МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ

РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ

ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«ВОРОНЕЖСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ

УНИВЕРСИТЕТ»

(ФГБОУ ВО «ВГТУ», ВГТУ)

Факультет информационных технологий и компьютерной безопасности

(факультет)

Кафедра Систем управления и информационных технологий в строительстве

КУРСОВОЙ ПРОЕКТ

по дисциплине: «Объектно-ориентированное программирование» .

Тема: «Разработка приложения «Сведения о промышленных предприятиях»» .

Расчетно-пояснительная записка

Разработал студент А.С. Кочетов

Подпись, дата Инициалы, фамилия

Руководитель О.В. Минакова

Подпись, дата Инициалы, фамилия

Нормоконтролер О.В. Минакова

Подпись, дата Инициалы, фамилия

Защищена \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Оценка \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

дата

ВОРОНЕЖ

2025

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ

РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ

ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«ВОРОНЕЖСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

(ФГБОУ ВО «ВГТУ», ВГТУ)

Кафедра Систем управления и информационных технологий в строительстве

ЗАДАНИЕ

на курсовой проект

по дисциплине: «Объектно-ориентированное программирование» .

Тема: «Разработка приложения «Сведения о промышленных предприятиях»» ..

Студент збПИЭ-211 Кочетов Александр Сергеевич .

Группа, фамилия, имя, отчество

Номер варианта. 14 .

Технические условия:. Среда разработки \*. Язык программирования java, редактор диаграмм draw.io.

Содержание и объем работы (графические работы, расчеты и прочее):. .

. 26 стр., 4 иллюстр. 1 табл. .

Сроки выполнения этапов:. анализ и постановка задачи ( ); разработка пошаговой детализации программы ( ); реализация программы ( ); тестирование программы ( ); оформление пояснительной записки ( ).

.

Срок защиты курсового проекта: .

Руководитель . О.В. Минакова .

Подпись, дата Инициалы, фамилия

Задание принял студент . А.С. Кочетов

Подпись, дата Инициалы, фамилия

СОДЕРЖАНИЕ

ВВЕДЕНИЕ…………………………………………..………………….…………4

1. Постановка задачи 5

1.1. Обоснование полей класса и методов основной сущности…………..

программы………………………………….. 5

1.2. CRC-карточка………………………………….. 7

2. Разработка интерфейсной части. 9

2.1 Описание графического интерфейса пользователя. 9

2.2 Диаграмма последовательностей взаимодействия. 12

3. Разработка кода программы.. 14

3.1 Диаграмма классов приложения и её описание 14

3.2. Алгоритм реализации методов. 23

ЗАКЛЮЧЕНИЕ…………………………………………...…...…….…………...25

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ.…………...………..….…….26

ВВЕДЕНИЕ

Современные промышленные предприятия играют ключевую роль в экономике любой страны, и эффективное управление их деятельностью требует наличия точной и актуальной информации. Сбор, анализ и представление данных о промышленных компаниях становятся непростыми задачами в условиях постоянно меняющихся экономических реалий. Настоящий курсовой проект посвящен разработке программного приложения «Сведения о промышленных предприятиях», которое позволяет пользователям удобно управлять информацией о предприятиях, включая их названия, адреса, численность сотрудников и годовые доходы.

Данная программа призвана значительно упростить процесс ввода и хранения данных, предоставляя интуитивно понятный графический интерфейс, где пользователи могут легко добавлять и просматривать информацию о различных предприятиях. С использованием языка программирования Java и технологий JavaFX, приложение обеспечивает современный и функциональный интерфейс, который поддерживает взаимодействие с пользователем и наглядное отображение данных.

В процессе работы над проектом будет рассмотрен процесс проектирования и реализации приложения, включающий в себя анализ требований, разработку интерфейса и кодов программы, а также тестирование функциональности. В результате, пользователи получат эффективные инструменты для управления информацией о промышленных предприятиях, что поможет им принимать обоснованные решения и оптимизировать процессы на уровне управления.

