(const string &lastName) override
```

```

{
    personalData.lastName = lastName;
}

string getLastName() const override
{
    return personalData.lastName;
}

void setMiddleName(const string &middleName) override
{
    personalData.middleName = middleName;
}

string getMiddleName() const override
{
    return personalData.middleName;
}

void setSalary(double salary) override
{
    personalData.salary = salary;
}

double getSalary() const override
{
    return personalData.salary;
}

void changeDestination(const string &newDestination); // новое место назначение
const vector<string> getCities(); // список городов
(доступных мест назначений)
void addLicenseCategory(const DriverLicense &category); // Метод для добавления
категории водительских прав
vector<DriverLicense> &getLicenseCategories(); // Метод для получения
категории водительских прав
void addVehicle(const VehicleType &vehicle); // Метод для добавления
транспортного средства
vector<VehicleType> &getVehicles(); // Метод вывода всех
транспортных средств

```

```

        void displayInfo() const;                                // отображение всей
информации класса
};

void Driver::addLicenseCategory(const DriverLicense &category)
{
    licenseCategories.push_back(category);
}
void Driver::addVehicle(const VehicleType &vehicle)
{
    vehicles.push_back(vehicle);
}
void Driver::changeDestination(const string &newDestination)
{
    destination = newDestination;
}
vector<DriverLicense> &Driver::getLicenseCategories()
{
    return licenseCategories;
}
vector<VehicleType> &Driver::getVehicles()
{
    return vehicles;
}
void Driver::displayInfo() const
{
    cout << left << setw(3) << "№ |"
        << setw(10) << "Должность|"
        << setw(45) << "    Фамилия    |    Имя    |    Отчество    |"
        << "    Зарплата    |" << endl;
    cout << string(2, '-') << '+' << string(9, '-') << '+' << string(15, '-') << '+' <<
string(15, '-') << '+' << string(15, '-') << '+' << string(14, '-') << '+' << endl;

    cout << left << setw(2) << 0
        << setw(10) << "|" + toStringEnum(Position::Driver)
        << setw(16) << "|" + getMiddleName()
        << setw(16) << "|" + getFirstName()
        << setw(16) << "|" + getLastName()
        << "|" << setw(9) << fixed << setprecision(2) << getSalary() << " py6.|" <<
endl;
}

```

```

        cout << string(2, '-') << '+' << string(9, '-') << '+' << string(15, '-') << '+' <<
string(15, '-') << '+' << string(15, '-') << '+' << string(14, '-') << '+' << endl;

        if (!licenseCategories.empty())
        {
            cout << "Имеющиеся категории прав : ";
            for (DriverLicense dl : licenseCategories)
            {
                cout << toStringEnum(dl) << " ";
            }
            cout << endl;
        }
        cout << endl;

        if (!vehicles.empty())
        {
            cout << "Имеющиеся транспортные средства : ";
            for (VehicleType vt : vehicles)
            {
                cout << toStringEnum(vt) << " ";
            }
            cout << endl;
        }
        cout << endl;

        cout << "Выехал в город - " << destination;
        cout << endl;
    }
    const vector<string> Driver::getCities()
    {
        return cities;
    }

    Position Accountant::getEmployeeType(Person *employee) // определяем что за работник
относительно Person *
    {
        if (dynamic_cast<Driver *>(employee))
            return Position::Driver;
        else if (dynamic_cast<SecurityGuard *>(employee))
            return Position::SecurityGuard;
        else if (dynamic_cast<Director *>(employee))

```

```
        return Position::Director;
    else if (dynamic_cast<Accountant *>(employee))
        return Position::Accountant;
    else if (dynamic_cast<Secretary *>(employee))
        return Position::Secretary;

    throw invalid_argument("Неизвестный тип работника");
}
```

Листинг (код) оставшихся классов (Menu, SalaryDatabase) и функций:

```
struct PersonalData // персональная информация
{
    string firstName;
    string lastName;
    string middleName;
    double salary;
};

struct Date // для хранения даты
{
    int day, month, year, isLeapyYear, daysInMonth;

    Date()
    {
        time_t t = time(nullptr);
        tm *now = localtime(&t);

        isLeapyYear = 0;
        daysInMonth = 31;

        this->day = now->tm_mday;
        this->month = now->tm_mon + 1;
        this->year = now->tm_year + 1900;
    };

    Date(int day, int month, int year)
    {
        isLeapyYear = 0;
        daysInMonth = 31;

        setYear(year);
        setMonth(month);
        setDay(day);
    }

    void setYear(const int &_year)
    {
        if (_year < 1970 || _year > 2025)
```



```

        throw invalid_argument("Неверно задан год \n");
    if (_year % 4 == 0 && _year % 100 != 0 || _year % 400 == 0 && _year % 100 == 0)
        isLeapyYear = 1;
    year = _year;
}

void setMonth(const int &_month)
{
    if (_month > 12 || _month < 1)
        throw invalid_argument("Неверно задан месяц");
    switch (_month)
    {
    case 2:
        daysInMonth = isLeapyYear ? 29 : 28;
        break;
    case 4:
    case 6:
    case 9:
    case 11:
        daysInMonth = 30;
        break;
    default:
        daysInMonth = 31;
        break;
    }
    month = _month;
}

void setDay(const int &_day)
{
    if (_day < 1 || _day > daysInMonth)
        throw invalid_argument("Неверно задан день");
    day = _day;
}

string toStringDate() const
{
    ostringstream oss;
    oss << setw(2) << setfill('0') << day << "."
        << setw(2) << setfill('0') << month << "."
        << year;
}

```

```

        return oss.str();
    }

    bool operator==(const Date &other) const
    {
        return day == other.day && month == other.month && year == other.year;
    }
};

struct Meeting // информация о встрече
{
    vector<Person *> employees;
    string place;
    Date date;

    Meeting(const vector<Person *> &emp, const string &p, const Date &d) :
    employees(emp), place(p), date(d) {}
};

struct Trip
{
    Date dateStart;    // Дата начала поездки
    string destination; // Место назначения
};

enum class Position // должность
{
    Driver,           // водитель
    Secretary,        // секретарь
    SecurityGuard,    // охранник
    Accountant,       // бухгалтер
    Director          // директор
};

enum class Department // названия отделов
{
    Transportation, // отдел транспорта    Driver
    Management,    // управление        Secretary
    Security,      // отдел безопасности SecurityGuard
    Finance,       // финансовый отдел  Accountant
    Executive,     // исполнительный отдел Director
};

```

```

        Bonuses          // бюджет на премии
};

enum class Equipment
{
    Baton,      // дубинка
    Taser,      // электрошокер
    Handcuffs,  // наручники
    Flashlight, // фонарик
    PepperSpray // перечный спрей
};

// Перечисление типов водительских прав
enum class DriverLicense
{
    CATEGORY_A,
    CATEGORY_B,
    CATEGORY_C,
    CATEGORY_D,
    CATEGORY_E,
    COUNT
};

// Перечисление типов транспорта
enum class VehicleType
{
    MOTORCYCLE,
    CAR,
    TRUCK,
    BUS,
    VAN,
    COUNT
};

string toStringEnum(Position position) // функция для получения строкового
представления должности
{
    switch (position)
    {
        case Position::Driver:
            return "Водитель";
    }
}

```

```

    case Position::Director:
        return "Директор";
    case Position::SecurityGuard:
        return "Охранник";
    case Position::Secretary:
        return "Секретарь";
    case Position::Accountant:
        return "Бухгалтер";
    default:
        return "Неизвестная должность"; // На всякий случай
    }
}

string toStringEnum(Equipment equipment) // Функция для получения строкового
представления оборудования
{
    switch (equipment)
    {
    case Equipment::Baton:
        return "Дубинка";
    case Equipment::Taser:
        return "Шокер";
    case Equipment::Handcuffs:
        return "Наручники";
    case Equipment::Flashlight:
        return "Фонарь";
    case Equipment::PepperSpray:
        return "Перцовый баллончик";
    default:
        return "Неизвестно";
    }
}

string toStringEnum(VehicleType vehicle) // Функция для конвертации типа транспорта в
строку
{
    switch (vehicle)
    {
    case VehicleType::MOTORCYCLE:
        return "Мотоцикл";
    case VehicleType::CAR:
        return "Легковой автомобиль";
    case VehicleType::TRUCK:

```

```

        return "Грузовик";
    case VehicleType::BUS:
        return "Автобус";
    case VehicleType::VAN:
        return "Фургон";
    default:
        return "Неизвестный тип транспорта";
    }
}

string toStringEnum(Department department) // Функция для получения строкового
представления отдела
{
    switch (department)
    {
    case Department::Transportation:
        return "Отдел транспорта";
    case Department::Management:
        return "Отдел управление";
    case Department::Security:
        return "Отдел безопасности";
    case Department::Finance:
        return "Финансовый отдел";
    case Department::Executive:
        return "Исполнит. отдел";
    case Department::Bonuses:
        return "Бюджет на премии";
    default:
        return "Неизвестный отдел";
    }
}

string toStringEnum(DriverLicense license) // функция для получения строкового
представления категории водительской лицензии
{
    switch (license)
    {
    case DriverLicense::CATEGORY_A:
        return "Категория А";
    case DriverLicense::CATEGORY_B:
        return "Категория В";
    case DriverLicense::CATEGORY_C:
        return "Категория С";
    }
}

```

```

        case DriverLicense::CATEGORY_D:
            return "Категория D";
        case DriverLicense::CATEGORY_E:
            return "Категория E";
        default:
            return "Неизвестная категория лицензии";
    }
}