1. Постановка задачи

1.1 Обоснование полей класса и методов основной сущности программы

В данном разделе рассматривается приложение «Сведения о промышленных предприятиях», цель которого - собрать и предоставить пользователю информацию о различных предприятиях [1]. Основной сущностью программы является класс ManufacturingEnterprise, который наследует класс Enterprise. Этот класс представляет собой предприятие, занимающееся производственной деятельностью, и включает в себя информацию о сотрудниках и продуктах, что отражает его ключевую роль в приложении "Сведения о промышленных предприятиях".

Основные поля класса ManufacturingEnterprise:

1. employees: List<Employee> список сотрудников, работающих на предприятии.
2. products: List<Product> список продуктов, производимых предприятием.

Основные методы класса ManufacturingEnterprise

1. Конструкторы:

ManufacturingEnterprise(): конструктор по умолчанию, который вызывает конструктор суперкласса с пустыми значениями.

ManufacturingEnterprise(String name, String address, int numberOfEmployees, double annualRevenue): конструктор, инициализирующий основные поля (суперкласса) и создающий пустые списки для сотрудников и продуктов.

1. Методы для добавления элементов:

void addEmployee(Employee employee): добавляет нового сотрудника в список employees.

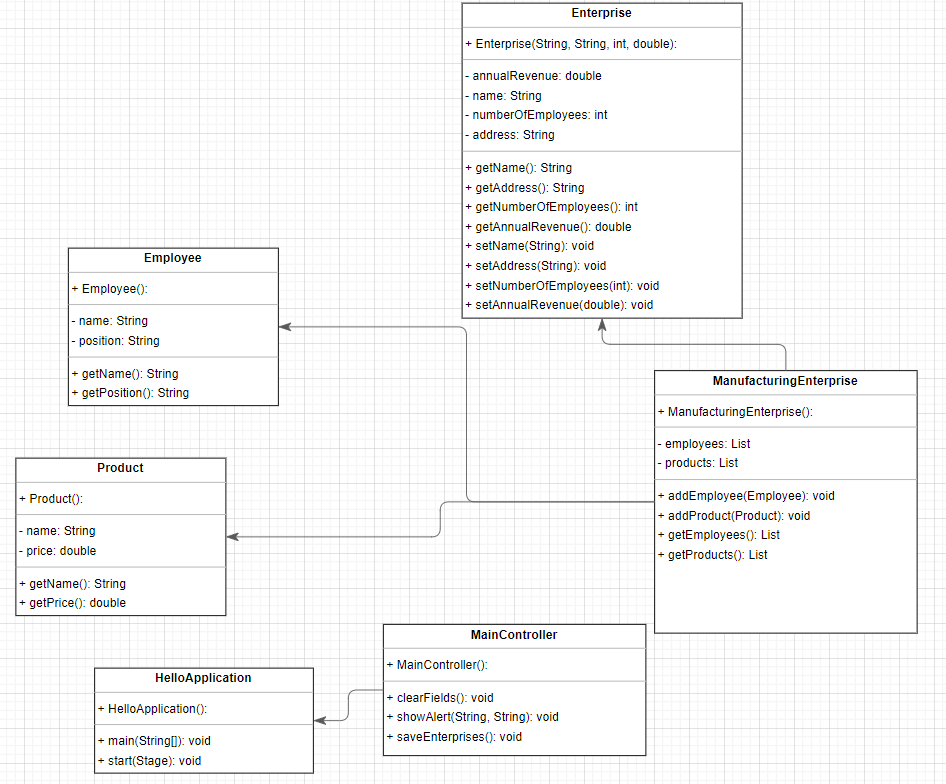
void addProduct(Product product): добавляет новый продукт в список products.

1. Геттеры:

List<Employee> getEmployees(): возвращает список сотрудников предприятия.

List<Product> getProducts(): возвращает список продуктов предприятия.

Класс ManufacturingEnterprise является центральной сущностью, которая объединяет информацию о сотрудниках и продуктах, производимых данным предприятием, и наследует общие характеристики от класса Enterprise. Это делает его основой для построения приложения о промышленных предприятиях.

Начало формы

Для наглядного представления структуры класса ManufacturingEnterprise можно использовать диаграмму классов, представленную на рисунке 1.1., которая позволяет визуализировать основные компоненты и характеристики данного класса [2]. Эта диаграмма позволяет увидеть, как организованы атрибуты и методы класса, а также демонстрирует взаимосвязи с другими классами в системе.

Рисунок 1.1. Диаграмма классов.

1.2 CRC-карточка

Вот подробная таблица 1.1 CRC-карточки для класса ManufacturingEnterprise [3]:

Таблица 1.1. CRC - карточка

|  |  |
| --- | --- |
| Class: ManufacturingEnterprise | |
| Ответственность | Коллаборации |
| Хранение информации о предприятии | Employee |
| Product |
| Enterprise |
| Предоставление методов для получения и установки атрибутов предприятия. | ManufacturingEnterprise(): |
| ManufacturingEnterprise(String name, String address, int numberOfEmployees, double annualRevenue) |
| void addEmployee(Employee employee) |
| void addProduct(Product product) |
| List<Employee> getEmployees() |
| List<Product> getProducts() |

Объяснение каждой коллаборации:

1. Employee: Список сотрудников, работающих на предприятии, (employees) хранит объекты типа Employee, представляющие работников предприятия. Этот список используется для управления и отображения информации о сотрудниках.
2. Product: Список продуктов, производимых предприятием, (products) - хранит объекты типа Product, представляющие продукцию, производимую предприятием. Данный список позволяет отслеживать и управлять ассортиментом продукции.
3. public ManufacturingEnterprise(): Конструктор по умолчанию. Инициализирует новый объект ManufacturingEnterprise с пустыми списками для сотрудников и продуктов и вызывает конструктор суперкласса с пустыми значениями.
4. public ManufacturingEnterprise(String name, String address, int numberOfEmployees, double annualRevenue): Конструктор, который принимает параметры для инициализации полей суперкласса (name, address, numberOfEmployees, annualRevenue). Также создает пустые списки для сотрудников и продуктов.
5. public void addEmployee(Employee employee): Метод для добавления нового сотрудника в список employees. Принимает объект Employee и добавляет его в коллекцию, тем самым обновляя данные о работниках предприятия.
6. public void addProduct(Product product): Метод для добавления нового продукта в список products. Принимает объект Product и добавляет его в коллекцию, что позволяет предприятиям управлять своим ассортиментом.
7. public List<Employee> getEmployees(): Метод для получения списка всех сотрудников предприятия. Возвращает коллекцию employees, что позволяет извлекать информацию о работниках.
8. public List<Product> getProducts(): Метод для получения списка всех продуктов предприятия. Возвращает коллекцию products, что позволяет извлекать информацию о производимой продукции.

Эта таблица CRC-карты дает чёткое представление о структуре класса ManufacturingEnterprise.

2. Разработка интерфейсной части

2.1 Описание графического интерфейса пользователя

Графический интерфейс пользователя (GUI) включает в себя несколько элементов управления, которые позволяют пользователю взаимодействовать с программой [5].

Основные элементы управления:

TextField для ввода названия предприятия, адреса, количества сотрудников и годовой выручки, информации о сотрудниках и продуктах.

TableView для отображения списка предприятий.

Button для добавления нового предприятия, сотрудника, продукта.

TableColumn для отображения отдельных полей в таблице.

Эти элементы позволяют пользователю вводить и просматривать данные о промышленных предприятиях, их сотрудниках и продуктах.

Произведем раскадровку рабочего окна, рисунок 2.1., на нем наглядно можно увидеть основные элементы управления.