// класс для работы с минимальной ЗП и назначения ЗП на разные должности
class SalaryDatabase
{
private:
    map<Position, double> salaryCoefficients; // Хранение коэффициентов по должностям
    double minimumWage;                      // Минимальный размер оплаты труда
public:
    SalaryDatabase() : minimumWage(19242)
    {
        // Инициализация коэффициентов для должностей
        salaryCoefficients[Position::Driver] = {1.5};
        salaryCoefficients[Position::Director] = {3.5};
        salaryCoefficients[Position::SecurityGuard] = {1.2};
        salaryCoefficients[Position::Secretary] = {1.8};
        salaryCoefficients[Position::Accountant] = {2.0};
    }
    SalaryDatabase(const double &minWage) : minimumWage(minWage)
    {
        // Инициализация коэффициентов для должностей
        salaryCoefficients[Position::Driver] = {1.5};
        salaryCoefficients[Position::Director] = {3.5};
        salaryCoefficients[Position::SecurityGuard] = {1.2};
        salaryCoefficients[Position::Secretary] = {1.8};
        salaryCoefficients[Position::Accountant] = {2.0};
    }

    // void setMinimumWage(double newMinWage);
    // изменения минимальной зарплаты
    // void updateSalaries(const map<Person *, Position> &employees);
    // обновление ЗП каждого сотрудника с учетом прошлой ставки
    void setCoefficient(const Position &position, const double &newCoefficient,
        Director &director); // назначения коэффициента зарплаты

```

```

        double getMinimalSalary(Position position) const;
// получения итоговой зарплаты для должности
        void printSalaryInfo() const;
// вывод информации по каждой должности
        map<Position, double> &getSalaryCoefficients();
// возвращает список коэффициентов по каждой должности
};

// void SalaryDatabase::setMinimumWage(double newMinWage)
// {
//     minimumWage = newMinWage;
// }

double SalaryDatabase::getMinimalSalary(Position position) const
{
    return minimumWage * salaryCoefficients.at(position);
}

void SalaryDatabase::printSalaryInfo() const
{
    cout << "MPOT = " << minimumWage << " руб.";
    for (const auto &entry : salaryCoefficients)
    {
        cout << "Должность: " << static_cast<int>(entry.first)
              << " | Коэффициент зарплаты: " << entry.second
              << " | Итоговая зарплата: " << getMinimalSalary(entry.first) << endl;
    }
}

map<Position, double> &SalaryDatabase::getSalaryCoefficients()
{
    return salaryCoefficients;
}

// -----MENU-----

class Menu
{
private:
    int choice;
    vector<Company> companies; // список всех компаний

```

```

public:
    // -----manualInput-----
    // определяем что за работник относительно Person *
    Position getEmployeeType(Person *employee) // получение типа сотрудника по объекту
    Person *
    {
        if (dynamic_cast<Driver *>(employee))
            return Position::Driver;
        else if (dynamic_cast<SecurityGuard *>(employee))
            return Position::SecurityGuard;
        else if (dynamic_cast<Director *>(employee))
            return Position::Director;
        else if (dynamic_cast<Accountant *>(employee))
            return Position::Accountant;
        else if (dynamic_cast<Secretary *>(employee))
            return Position::Secretary;

        throw invalid_argument("Неизвестный тип работника");
    }

    void displayAllEmployees(Director &director, vector<Person *> &employees) // вывод
    всех сотрудников в таблицу (должность, ФИО, ЗП)
    {
        int employeeNumber = 1;
        for (int i = 0; i <= 4; ++i)
        {
            for (const auto &pair : director.getEmployees())
            {
                Person *employee = pair.first;
                string positionName = toStringEnum(pair.second);

                if (i == 1 && pair.second == Position::Accountant ||
                    i == 2 && pair.second == Position::Secretary ||
                    i == 3 && pair.second == Position::Driver ||
                    i == 4 && pair.second == Position::SecurityGuard)
                {
                    cout << fixed << setprecision(2);
                    cout << left << setw(2) << employeeNumber++
                        << setw(10) << "|" + positionName
                        << setw(16) << "|" + employee->getMiddleName()

```



```

        << setw(16) << "|" + employee->getFirstName()
        << setw(16) << "|" + employee->getLastName()
        << "|" << setw(9) << fixed << setprecision(2) << employee-
>getSalary() << " py6." << endl;
        cout << string(2, '-') << '+' << string(9, '-') << '+' <<
string(15, '-') << '+' << string(15, '-') << '+' << string(15, '-') << '+' <<
string(14, '-') << '+' << endl;
        employees.push_back(employee);
    }
}
}

void displayAllEmployeesWithInt(Director &director, vector<int> &indexs) // вывод
определенного сотрудника в таблицу (должность, ФИО, ЗП)
{
    int employeeNumber = 1;
    for (int i = 0; i <= 4; ++i)
    {
        for (const auto &pair : director.getEmployees())
        {
            Person *employee = pair.first;
            string positionName = toStringEnum(pair.second);

            if (i == 1 && pair.second == Position::Accountant ||
                i == 2 && pair.second == Position::Secretary ||
                i == 3 && pair.second == Position::Driver ||
                i == 4 && pair.second == Position::SecurityGuard)
            {
                if (find(indexs.begin(), indexs.end(), employeeNumber) !=
indexs.end())
                {
                    employeeNumber++;
                    continue;
                }
                cout << fixed << setprecision(2);
                cout << left << setw(2) << employeeNumber++
                    << setw(10) << "|" + positionName
                    << setw(16) << "|" + employee->getMiddleName()
                    << setw(16) << "|" + employee->getFirstName()
                    << setw(16) << "|" + employee->getLastName()

```

```

        << "|" << setw(9) << fixed << setprecision(2) << employee-
>getSalary() << " py6." << endl;

        cout << string(2, '-') << '+' << string(9, '-') << '+' <<
string(15, '-') << '+' << string(15, '-') << '+' << string(15, '-') << '+' <<
string(14, '-') << '+' << endl;
    }
}
}

void displayAllEmployeesWithRatesAndBonuses(Director &director, Accountant
&accountant) // вывод всех сотрудников в таблицу (должность, ФИО, ЗП, ставка, премия)
{
    clearConsole();
    cout << left << setw(3) << "№ |"
        << setw(10) << "Должность|"
        << setw(45) << "    Фамилия    |    Имя    |    Отчество    |"
        << "    Зарплата    |" << setw(6) << "Ставка|" << setw(14) << "    Премии
|" << endl;
    cout << string(2, '-') << '+' << string(9, '-') << '+' << string(15, '-') <<
'+'
        << string(15, '-') << '+' << string(15, '-') << '+' << string(14, '-')
        << '+' << string(6, '-') << '+' << string(14, '-') << '+' << endl;
    // Вывод информации о директоре
    cout << left << setw(2) << 0
        << setw(10) << "|" + toStringEnum(Position::Director)
        << setw(16) << "|" + director.getMiddleName()
        << setw(16) << "|" + director.getFirstName()
        << setw(16) << "|" + director.getLastName()
        << "|" << setw(9) << fixed << setprecision(2) << director.getSalary() << "
py6."
        << setw(6) << "1.00 |"
        << setw(9) << fixed << setprecision(2) <<
accountant.getEmployeeBonus(&director) << " py6." << endl;
    cout << string(2, '-') << '+' << string(9, '-') << '+' << string(15, '-')
        << '+' << string(15, '-') << '+' << string(15, '-') << '+' << string(14,
'-')
        << '+' << string(6, '-') << '+' << string(14, '-') << '+' << endl;

    // Переменная для нумерации сотрудников
    int employeeNumber = 1;

```

```

for (int i = 0; i <= 4; ++i)
{
    for (const auto &pair : director.getEmployees())
    {
        Person *employee = pair.first;
        string positionName = toStringEnum(pair.second);

        // Вывод информации в зависимости от должности
        if (i == 1 && pair.second == Position::Accountant ||
            i == 2 && pair.second == Position::Secretary ||
            i == 3 && pair.second == Position::Driver ||
            i == 4 && pair.second == Position::SecurityGuard)
        {
            cout << fixed << setprecision(2);
            cout << left << setw(2) << employeeNumber++
                << setw(10) << "|" + positionName
                << setw(16) << "|" + employee->getMiddleName()
                << setw(16) << "|" + employee->getFirstName()
                << setw(16) << "|" + employee->getLastName()
                << "|" << setw(9) << fixed << setprecision(2) << employee-
>getSalary() << " py6.|"
                << setw(6) << fixed << setprecision(2) <<
accountant.getEmployeeRate(employee) << "|"
                << setw(9) << fixed << setprecision(2) <<
accountant.getEmployeeBonus(employee) << " py6.|" << endl;

            // Разделитель для строк
            cout << string(2, '-') << '+' << string(9, '-') << '+' <<
string(15, '-')
                << '+' << string(15, '-') << '+' << string(15, '-') << '+' <<
string(14, '-')
                << '+' << string(6, '-') << '+' << string(14, '-') << '+' <<
endl;
        }
    }
}

void displayAllEmployeesWithRates(Director &director, Accountant &accountant,
vector<Person *> &employees) // вывод всех сотрудников в таблицу (должность, ФИО, ЗП,
ставка)
{

```

```

// Переменная для нумерации сотрудников
int employeeNumber = 1;

for (int i = 0; i <= 4; ++i)
{
    for (const auto &pair : director.getEmployees())
    {
        Person *employee = pair.first;
        string positionName = toStringEnum(pair.second);

        // Вывод информации в зависимости от должности
        if (i == 1 && pair.second == Position::Accountant ||
            i == 2 && pair.second == Position::Secretary ||
            i == 3 && pair.second == Position::Driver ||
            i == 4 && pair.second == Position::SecurityGuard)
        {
            cout << fixed << setprecision(2);
            cout << left << setw(2) << employeeNumber++
                << setw(10) << "|" + positionName
                << setw(16) << "|" + employee->getMiddleName()
                << setw(16) << "|" + employee->getFirstName()
                << setw(16) << "|" + employee->getLastName()
                << "|" << setw(9) << fixed << setprecision(2) << employee-
>getSalary() << " руб.|"
                << setw(6) << fixed << setprecision(2) <<
accountant.getEmployeeRate(employee) << "|" << endl;

            // Разделитель для строк
            cout << string(2, '-') << '+' << string(9, '-') << '+' <<
string(15, '-') << '+'
                << string(15, '-') << '+' << string(15, '-') << '+' <<
string(14, '-')
                << '+' << string(6, '-') << '+' << endl;
            employees.push_back(employee);
        }
    }
}

void displayAllEmployeesWithBonuses(Director &director, Accountant &accountant,
vector<Person *> &employees) // вывод всех сотрудников в таблицу (должность, ФИО, ЗП,
премия)

```

```

{
    // Переменная для нумерации сотрудников
    int employeeNumber = 1;

    for (int i = 0; i <= 4; ++i)
    {
        for (const auto &pair : director.getEmployees())
        {
            Person *employee = pair.first;
            string positionName = toStringEnum(pair.second);

            // Вывод информации в зависимости от должности
            if (i == 1 && pair.second == Position::Accountant ||
                i == 2 && pair.second == Position::Secretary ||
                i == 3 && pair.second == Position::Driver ||
                i == 4 && pair.second == Position::SecurityGuard)
            {
                cout << fixed << setprecision(2);
                cout << left << setw(2) << employeeNumber++
                     << setw(10) << "|" + positionName
                     << setw(16) << "|" + employee->getMiddleName()
                     << setw(16) << "|" + employee->getFirstName()
                     << setw(16) << "|" + employee->getLastName()
                     << "|" << setw(9) << fixed << setprecision(2) << employee-
>getSalary() << " руб."
                     << "|" << setw(9) << fixed << setprecision(2) <<
accountant.getEmployeeBonus(employee) << " руб." << endl;

                // Разделитель для строк
                cout << string(2, '-') << '+' << string(9, '-') << '+' <<
string(15, '-') << '+'
                     << string(15, '-') << '+' << string(15, '-') << '+' <<
string(14, '-')
                     << '+' << string(14, '-') << '+' << endl;
                employees.push_back(employee);
            }
        }
    }
}

void displayALLPositionsWithKoeff(Accountant &accountant) // вывод всех должностей
их коэффициент и зарплату

```

```

{
    SalaryDatabase &salaryDatabase = accountant.getSalaryDataBase(); // Получаем
базу данных зарплат
    // Заголовок таблицы
    cout << left << setw(2) << "№ |"
        << setw(10) << "Должность|"
        << setw(11) << "Коэффициент|"
        << "    Зарплата    |" << endl;
    cout << string(2, '-') << '+' << string(9, '-') << '+' << string(11, '-') <<
'+' << string(14, '-') << '+' << endl;

    // Вывод данных
    const auto &coefficients = salaryDatabase.getSalaryCoefficients(); //
Предполагается, что метод возвращает данные по должностям

    int i = 1;
    for (const auto &[position, coefficient] : coefficients)
    {
        cout << left << setw(2) << i << "|"
            << setw(9) << toStringEnum(position) << "|"
            << setw(11) << fixed << setprecision(2) << coefficient << "|"
            << setw(9) << fixed << setprecision(2) <<
salaryDatabase.getMinimalSalary(position) << " руб.|" << endl;

        // Линия-разделитель после каждой строки данных
        cout << string(2, '-') << '+' << string(9, '-') << '+' << string(11, '-')
<< '+' << string(14, '-') << '+' << endl;
        ++i;
    }
}

vector<Company> getListOfCompanies() const // получение массива компаний
{
    return companies;
}

bool createCompanyManually(Company &company) // создание компании в ручную
{
    clearConsole();

    string companyName;
    cout << "Введите название компании: ";

```

```

    if (!inputString(companyName, 2, 15))
        return 0;
    clearConsole();

    string firstName;
    cout << "Введите имя директора: ";
    if (!inputString(firstName, 2, 15))
        return 0;
    clearConsole();

    string middleName;
    cout << "Введите фамилию директора: ";
    if (!inputString(middleName, 2, 15))
        return 0;
    clearConsole();

    string lastName;
    cout << "Введите отчество директора: ";
    if (!inputString(lastName, 2, 15))
        return 0;
    clearConsole();

    Director director;
    director.setFirstName(firstName);
    director.setLastName(lastName);
    director.setMiddleName(middleName);
    Company temp(companyName, director);
    company = temp;
    return 1;
}

bool changeNameCompany(Company &Company) // изменение названия компании
{
    clearConsole();

    string companyName;
    cout << "Введите новое название компании: ";
    if (!inputString(companyName, 2, 15))
        return 0;
    clearConsole();
    Company.setName(companyName);

```

```

        cout << "Новое имя компании " << "\"" << companyName << "\"" << " успешно
задано \n";
        return 0;
    }
    bool deleteCompany(Company &companyToRemove) // удаление компании
    {
        clearConsole();

        // Удаление компании из вектора `companies`
        for (auto it = companies.begin(); it != companies.end(); )
        {
            if (&(*it) == &companyToRemove) // Проверяем адрес, чтобы найти нужную
компанию
            {
                it = companies.erase(it); // Удаляем компанию и обновляем итератор
                cout << "Компания успешно удалена.\n";
                break;
            }
            else
            {
                ++it;
            }
        }

        system("pause");
        return 0;
    }

    bool setFIO(Person &person) // изменение ФМО сотрудника
    {
        clearConsole();
        string firstName;
        cout << "Введите имя: ";
        if (!inputString(firstName, 2, 15))
            return 0;
        clearConsole();

        string middleName;
        cout << "Введите фамилию: ";
        if (!inputString(middleName, 2, 15))
            return 0;
    }

```



```

clearConsole();

string lastName;
cout << "Введите отчество: ";
if (!inputString(lastName, 2, 15))
    return 0;
clearConsole();

person.setFirstName(firstName);
person.setLastName(lastName);
person.setMiddleName(middleName);
return 1;
}

bool hireEmployeeMenu(Director &director) // меню нанять на работу от лица
директора, с выбором должности нового рабочего, ввода ФИО (проверка что бухгалтер и
секретарь уже есть в компании и что бюджета отдела хватит на найм рабочего)
{
    clearConsole();
    Accountant *acc = dynamic_cast<Accountant *>(director.findAccountant());
    Secretary *sec = dynamic_cast<Secretary *>(director.findSecretary());
    // Список доступных должностей для найма
    vector<Position> availablePositions = {Position::Driver, Position::Secretary,
    Position::SecurityGuard, Position::Accountant};

    cout << "Выберите тип работника для найма:\n";
    for (size_t i = 0; i < availablePositions.size(); ++i)
        cout << i + 1 << ". " << toStringEnum(availablePositions[i]) << endl;

    if (!inputSingleInt(choice, 1, 4))
        return 0;

    Position selectedPosition = availablePositions[choice - 1];

    // Проверка на уникальные должности (бухгалтер и секретарь )
    if ((acc != nullptr && selectedPosition == Position::Accountant) || (sec !=
    nullptr && selectedPosition == Position::Secretary))
        throw invalid_argument("Такой работник уже нанят");

    // Создаем новый объект сотрудника в зависимости от выбранной позиции
    Person *newEmployee = nullptr;
    PersonalData pd;

```

```

switch (selectedPosition)
{
case Position::Driver:
{
    DriverLicense dl = generateRandomDriverLicense();
    newEmployee = new Driver(pd, dl, generateRandomVehicle(dl));
    break;
}
case Position::Secretary:
    newEmployee = new Secretary(pd);
    break;
case Position::SecurityGuard:
    newEmployee = new SecurityGuard(pd, generateRandomSpecialEquipment());
    break;
case Position::Accountant:
    newEmployee = new Accountant(pd, director.getSalaryDataBase());
    break;
default:
    cout << "Ошибка: Неверная должность.\n";
    return false;
}
if (!setFIO(*newEmployee))
    return 0;
try
{
    // Нанимаем сотрудника с выбранной позицией
    director.hireEmployee(newEmployee, selectedPosition);
    Accountant *acc = dynamic_cast<Accountant *>(director.findAccountant());
    if (acc != nullptr)
        acc->updateEmployeesRate(director);
    system("pause");
    return true;
}
catch (const invalid_argument &e)
{
    cout << "Ошибка при найме сотрудника: " << e.what() << endl;
    delete newEmployee; // Освобождаем память, если сотрудник не был нанят
    system("pause");
    return false;
}
}

```

```

bool terminateEmployeeMenu(Director &director) // меню выбора сотрудника на
увольнение (ЗП сотрудника возвращается в бюджет отдела)
{
    clearConsole();

    // Получаем всех работников из директора
    map<Person *, Position> employees = director.getEmployees();

    // Проверяем, есть ли сотрудники для увольнения
    if (employees.empty())
    {
        cout << "Нет сотрудников для увольнения.\n";
        system("pause");
        return false;
    }

    // Отображаем всех сотрудников с их должностями
    cout << "Список сотрудников:\n\n";

    vector<Person *> employeeList;

    cout << left << setw(3) << "№ |"
        << setw(10) << "Должность|"
        << setw(45) << "    Фамилия    |    Имя    |    Отчество    |"
        << "    Зарплата    |" << endl;
    cout << string(2, '-') << '+' << string(9, '-') << '+' << string(15, '-') <<
    '+' << string(15, '-') << '+' << string(15, '-') << '+' << string(13, '-') << '+' <<
    endl;

    displayAllEmployees(director, employeeList);

    // Получаем выбор пользователя
    cout
        << "Введите номер сотрудника, которого хотите уволить: ";
    int choice;
    if (!inputInt(choice, 1, employeeList.size()))
        return 0;

    // Выбор сотрудника и увольнение
    Person *employeeToFire = employeeList[choice - 1];
    try

```