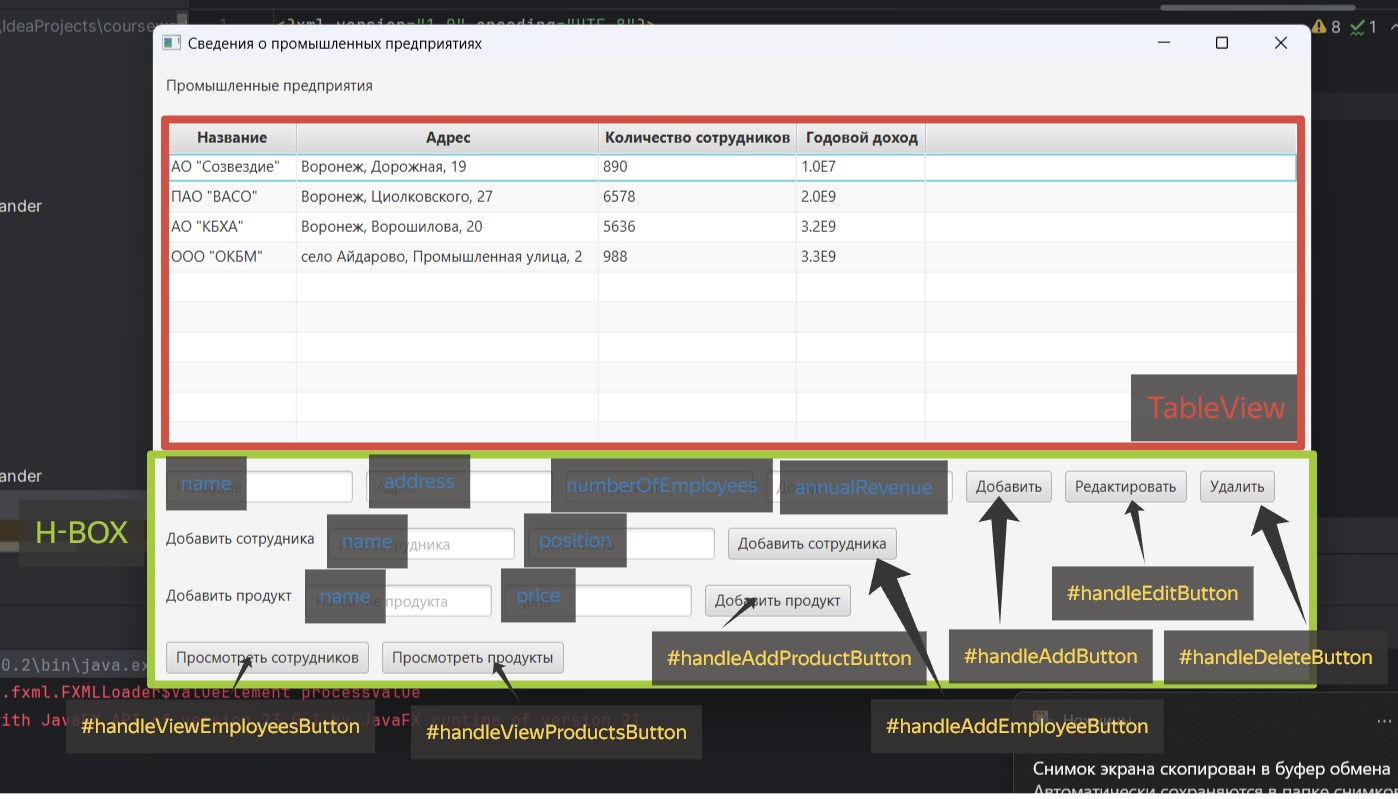


Рисунок 2.1. Раскадровка рабочего окна.

VBox используется как основной контейнер для размещения элементов управления вертикально. Менеджер компоновки VBox обеспечивает равномерное распределение свободного пространства между элементами, позволяя таблице занимать оставшуюся часть окна.

HBox применяется для горизонтальной группировки элементов ввода и кнопки добавления. Менеджер компоновки HBox гарантирует, что все элементы располагаются в одной строке, обеспечивая удобный доступ ко всем полям ввода одновременно.

Выбор менеджеров компоновки обусловлен необходимостью обеспечить удобное расположение элементов управления на экране. Использование VBox позволяет легко управлять вертикальным расположением элементов, что особенно важно для таблицы, которая должна занимать большую часть экрана. HBox, в свою очередь, помогает организовать элементы ввода в одну строку, делая процесс заполнения формы интуитивно понятным и удобным для пользователя.

Раскадровка рабочего окна программы включает два основных этапа:

1. Этап ввода данных:

- Пользователь вводит данные о новом предприятии через текстовое поле. После завершения ввода нажимается кнопка «Добавить».

- Далее заполняется информация о сотрудниках и продуктах предприятия через соответствующие текстовые поля, где также нажимает кнопку «Добавить сотрудника» или «Добавить продукт».