```

    {
        clearConsole();
        director.terminateEmployee(employeeToFire); // Удаление сотрудника из map в
директоре
        system("pause");
        return true;
    }
    catch (const exception &e)
    {
        cout << "Ошибка при увольнении сотрудника: " << e.what() << endl;
        return false;
    }
}

```

bool createMeeting(Secretary &secretary, Director &director) // создание встреч.  
 Задается время, название, участники и место встречи

```

{
    clearConsole();
    // Ввод названия комнаты
    string roomName;
    cout << "Введите название комнаты для встречи: ";
    if (!inputString(roomName, 2, 5, 0, 1))
        return 0;
    clearConsole();

    // Ввод даты встречи
    int day, month, year;
    cout << "Введите год проведения встречи : ";
    if (!inputInt(year, 2024, 2025))
        return 0;

    Date meetingDate;
    meetingDate.setYear(year);
    clearConsole();

    cout << "Введите месяц проведения встречи : ";
    if (!inputInt(month, 1, 12))
        return 0;
    meetingDate.setMonth(month);
    clearConsole();
}

```

```

cout << "Введите день проведения встречи : ";
if (!inputInt(day, 1, 31))
    return 0;
meetingDate.setDay(day);
clearConsole();

// после выбора сотрудников опять выводим

vector<Person *> listEmployees;

vector<Person *> employees;

displayAllEmployees(director, employees);

vector<int> selectedEmployeeNumbers;

while (true)
{
    clearConsole();
    if (selectedEmployeeNumbers.size() == employees.size())
    {
        cout << "все сотрудники добавлены на встречу";
        system("pause");
        break;
    }
    cout << left << setw(3) << "№ |"
        << setw(10) << "Должность|"
        << setw(45) << "    Фамилия    |        Имя        |    Отчество    |"
        << "    Зарплата    |" << endl;
    cout << string(2, '-') << '+' << string(9, '-') << '+' << string(15, '-')
    << '+' << string(15, '-') << '+' << string(15, '-') << '+' << string(14, '-') << '+' <<
    endl;

    if (find(selectedEmployeeNumbers.begin(), selectedEmployeeNumbers.end(), 0)
    == selectedEmployeeNumbers.end())
    {
        cout << left << setw(2) << 0
            << setw(10) << "|" + toStringEnum(Position::Director)
            << setw(16) << "|" + director.getMiddleName()
            << setw(16) << "|" + director.getFirstName()
            << setw(16) << "|" + director.getLastName()

```

```

        << "|" << setw(9) << fixed << setprecision(2) <<
director.getSalary() << " py6.|" << endl;
        cout << string(2, '-') << '+' << string(9, '-') << '+' << string(15, '-')
'') << '+' << string(15, '-') << '+' << string(15, '-') << '+' << string(14, '-') << '+'
<< endl;
    }

    displayAllEmployeesWithInt(director, selectedEmployeeNumbers);

    cout << "Выберите сотрудника для назначения на встречу: ";

    if (!inputInt(choice, 0, director.getEmployees().size()))
        break;
    ;

    if (find(selectedEmployeeNumbers.begin(), selectedEmployeeNumbers.end(),
choice) != selectedEmployeeNumbers.end())
    {
        continue;
    }

    selectedEmployeeNumbers.push_back(choice);

    if (choice == 0)
    {
        listEmployees.push_back(&director);
    }
    else if (choice <= director.getEmployees().size())
    {
        listEmployees.push_back(employees[choice - 1]);
    }
    else
    {
        throw("Неверный индекс выбора персонала на встречу\n");
    }

    // clearConsole();

    // cout << "Хотите добавить еще сотрудника на встречу ?\n";
    // cout << "1. Да\n";
    // cout << "2. Нет\n\n";

```

```

        // if (!inputSingleInt(choice, 1, 2))
        //     return 0;

        // if (choice == 1)
        //     continue;
        // else
        // {
        //     break;
        // }
        // Создание встречи
    }
    clearConsole();
    if (!selectedEmployeeNumbers.empty())
    {
        secretary.scheduleMeeting(listEmployees, roomName, meetingDate);
    }
    return 0;
}

bool getParticipantOfMeeting(Secretary &secretary) // вывод всех созданных встреч
{
    vector<Meeting> listOfMeeting = secretary.listScheduledMeetings();
    clearConsole();
    if (listOfMeeting.empty())
    {
        cout << "Список встреч пуст\n";
        system("pause");
        return 0;
    }
    secretary.displayMeetings();
    cout << endl;
    if (!inputSingleInt(choice, 1, listOfMeeting.size()))
        return 0;
    clearConsole();

    int i = 1;
    for (Meeting meeting : listOfMeeting)
    {
        if (i == choice)
            secretary.participantsOfMeeting(meeting);
        i++;
    }
}

```

```

        system("pause");
        return 0;
    }
bool deleteMeeting(Secretary &secretary) // выбор встречи для удаление
{
    vector<Meeting> &listOfMeeting = secretary.listScheduledMeetings();
    clearConsole();
    if (listOfMeeting.empty())
    {
        cout << "Список встреч пуст\n";
        system("pause");
        return 0;
    }
    secretary.displayMeetings();
    cout << endl;
    if (!inputSingleInt(choice, 1, listOfMeeting.size()))
        return 0;
    clearConsole();

    int i = 1;
    for (Meeting meeting : listOfMeeting)
    {
        if (i == choice)
            listOfMeeting.erase(listOfMeeting.begin() + choice - 1);
        i++;
    }
    cout << "Встреча успешно удалена\n";
    system("pause");
    return 0;
}

// -----RANDOM-----
// Функция для генерации случайного имени
string generateRandomFirstName() const
{
    const vector<string> firstNames = {"Никита", "Алексей", "Андрей", "Александр",
    "Дмитрий", "Даниил"};
    return firstNames[rand() % firstNames.size()];
}

// Функция для генерации случайной фамилии
string generateRandomMiddleName() const
{

```



```

        const vector<string> middlenames = {"Иванов", "Смирнов ", "Кузнецов", "Попов",
"Васильев", "Петров", "Соколов", "Михайлов", "Новиков", "Федоров", "Морозов", "Волков",
"Алексеев", "Лебедев", "Семенов", "Егоров", "Павлов", "Козлов", "Степанов",
"Николаев"};

        return middlenames[rand() % middlenames.size()];
    }

    // Функция для генерации случайного отчества
    string generateRandomLastName() const
    {
        const vector<string> lastNames = {"Владимирович", "Александрович",
"Николаевич", "Сергеевич", "Викторович", "Анатольевич", "Иванович", "Юрьевич",
"Михайлович", "Алексеевич", "Петрович", "Станиславович", "Григорьевич", "Дмитриевич",
"Никитич", "Станиславович", "Алексиевич", "Евгеньевич", "Тимофеевич", "Матвеевич"};

        return lastNames[rand() % lastNames.size()];
    }

    // Функция для генерации случайного названия компании
    string generateRandomCompanyName() const
    {
        const vector<string> companyNames = {
            "ТехноЛаб", "ИнноваСофт", "ПроектГрупп", "ЭкспертСистем", "ДельтаТек",
            "ГлобалСервис", "ЭкоТек", "СуперМаркет", "АльфаКорп", "БетаМедиа"};

        return companyNames[rand() % companyNames.size()];
    }

    // Функция для генерации случайного специального оборудования
    Equipment generateRandomSpecialEquipment()
    {
        return static_cast<Equipment>(rand() % 5); // 5 - количество типов оборудования
    }

    // Функция для генерации случайного города
    string generateRandomCity()
    {
        const vector<string> cities = {"Москва", "Санкт-Петербург", "Новосибирск",
"Екатеринбург", "Нижний Новгород", "Казань", "Челябинск", "Омск", "Ростов-на-Дону",
"Уфа"};

        return cities[rand() % cities.size()];
    }

    // Функция для генерации случайной даты в диапазоне от сегодня до года вперед
    Date generateRandomDate()
    {
        Date today;

```

```

// случайное количество дней от 0 до 364
int randomDays = rand() % 365;

tm tmDate = {};
tmDate.tm_year = today.year - 1900;
tmDate.tm_mon = today.month - 1;
tmDate.tm_mday = today.day;

time_t currentTime = mktime(&tmDate);

currentTime += randomDays * 24 * 3600;

tm *randomTm = localtime(&currentTime);

return Date(randomTm->tm_mday, randomTm->tm_mon + 1, randomTm->tm_year + 1900);
}
// Функция для генерации случайной комнаты
string generateRandomRoom()
{
    vector<string> rooms = {"Зал для совещаний", "Конференц-зал", "Студия",
"Aудитория 12", "Лаунж зона", "Кабинета 4", "Выставочный зал", "Тренинговый центр",
"Зал для презентаций", "Зал для банкетов"};

    int randomIndex = rand() % rooms.size();
    return rooms[rand() % rooms.size()];
}
// Функция для генерации случайного типа прав для водителя
DriverLicense generateRandomDriverLicense()
{
    return static_cast<DriverLicense>(rand() %
static_cast<int>(DriverLicense::COUNT));
}
// Функция для генерации случайного транспорта в зависимости от типа прав
VehicleType generateRandomVehicle(const DriverLicense &license)
{
    switch (license)
    {
    case DriverLicense::CATEGORY_A:
        return VehicleType::MOTORCYCLE; // Для категории А
    case DriverLicense::CATEGORY_B:
        return VehicleType::CAR; // Для категории В
    }
}

```

```

case DriverLicense::CATEGORY_C:
    return VehicleType::TRUCK; // Для категории C
case DriverLicense::CATEGORY_D:
    return VehicleType::BUS; // Для категории D
case DriverLicense::CATEGORY_E:
    return VehicleType::VAN; // Для категории E
default:
    return VehicleType::CAR; // По умолчанию
}
}