1. Этап просмотра данных:

- Данные о введенном предприятии отображаются в таблице, где пользователь может просмотреть и, при необходимости, изменить или удалить информацию.

- Данные о сотрудниках и продуктах предприятия можно посмотреть, нажав соответствующие кнопки «Просмотреть сотрудников» и «Просмотреть продукты», предварительно выбрав нужное предприятие.

Иерархия элементов управления:

- Главное окно (Stage)

- Панель (GridPane)

- TableView таблица для отображения предприятий

- TableColumn колонка для названия предприятия

- TableColumn колонка для адреса предприятия

-TableColumn колонка для количества сотруд. предприятия

- TableColumn колонка для годовой выручки предприятия

- Панель (HBox) панель для введения информации о предприятии

- TextField для ввода названия предприятия

- TextField для ввода адреса

- TextField для ввода количества сотрудников

- TextField для ввода годовой выручки

- Button для добавления нового предприятия

- Панель (HBox) панель для добавления сотрудника

- TextField для ввода ФИО сотрудника

- TextField для ввода должности сотрудника

- Button для добавления нового сотрудника

- Панель (HBox) панель для добавления продукта

- TextField для ввода названия продукта

- TextField для ввода стоимости продукта

- Button для добавления нового продукта

- Панель (HBox) панель для просмотра сотрудников и продуктов

- Button для отображения списка сотрудников

- Button для отображения списка продуктов

2.2 Диаграмма последовательностей взаимодействия

Для более наглядной демонстрации взаимодействия пользователя с элементами управления и объектами классами можно эффективно использовать диаграммы последовательностей, которые позволяют визуализировать последовательность взаимодействий между различными компонентами системы в течение определенного сценария. Такую диаграмму можно считать своеобразной картой, показывающей, как именно происходит взаимодействие между пользователем и программой, включая все этапы и шаги, которые необходимо выполнить.

В целом, диаграмма взаимодействия последовательностей (рисунок 2.2.) служит важным инструментом, позволяющим визуализировать и систематизировать шаги, которые пользователь проходит в процессе работы с программой, обеспечивая тем самым более глубокое понимание функциональности и эффективности взаимодействия с программным обеспечением.

Диаграмма последовательностей для добавления нового предприятия, сотрудника и продукта представлена на рисунке 2.2. На данном рисунке отображены ключевые моменты, начиная от инициирования процесса пользователем, где происходит нажатие кнопки для добавления нового предприятия, и заканчивая подтверждением успешного добавления записи в систему. Каждый шаг, который пользователь предпринимает, и каждый ответ системы на эти действия четко продемонстрированы, что помогает лучше понять процесс, а также выявить возможные узкие места или сложности, с которыми может столкнуться пользователь [6].