// Функция для создания компании с рандомными значениями
Company createCompanyRandom()
{
    clearConsole();
    SalaryDatabase salaryDatabase(19242); // МРОТ по Москве
    PersonalData pd;
    DriverLicense dl;
    VehicleType vt;
    Equipment equipment;

    pd.firstName = generateRandomCompanyName();
    pd.firstName = generateRandomFirstName();
    pd.middleName = generateRandomMiddleName();
    pd.lastName = generateRandomLastName();
    Director director(pd, salaryDatabase); // Директор

    pd.firstName = generateRandomCompanyName();
    pd.firstName = generateRandomFirstName();
    pd.middleName = generateRandomMiddleName();
    pd.lastName = generateRandomLastName();
    Accountant *accountant = new Accountant(pd, salaryDatabase); // Бухгалтер
    director.hireEmployee(accountant, Position::Accountant);

    pd.firstName = generateRandomCompanyName();
    pd.firstName = generateRandomFirstName();
    pd.middleName = generateRandomMiddleName();
    pd.lastName = generateRandomLastName();
    Secretary *secretary = new Secretary(pd); // Секретарь
    // secretary-
    >setSalary(director.getSalaryDataBase().getMinimalSalary(Position::Secretary));
}

```

```

director.hireEmployee(secretary, Position::Secretary);

pd.firstName = generateRandomCompanyName();
pd.firstName = generateRandomFirstName();
pd.middleName = generateRandomMiddleName();
pd.lastName = generateRandomLastName();
dl = generateRandomDriverLicense();
vt = generateRandomVehicle(dl);
Driver *driver = new Driver(pd, dl, vt); // Водитель
// driver-
>setSalary(director.getSalaryDataBase().getMinimalSalary(Position::Driver));
director.hireEmployee(driver, Position::Driver);

pd.firstName = generateRandomCompanyName();
pd.firstName = generateRandomFirstName();
pd.middleName = generateRandomMiddleName();
pd.lastName = generateRandomLastName();
dl = generateRandomDriverLicense();
vt = generateRandomVehicle(dl);
Driver *driver2 = new Driver(pd, dl, vt); // Водитель
// driver2-
>setSalary(director.getSalaryDataBase().getMinimalSalary(Position::Driver));
director.hireEmployee(driver2, Position::Driver);

pd.firstName = generateRandomCompanyName();
pd.firstName = generateRandomFirstName();
pd.middleName = generateRandomMiddleName();
pd.lastName = generateRandomLastName();
equipment = generateRandomSpecialEquipment();
SecurityGuard *securityGuard = new SecurityGuard(pd, equipment); // Охранник
// securityGuard-
>setSalary(director.getSalaryDataBase().getMinimalSalary(Position::SecurityGuard));
director.hireEmployee(securityGuard, Position::SecurityGuard);

pd.firstName = generateRandomCompanyName();
pd.firstName = generateRandomFirstName();
pd.middleName = generateRandomMiddleName();
pd.lastName = generateRandomLastName();
equipment = generateRandomSpecialEquipment();
SecurityGuard *securityGuard2 = new SecurityGuard(pd, equipment); // Охранник

```

```

        // securityGuard2-
>setSalary(director.getSalaryDataBase().getMinimalSalary(Position::SecurityGuard));
    director.hireEmployee(securityGuard2, Position::SecurityGuard);

    // создание встреч
    vector<Person *> emp;
    emp.push_back(securityGuard2);
    emp.push_back(securityGuard);
    emp.push_back(secretary);
    vector<Person *> emp2;
    emp2.push_back(accountant);
    emp2.push_back(driver2);
    emp2.push_back(driver);
    secretary->scheduleMeeting(emp, generateRandomRoom(), generateRandomDate());
    secretary->scheduleMeeting(emp2, generateRandomRoom(), generateRandomDate());

    accountant->updateEmployeesRate(director);
    return Company(generateRandomCompanyName(), director);
}

// -----UI-----
bool inputSingleInt(int &digit, const int start, const int end);
// ввод положительной цифры
bool inputString(string &str, const int &minLen = 2, const int &maxLen = 15, bool
firstBigLetter = 1, bool allowNumbers = 0); // ввод строки
bool inputInt(int &number, const int &min = INT_MIN / 10, const int &max = INT_MAX
/ 10,
                const string &key = "NULL", const bool minus = 0, const bool point =
0); // ввод числа int
bool inputDouble(double &number, const double &min = __DBL_MIN__ / 10, const double
&max = __DBL_MAX__ / 10,
                const string &key = "NULL", const bool minus = 0, const bool point
= 1, int decimalLimit = 2); // ввод double
void clearConsole();
// очистка консоли и вывод сообщения "нажмите esc чтобы выйти"

// -----MENU-----
// меню создания компании
bool createCompanyMenu()
{
    clearConsole();

```

```

    if (companies.size() >= 9)
        throw invalid_argument("Максимальное количество компаний");

    cout << "Меню создания компании" << '\n';
    cout << "1. Создать компанию вручную\n";
    cout << "2. Создать компанию с случайными значениями\n";

    if (!inputSingleInt(choice, 1, 2))
        return 0;

    Company company;

    switch (choice)
    {
    case 1:
        if (!createCompanyManually(company))
            return 1;
        companies.emplace_back(company);
        break;
    case 2:
        try
        {
            companies.emplace_back(createCompanyRandom());
        }
        catch (const invalid_argument &e)
        {
            cout << "ОШИБКА: " << e.what() << endl;
        }
        break;
    default:
        return 1;
    }

    cout << "Компания создана успешно:\n";
    companies[companies.size() - 1].displayInfo();
    return 0;
}

// меню выбора создать или управлять компаниями
bool CompanyMenu()
{

```

```

clearConsole();

cout << "Выберите действие:\n";
cout << "1. Создать новую компанию\n";
cout << "2. Управление компаниями\n";

if (!inputSingleInt(choice, 1, 2))
    return 0;

switch (choice)
{
case 1:
    try
    {
        while (createCompanyMenu())
            ;
    }
    catch (const invalid_argument &e)
    {
        system("cls");
        cerr << "ОШИБКА: " << e.what() << '\n';
        system("pause");
    }
    break;
case 2:
    while (manageCompaniesMenu())
        ;
    break;
default:
    return 1;
}
return 1;
}

// меню выбора компании для управления
bool manageCompaniesMenu()
{
    clearConsole();

    if (companies.empty()) // Список компаний пуст
    {
        cout << "Список компаний пуст \n";
    }
}

```

```

        system("pause");
        return 0;
    }

    int i = 0;

    for (Company company : companies) // Вывод компаний
    {
        i++;
        cout << i << ") ";
        company.displayInfo();
        cout << endl;
    }

    cout << "Какой компанией вы хотите управлять ?\n";

    if (!inputSingleInt(choice, 1, companies.size()))
        return 0;

    try
    {
        int numOfCompany = choice - 1;
        while (workWithCompany(companies[numOfCompany]))
            ;
        return 1;
    }
    catch (const invalid_argument &e)
    {
        system("cls");
        cerr << "ОШИБКА: " << e.what() << '\n';
        system("pause");
        return 1;
    }
}

// меню управления одной компанией
bool workWithCompany(Company &company)
{
    clearConsole();
    company.displayInfo();
    cout << endl;
    cout << "Список возможных функций:" << '\n'

```



```

        << "1. Изменить имя компании" << '\n'
        << "2. Перейти к управлению работниками компании" << '\n'
        << "3. Удалить компанию" << '\n';

if (!inputSingleInt(choice, 1, 3))
    return 0;

switch (choice)
{
case 1: // изменение имени файла

    while (changeNameCompany(company))
        ;
    system("pause");
    return 1;
case 2: // Перейти к управлению работниками компании
    while (manageEmployeesMenu(company.getDirector()))
        ;
    return 1;
case 3: // удалить компанию
    while (deleteCompany(company))
        ;
    return 0;
default:
    return 1;
}
return 1;
}

// управление сотрудниками компании
bool manageEmployeesMenu(Director &director)
{
    clearConsole();
    cout << "Сотрудники данной компании: \n\n";

    cout << left << setw(3) << "№ |"
        << setw(10) << "Должность|"
        << setw(45) << "    Фамилия    |    Имя    |    Отчество    |"
        << "    Зарплата    |" << endl;

    cout << string(2, '-') << '+' << string(9, '-') << '+' << string(15, '-') <<
    '+' << string(15, '-') << '+' << string(15, '-') << '+' << string(14, '-') << '+' <<
    endl;

```

```

        cout << left << setw(2) << 0
            << setw(10) << "|" + toStringEnum(Position::Director)
            << setw(16) << "|" + director.getMiddleName()
            << setw(16) << "|" + director.getFirstName()
            << setw(16) << "|" + director.getLastName()
            << "|" << setw(9) << fixed << setprecision(2) << director.getSalary() << "
py6.|" << endl;

        cout << string(2, '-') << '+' << string(9, '-') << '+' << string(15, '-') <<
        '+' << string(15, '-') << '+' << string(15, '-') << '+' << string(14, '-') << '+' <<
        endl;

        vector<Person *> employeeList;
        displayAllEmployees(director, employeeList);

        cout << "\nВыберите сотрудника: ";
        if (!inputInt(choice, 0, director.getEmployees().size()))
            return 0;

        if (choice == 0)
        {
            while (manageEmployeeMenu(director))
                ;
            return 1;
        }

        Person *employee = employeeList[choice - 1];
        Position pos = getEmployeeType(employee);
        try
        {
            if (pos == Position::Accountant)
                while (manageEmployeeMenu(dynamic_cast<Accountant *>(employee),
director))
                    ;
            if (pos == Position::Secretary)
                while (manageEmployeeMenu(dynamic_cast<Secretary *>(employee),
director))
                    ;
            if (pos == Position::Driver)
                while (manageEmployeeMenu(dynamic_cast<Driver *>(employee)))
                    ;

```

```

        if (pos == Position::SecurityGuard)
            while (manageEmployeeMenu(dynamic_cast<SecurityGuard *>(employee)))
                ;
    }
    catch (const invalid_argument &e)
    {
        system("cls");
        cout << e.what();
        system("pause");
    }
    return 1;
}

// управление бюджетом отделов
bool manageBudgetDepartmentsMenu(Director &director)
{
    clearConsole();
    // Получаем список отделов и их текущие бюджеты
    map<Department, double> departmentBudgets = director.getApprovedBudget();

    // Список для выбора отдела
    cout << "Бюджет отделов:\n";
    int i = 1;
    vector<Department> departments;
    for (const auto &entry : departmentBudgets)
    {
        cout << i++ << ". " << toStringEnum(entry.first) << " \t(Бюджет: " <<
entry.second << " руб.)\n";
        departments.push_back(entry.first); // Сохраняем отдел для выбора
    }

    if (!inputSingleInt(choice, 1, departments.size()))
        return 0;

    Department selectedDepartment = departments[choice - 1];

    // Изменение бюджета выбранного отдела
    clearConsole();
    double newBudget;
    cout << "Введите новый бюджет для " << toStringEnum(selectedDepartment) << ":
";

    if (!inputDouble(newBudget, 0, 500000, "New Bonus", false, true))

```

```

        return false;
    clearConsole();
    cout << "Прошлый бюджет " << toStringEnum(selectedDepartment) << " = " <<
departmentBudgets[selectedDepartment] << " руб." << endl;
    director.approveBudget(selectedDepartment, newBudget);
    cout << "Нынешний бюджет " << toStringEnum(selectedDepartment) << " успешно
изменен на " << newBudget << " руб.\n";
    system("pause");
    return 1;
}
// изменение ставок сотрудников
bool setEmployeesRatesMenu(Accountant &accountant, Director &director)
{
    clearConsole();
    cout << "Список сотрудников и их текущая ставка:\n\n";

    vector<Person *> employees;

    cout << left << setw(3) << "№ |"
        << setw(10) << "Должность|"
        << setw(45) << "    Фамилия    |    Имя    |    Отчество    |"
        << "    Зарплата    |" << setw(6) << "Ставка|" << endl;
    cout << string(2, '-') << '+' << string(9, '-') << '+' << string(15, '-') <<
'+'
        << string(15, '-') << '+' << string(15, '-') << '+' << string(14, '-')
        << '+' << string(6, '-') << '+' << endl;

    displayAllEmployeesWithRates(director, accountant, employees);

    // Запрашиваем выбор сотрудника для изменения ставки
    cout << "\nВыберите номер сотрудника для изменения ставки: ";
    if (!inputInt(choice, 1, employees.size()))
        return 0;

    // Определяем выбранного сотрудника
    Person *selectedEmployee = nullptr;
    int index = 1;
    for (const auto &employee : employees)
    {
        if (index++ == choice)
        {

```

```

        selectedEmployee = employee;
        break;
    }
}
clearConsole();
// Ввод новой ставки для выбранного сотрудника
cout << "Введите новую ставку для " << selectedEmployee->getMiddleName() << " "
    << selectedEmployee->getFirstName() << ": ";
double newRate;
if (!inputDouble(newRate, 0.001, 1.0)) // Например, диапазон ставок от 0.0 до
1.0
    return 0;

clearConsole();
// Обновление ставки сотрудника через бухгалтера
accountant.setRate(selectedEmployee, newRate, director);
system("pause");
return true;
}
// изменение ставок сотрудников
bool setEmployeesBonusesMenu(Accountant &accountant, Director &director)
{
    clearConsole();
    cout << "Сотрудники и их премии:\n\n";

    cout << left << setw(3) << "№ |"
        << setw(10) << "Должность|"
        << setw(45) << "    Фамилия    |    Имя    |    Отчество    |"
        << "    Зарплата    |" << "    Премии    |" << endl;
    cout << string(2, '-') << '+' << string(9, '-') << '+' << string(15, '-') <<
'+'
        << string(15, '-') << '+' << string(15, '-') << '+' << string(14, '-')
        << '+' << string(14, '-') << '+' << endl;

    // Выводим список сотрудников и их премий
    vector<Person *> currentEmployees;

    displayAllEmployeesWithBonuses(director, accountant, currentEmployees);

    cout << "Бюджет выделенный на премии = " << fixed << setprecision(2) <<
    director.getApprovedBudget().at(Department::Bonuses) << " руб. \n";
}

```

```

        cout << "\nВыберите сотрудника, которому нужно изменить премию (введите номер):
";

        if (!inputInt(choice, 1, currentEmployees.size()))
            return 0;

        clearConsole();

        // Получаем выбранного сотрудника
        auto selectedEmployee = currentEmployees[choice - 1]; // Получаем указатель на
выбранного сотрудника

        double newBonus;
        cout << "Введите новую премию для работника " << selectedEmployee-
>getMiddleName() << " " << selectedEmployee->getFirstName() << ": ";
        if (!inputDouble(newBonus, 0, director.getApprovedBudget()[Department::Bonuses]
+ accountant.getEmployeeBonus(selectedEmployee), "New Bonus", false, true))
            return false;

        // Устанавливаем новую премию
        clearConsole();
        try
        {
            accountant.applyBonuses(selectedEmployee, newBonus, director);
        }
        catch (const invalid_argument &e)
        {
            cout << e.what();
        }
        system("pause");
        return true;
    }

    // изменение коэффициента должности
    bool setCoefficientMenu(Accountant &accountant, Director &director)
    {
        clearConsole();
        SalaryDatabase &salaryDatabase = accountant.getSalaryDataBase(); // Получаем
базу данных зарплат

        cout << "Список должностей, их коэффициентов и ЗП:\n\n";
        const auto &coefficients = salaryDatabase.getSalaryCoefficients();

```

```

displayALLPositionsWithKoeff(accountant);

cout << "\nБюджет отделов:\n";
for (auto &department : director.getApprovedBudget())
{
    cout << setw(20) << toStringEnum(department.first) // Название отдела
        << setw(7) << department.second                // Бюджет
        << " руб." << endl;
}

cout << "\nВыберите должность для изменения коэффициента\n";
if (!inputSingleInt(choice, 1, coefficients.size()))
    return false;

// Получаем выбранную должность
Position selectedPosition = next(coefficients.begin(), choice - 1)->first;

clearConsole();

cout << "Введите новый коэффициент для должности " <<
toStringEnum(selectedPosition) << ": ";
double newCoefficient;
if (!inputDouble(newCoefficient, 0.1, 5, "null", false, true)) // Пример
диапазона от 0.1 до 10
    return false;

// Устанавливаем новый коэффициент через метод SalaryDatabase
try
{
    clearConsole();
    salaryDatabase.setCoefficient(selectedPosition, newCoefficient, director);
    cout << "Коэффициент для должности " << toStringEnum(selectedPosition) << "
успешно изменен на "
        << newCoefficient << ".\n";
}
catch (const invalid_argument &e)
{
    cout << e.what();
}
system("pause");

```

```

        return true;
    }
    // меню управления встречами
    bool meetingManageMenu(Secretary &secretary, Director &director)
    {
        clearConsole();
        secretary.displayMeetings();
        cout << endl;
        cout << "Список возможных функций:" << '\n'
            << "1. Создать встречу" << '\n'
            << "2. Вывести участников встречи" << '\n'
            << "3. Удалить встречу" << '\n';

        if (!inputSingleInt(choice, 1, 3))
            return 0;

        switch (choice)
        {
        case 1: // Создать встречу

            while (createMeeting(secretary, director))
                ;
            system("pause");
            return 1;
        case 2: // Вывести участников встречи
            while (getParticipantOfMeeting(secretary))
                ;
            return 1;
        case 3: // Удалить встречу
            while (deleteMeeting(secretary))
                ;
            return 1;
        default:
            return 1;
        }
        return 1;
    }
    // меню изменения статуса отделов
    bool manageStatusLocationsMenu(SecurityGuard &securityGuard)
    {
        clearConsole();

```



```

// Список всех объектов
vector<string> departments = securityGuard.allDepartments();
// Список возможных статусов
vector<string> statuses = securityGuard.allStatuses();

cout << "Выберите отдел для проверки статуса:\n";
securityGuard.generateIncidentReport();
if (!inputSingleInt(choice, 1, securityGuard.allDepartments().size()))
    return 0;

string selectedDepartment = departments[choice - 1];
clearConsole();
// Выбор статуса
cout << "\nВыберите статус для отдела " << selectedDepartment << ":\n";
for (size_t i = 0; i < statuses.size(); ++i)
{
    cout << i + 1 << ". " << statuses[i] << endl;
}
if (!inputSingleInt(choice, 1, securityGuard.allStatuses().size()))
    return 0;

// Сохранение статуса и комментария для выбранного отдела
securityGuard.setSecurityStatus(selectedDepartment, statuses[choice - 1]);
cout << "Статус для отдела " << selectedDepartment << " успешно обновлен.\n";
system("pause");
return 1;
}

// меню изменения спец средств
bool manageSpecialEquipmentMenu(SecurityGuard &securityGuard)
{
    clearConsole();

    cout << "Выберите, какое спец. средство дать охраннику:\n";
    for (int i = 0; i < int(Equipment::PepperSpray); i++)
    {
        cout << i + 1 << ") " << toStringEnum(Equipment(i)) << '\n';
    }
    if (!inputSingleInt(choice, 1, 4))
        return 0;
    securityGuard.setSpecialEquipment(Equipment(choice - 1));
    return 0;
}

```

```

}
// меню выбора места, куда поедет водитель
bool manageDriverDestinationMenu(Driver &driver)
{
    clearConsole();
    cout << "Выберите город, куда отправится водитель:\n";
    int i = 1;
    for (string dest : driver.getCities())
    {
        cout << i++ << " ) " << dest << endl;
    }

    if (!inputSingleInt(choice, 1, 9))
        return 0;

    driver.changeDestination(driver.getCities()[choice - 1]);

    clearConsole();
    cout << "Место назначение задано\n";
    system("pause");
    return 1;
}
// меню добавления/удаление водительских прав
bool manageDriverLicenseCategoriesMenu(Driver &driver)
{
    clearConsole();
    cout << "Выберите категории прав для добавления (введите номер категории):\n";
    for (size_t i = 0; i < int(DriverLicense::COUNT); ++i)
    {
        cout << i + 1 << " ) " << toStringEnum(DriverLicense(i)) << endl;
    }

    if (!inputSingleInt(choice, 1, 5))
        return 0;

    vector<DriverLicense> &licenseCategories = driver.getLicenseCategories();

    clearConsole();

    if (find(licenseCategories.begin(), licenseCategories.end(),
DriverLicense(choice - 1)) == licenseCategories.end())

```

```

        {
            driver.addLicenseCategory(DriverLicense(choice - 1));
            cout << "Категория " << toStringEnum(DriverLicense(choice - 1)) << "
добавлена.\n";
        }
        else
        {
            driver.getLicenseCategories().erase(find(licenseCategories.begin(),
licenseCategories.end(), DriverLicense(choice - 1)));
            vector<VehicleType> &vehicles = driver.getVehicles(); // Ссылка на
транспортные средства водителя
            cout << "Категория " << toStringEnum(DriverLicense(choice - 1)) << "
удалена.\n";
            auto vehicleIt = find(vehicles.begin(), vehicles.end(), VehicleType(choice
- 1));
            if (vehicleIt != vehicles.end())
            {
                vehicles.erase(vehicleIt);
                cout << "Транспортное средство " << toStringEnum(VehicleType(choice -
1)) << " удалено.\n";
            }
        }
        system("pause");
        return 1;
    }
    // меню добавления/удаление транспорта
    bool manageDriverVehiclesMenu(Driver &driver)
    {
        clearConsole();

        vector<DriverLicense> &licenseCategories = driver.getLicenseCategories();

        if (licenseCategories.empty())
        {
            cout << "У водителя нет прав\n";
            system("pause");
            return 0;
        }

        vector<VehicleType> currentLC;

```

```

int i = 1;
for (const DriverLicense &category : licenseCategories)
{
    cout << i++ << " ) " << toStringEnum(VehicleType(int(category))) << endl;
    currentLC.push_back(VehicleType(int(category)));
}

// Выводим список доступных транспортных средств для выбора
cout << "Выберите транспортное средство для добавления:\n";

if (!inputSingleInt(choice, 1, 5))
    return 0;

VehicleType selectedVehicle = currentLC[choice - 1];
vector<VehicleType> &vehicles = driver.getVehicles(); // Ссылка на транспортные
средства водителя

clearConsole();

auto vehicleIt = find(vehicles.begin(), vehicles.end(), selectedVehicle);
if (vehicleIt == vehicles.end())
{
    // Добавление, если транспортное средство не найдено
    vehicles.push_back(selectedVehicle);
    cout << "Транспортное средство " << toStringEnum(selectedVehicle) << "
добавлено.\n";
}
else
{
    // Удаление, если транспортное средство уже существует
    vehicles.erase(vehicleIt);
    cout << "Транспортное средство " << toStringEnum(selectedVehicle) << "
удалено.\n";
}

system("pause");
return 1;
}

// функции директора
bool manageEmployeeMenu(Director &director)

```

```

{
    clearConsole();

    director.displayInfo();
    cout << endl;
    cout << "Список возможных функций:" << '\n'
        << "1. Изменить ФИО" << '\n'
        << "2. Нанять работника" << '\n'
        << "3. Уволить работника" << '\n'
        << "4. Вывести всех работников" << '\n'
        << "5. Управление бюджетом отделов" << '\n';

    if (!inputSingleInt(choice, 1, 5))
        return 0;

    switch (choice)
    {
    case 1: // изменение ФИО
        setFIO(director);
        return 1;
    case 2: // Нанять работника
        try
        {
            while (hireEmployeeMenu(director))
                ;
        }
        catch (const invalid_argument &e)
        {
            clearConsole();
            cout << "Ошибка при найме сотрудника: " << e.what() << endl;

            system("pause");
        }
        return 1;
    case 3: // уволить работника
        while (terminateEmployeeMenu(director))
            ;
        return 1;
    case 4: // вывести всех работников данного директора
    {
        clearConsole();

```

```

vector<Person *> vect;

cout << left << setw(3) << "№ |"
    << setw(10) << "Должность|"
    << setw(45) << "    Фамилия    |    Имя    |    Отчество    |"
    << "    Зарплата    |" << endl;
cout << string(2, '-') << '+' << string(9, '-') << '+' << string(15, '-')
<< '+' << string(15, '-') << '+' << string(15, '-') << '+' << string(13, '-') << '+' <<
endl;

    displayAllEmployees(director, vect);
    system("pause");
    return 1;
}
case 5: // Управление бюджетом отделов
    while (manageBudgetDepartmentsMenu(director))
        ;
    return 1;
default:
    return 1;
}
return 1;
}
// функции бухгалтера
bool manageEmployeeMenu(Accountant *accountant, Director &director)
{
    clearConsole();
    accountant->displayInfo();
    cout << endl;
    cout << "Список возможных функций:" << '\n'
        << "1. Изменить ФИО" << '\n'
        << "2. Изменить ставку сотруднику " << '\n'
        << "3. Назначить премиальные сотруднику" << '\n'
        << "4. Изменить коэффициент ЗП должности" << '\n'
        << "5. Вывести информацию о всех работниках" << '\n';

    if (!inputSingleInt(choice, 1, 5))
        return 0;

    switch (choice)
    {

```

```

case 1: // изменение ФИО
    setFIO(*accountant);
    return 1;
case 2: // Изменить ставку сотруднику
    while (setEmployeesRatesMenu(*accountant, director))
        ;
    return 1;
case 3: // Назначить премиальные сотруднику
    while (setEmployeesBonusesMenu(*accountant, director))
        ;
    return 1;
case 4: // Изменить коэффициент ЗП должности
    while (setCoefficientMenu(*accountant, director))
        ;
    return 1;
case 5: // Вывести информацию о всех работниках
    displayAllEmployeesWithRatesAndBonuses(director, *accountant);
    system("pause");
    return 1;
default:
    return 1;
}
return 1;
}

// функции секретаря
bool manageEmployeeMenu(Secretary *secretary, Director &director)
{
    clearConsole();

    map<Person *, Position> securityOrDriver;

    secretary->displayInfo();
    cout << endl;
    cout << "Список возможных функций:" << '\n'
        << "1. Изменить ФИО" << '\n'
        << "2. Вывести сотрудников, данного директора" << '\n'
        << "3. Вывести данные об охранниках" << '\n'
        << "4. Вывести данные о водителях" << '\n'
        << "5. Управление встречами" << '\n';

    if (!inputSingleInt(choice, 1, 5))

```

```

        return 0;

    switch (choice)
    {
    case 1: // изменение ФИО
        setFIO(*secretary);
        return 1;
    case 2: // Вывести сотрудников, данного директора
        clearConsole();

        cout << left << setw(3) << "№ |"
             << setw(10) << "Должность|"
             << setw(45) << "    Фамилия    |    Имя    |    Отчество    |"
             << "    Зарплата    |" << endl;

        cout << string(2, '-') << '+' << string(9, '-') << '+' << string(15, '-')
        << '+' << string(15, '-') << '+' << string(15, '-') << '+' << string(14, '-') << '+' <<
        endl;

        secretary->displayEmployeeListInTable(director.getEmployees());
        system("pause");
        return 1;
    case 3: // Вывести данные об охранниках
        securityOrDriver.clear();
        clearConsole();

        cout << left << setw(3) << "№ |"
             << setw(10) << "Должность|"
             << setw(45) << "    Фамилия    |    Имя    |    Отчество    |"
             << "    Зарплата    |" << endl;

        cout << string(2, '-') << '+' << string(9, '-') << '+' << string(15, '-')
        << '+' << string(15, '-') << '+' << string(15, '-') << '+' << string(14, '-') << '+' <<
        endl;

        for (const auto pair : director.getEmployees())
        {
            if (pair.second == Position::SecurityGuard)
                securityOrDriver[pair.first] = pair.second;
        }
        if (securityOrDriver.empty())
        {
            clearConsole();

```



```

        cout << "Сотрудников нет\n";
        system("pause");
        return 1;
    }
    secretary->displayEmployeeListInTable(securityOrDriver);
    system("pause");
    return 1;
case 4: // Вывести данные о водителях
    securityOrDriver.clear();
    clearConsole();

    cout << left << setw(3) << "№ |"
        << setw(10) << "Должность|"
        << setw(45) << "    Фамилия    |    Имя    |    Отчество    |"
        << "    Зарплата    |" << endl;
    cout << string(2, '-') << '+' << string(9, '-') << '+' << string(15, '-')
    << '+' << string(15, '-') << '+' << string(15, '-') << '+' << string(14, '-') << '+' <<
    endl;

    for (const auto pair : director.getEmployees())
    {
        if (pair.second == Position::Driver)
            securityOrDriver[pair.first] = pair.second;
    }
    if (securityOrDriver.empty())
    {
        clearConsole();
        cout << "Сотрудников нет\n";
        system("pause");
        return 1;
    }
    secretary->displayEmployeeListInTable(securityOrDriver);
    system("pause");
    return 1;
case 5: // Управление встречами
    while (meetingManageMenu(*secretary, director))
        ;
    return 1;
default:
    return 1;
}

```

```

        return 1;
    }
    // функции охранника
    bool manageEmployeeMenu(SecurityGuard *securityGuard)
    {
        clearConsole();

        securityGuard->displayInfo();
        cout << endl;
        cout << "Список возможных функций:" << '\n'
             << "1. Изменить ФИО" << '\n'
             << "2. Управление спец. средством" << '\n'
             << "3. Управление статусом объектов" << '\n';

        if (!inputSingleInt(choice, 1, 3))
            return 0;

        switch (choice)
        {
        case 1: // изменение ФИО
            setFIO(*securityGuard);
            return 1;
        case 2: // Управление спец. средством
            clearConsole();
            while (manageSpecialEquipmentMenu(*securityGuard))
                ;
            return 1;
        case 3: // Управление статусом объектов
            clearConsole();
            while (manageStatusLocationsMenu(*securityGuard))
                ;
            system("pause");
            return 1;
        default:
            return 1;
        }
        return 1;
    }
    // функции водителя
    bool manageEmployeeMenu(Driver *driver)
    {

```

```

clearConsole();

driver->displayInfo();
cout << endl;
cout << "Список возможных функций:" << '\n'
    << "1. Изменить ФИО" << '\n'
    << "2. Управление водительскими правами " << '\n'
    << "3. Управление транспортом" << '\n'
    << "4. Отправить водителя в путешествие" << '\n';

if (!inputSingleInt(choice, 1, 4))
    return 0;

switch (choice)
{
case 1: // изменение ФИО
    setFIO(*driver);
    return 1;
case 2: // Управление водительскими правами
    clearConsole();
    while (manageDriverLicenseCategoriesMenu(*driver))
        ;
    return 1;
case 3: // Управление транспортом
    clearConsole();
    while (manageDriverVehiclesMenu(*driver))
        ;
    system("pause");
    return 1;
case 4: // Отправить водителя в путешествие
    clearConsole();
    while (manageDriverDestinationMenu(*driver))
        ;
    system("pause");
    return 1;
default:
    return 1;
}
return 1;
}
};

```

```

char getUppercaseSymbol(int value)
{
    // Проверяем, существует ли число в массиве и возвращаем соответствующий символ
    switch (value)
    {
        // Русские символы
        case 241:
            return 'Ё';
        case 169:
            return 'Й';
        case 230:
            return 'Ц';
        case 227:
            return 'У';
        case 170:
            return 'К';
        case 165:
            return 'Е';
        case 173:
            return 'Н';
        case 163:
            return 'Г';
        case 232:
            return 'Ш';
        case 233:
            return 'Щ';
        case 167:
            return 'З';
        case 229:
            return 'Х';
        case 234:
            return 'б';
        case 228:
            return 'ф';
        case 235:
            return 'bl';
        case 162:
            return 'В';
        case 160:
            return 'А';
    }
}

```

```

case 175:
    return 'П';
case 224:
    return 'Р';
case 174:
    return 'О';
case 171:
    return 'Л';
case 164:
    return 'Д';
case 166:
    return 'Ж';
case 237:
    return 'Э';
case 239:
    return 'Я';
case 231:
    return 'Ч';
case 225:
    return 'С';
case 172:
    return 'М';
case 168:
    return 'И';
case 226:
    return 'Т';
case 236:
    return 'Ь';
case 161:
    return 'Б';
case 238:
    return 'Ю';

// Английские символы
case 113:
    return 'Q'; // q
case 119:
    return 'W'; // w
case 101:
    return 'E'; // e
case 114:

```

```
        return 'R'; // r
case 116:
        return 'T'; // t
case 121:
        return 'Y'; // y
case 117:
        return 'U'; // u
case 105:
        return 'I'; // i
case 111:
        return 'O'; // o
case 112:
        return 'P'; // p
case 97:
        return 'A'; // a
case 115:
        return 'S'; // s
case 100:
        return 'D'; // d
case 102:
        return 'F'; // f
case 103:
        return 'G'; // g
case 104:
        return 'H'; // h
case 106:
        return 'J'; // j
case 107:
        return 'K'; // k
case 108:
        return 'L'; // l
case 122:
        return 'Z'; // z
case 120:
        return 'X'; // x
case 99:
        return 'C'; // c
case 118:
        return 'V'; // v
case 98:
        return 'B'; // b
```

```

        case 110:
            return 'N'; // n
        case 109:
            return 'M'; // m
        default:
            return '\0';
    }
}

bool Menu::inputSingleInt(int &digit, const int start, const int end) // ввод
положительной цифры
{
    digit = 0;
    cout << "Введите число: ";
    char ch;
    while (true)
    {
        ch = getch();
        if (ch == 27)
            return 0;
        if (isdigit(ch))
            digit = int(ch) - 48;
        if (digit >= start && digit <= end)
            return 1;
    }
}

bool Menu::inputString(string &str, const int &minLen, const int &maxLen, bool
firstBigLetter, bool allowNumbers) // ввод строки
{
    wchar_t ch;
    while (true)
    {
        ch = getch();

        // Выход при нажатии ESC
        if (int(ch) == 27) // esc
            return 0;

        // Завершение ввода при нажатии Enter, если длина строки подходит

```

```

        if (int(ch) == 13 && !str.empty() && str.length() >= minLen && str.back() != '-'
') // enter
        {
            return 1;
        }

        // Удаление последнего символа при нажатии Backspace
        if (int(ch) == 8 && !str.empty()) // del
        {
            cout << "\b \b";
            str.pop_back();
        }

        // Добавление символа, если он соответствует критериям
        if (int(ch) != 13 && int(ch) != 9 && int(ch) != 8 && str.length() < maxLen)
        {
            // Проверка на дефис: разрешен, если не первый символ и предыдущий символ
            не '-'

            if (ch == '-' && !str.empty() && str.back() != '-' && str.length() !=
maxLen - 1 && str.find('-') == string::npos)
            {
                str += char(ch);
                cout << char(ch);
            }

            // Проверка на буквы с учетом заглавной первой буквы
            else if (getUppercaseSymbol(int(ch)) != '\0')
            {
                if ((str.empty() && firstBigLetter) || (str.back() == '-' &&
firstBigLetter))
                {
                    str += getUppercaseSymbol(int(ch));
                    cout << getUppercaseSymbol(int(ch));
                }
                else
                {
                    str += char(ch);
                    cout << char(ch);
                }
            }

            // Проверка на цифры, если разрешен их ввод
            else if (allowNumbers && isdigit(ch))
            {
                str += char(ch);
            }
        }
    }
}

```



```

        cout << char(ch);
    }
}
}

bool Menu::inputInt(int &number, const int &min, const int &max, const string &key,
const bool minus, const bool point)
{
    string ans = "";
    char ch;
    while (true)
    {
        ch = getch();
        if (int(ch) == 27) // esc
            return 0;
        if (ch == '-' && ans.empty() && minus) // -
        {
            ans.append("-");
            cout << '-';
        }
        if (int(ch) == 13 && !ans.empty() && stoi(ans) >= min && (!ans.empty() &&
!(ans.length() == 1 && ans[0] == '-'))) // enter
            break;
        if (int(ch) == 9 && key == "Tab")
        {
            number = min - 1; // ЧИСЛО = -1 ЕСЛИ НАЖАЛИ TAB
            return 1;
        }
        if (int(ch) == 8 && !ans.empty()) // del
        {
            cout << "\b \b";
            ans.pop_back();
        }
        if (isdigit(ch) && stoi(ans + ch) <= max && ans.length() <=
to_string(max).length())
        {
            ans += ch;
            cout << ch;
        }
    }
}

```

```

        if (!ans.empty())
            number = stoi(ans);
        return 1;
    }

bool Menu::inputDouble(double &number, const double &min, const double &max, const
string &key, const bool minus, const bool point, int decimalLimit)
{
    string ans = "";
    char ch;
    bool hasPoint = false; // Флаг для отслеживания точки
    int decimalsCount = 0; // Счётчик знаков после точки

    while (true)
    {
        ch = getch();

        if (int(ch) == 27) // ESC
            return false;

        if (ch == '-' && ans.empty() && minus) // Обработка знака "-"
        {
            ans.append("-");
            cout << '-';
        }

        // Автозаполнение "0." при вводе нуля первым символом
        if (ch == '0' && ans.empty() && point)
        {
            ans = "0.";
            hasPoint = true;
            cout << "0.";
            continue;
        }

        // Обработка ввода точки
        if (ch == '.' && point && !hasPoint && !ans.empty() && ans != "-")
        {
            ans.append(".");
            hasPoint = true;
            cout << '.';
        }
    }
}

```

```

        continue;
    }

    if (int(ch) == 13 && (!ans.empty() && !(ans.length() == 1 && ans[0] == '-'))
// ENTER
        break;

    if (int(ch) == 9 && key == "Tab") // TAB
    {
        number = min - 1; // Вернуть значение ниже минимума, если нажата Tab
        return true;
    }

    if (int(ch) == 8 && !ans.empty()) // Удаление символа (Backspace)
    {
        if (ans.back() == '.')
            hasPoint = false;
        else if (hasPoint && decimalsCount > 0)
            decimalsCount--;

        cout << "\b \b";
        ans.pop_back();
    }

    // Обработка ввода цифр с учетом диапазона min и max
    if (isdigit(ch) && (stod(ans + ch) <= max && stod(ans + ch) >= min))
    {
        // Ограничение на количество знаков после точки
        if (hasPoint && decimalsCount >= decimalLimit)
            continue;

        ans += ch;
        cout << ch;

        if (hasPoint)
            decimalsCount++;
    }
}

if (!ans.empty())
    number = stod(ans); // Преобразование строки в число с плавающей точкой

```

```

        return true;
    }

void Menu::clearConsole() // очистка консоли и вывод сообщения "нажмите esc чтобы
выйти"
{
    system("cls");
    cout << "нажмите esc чтобы выйти" << '\n'
        << '\n';
}

int main(){
    Menu menu;
    srand(static_cast<unsigned int>(time(nullptr)));
    while (menu.createCompanyMenu())
        ;
    while (menu.CompanyMenu())
        ;
    system("cls");
    cout << "Завершения выполнения программы \n";

    system("pause");
    return 0;
}

```