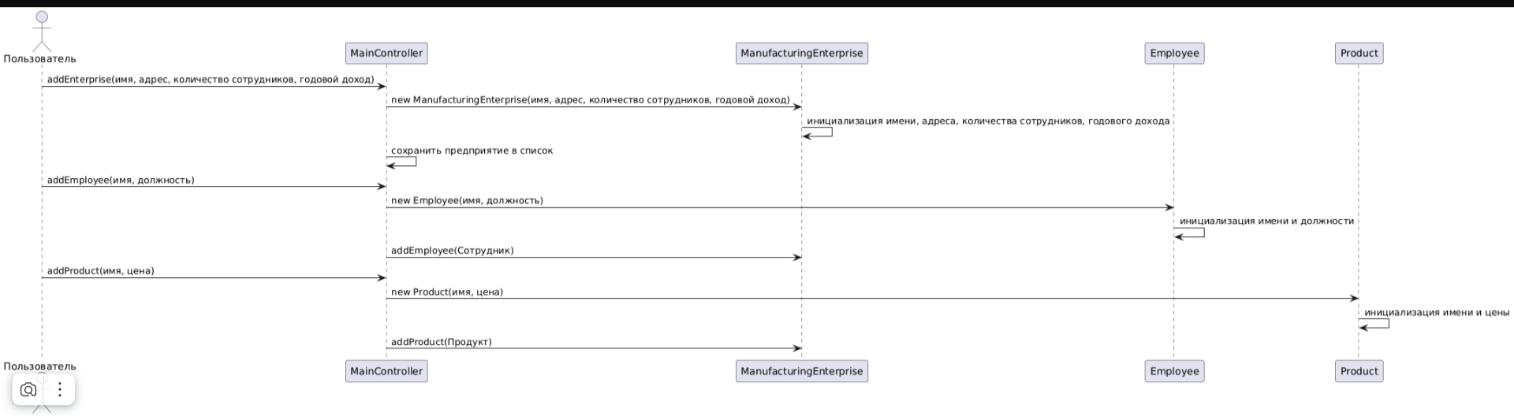


Рисунок 2.2. Диаграмма взаимодействия последовательностей для добавления нового предприятия, сотрудника и продукта

Таким образом, диаграммы последовательностей не только иллюстрируют ход взаимодействия, но и имеют важное значение для анализа и оптимизации пользовательского опыта, поскольку они дают возможность разработчикам видеть полный контекст работы программы и предпочтения пользователей.

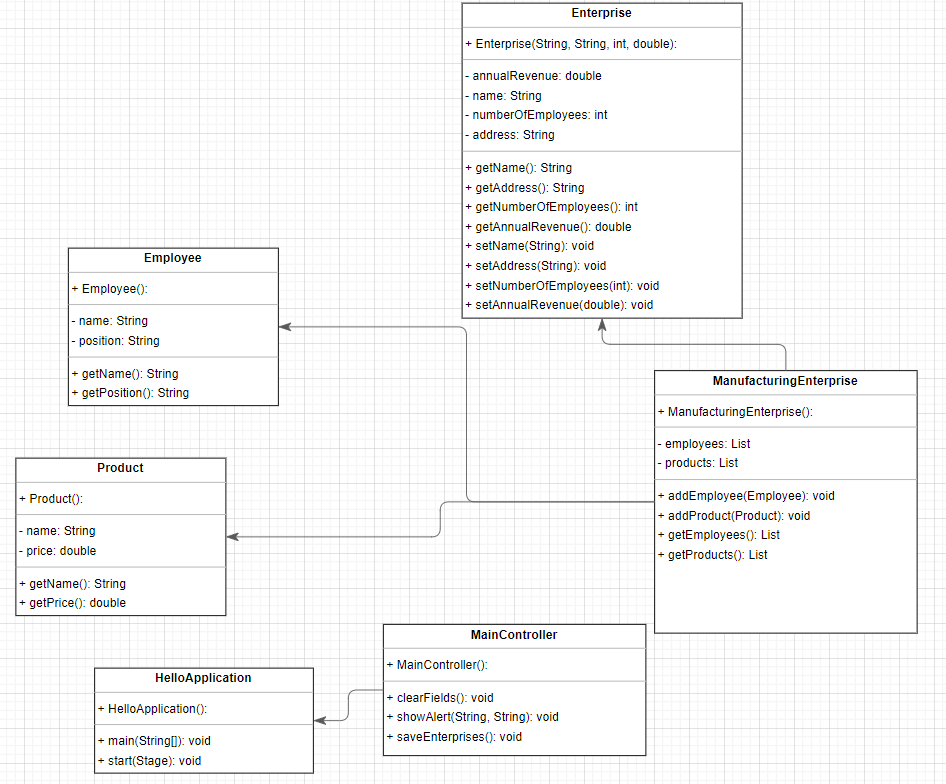
3. Построение кода программы

3.1 Диаграмма классов приложения и её описание

Диаграмма классов приложения (рисунок 3.1.) отражает структуру классов и их взаимосвязь, что является очень важным аспектом для понимания архитектуры всего приложения. Каждый класс в этой диаграмме имеет свои уникальные атрибуты и методы, которые определяют его поведение и функции в рамках работы системы. Атрибуты представляют собой характеристики, которые описывают состояние объекта, а методы — это действия, которые объект может выполнять, создавая тем самым функциональность приложения [8].

Кроме того, взаимосвязи между классами демонстрируют, как они взаимодействуют друг с другом, что позволяет выстраивать осмысленную и эффективную логику работы программы. Наличие различных типов связей, таких как ассоциации, агрегации и композиций, также подчеркивает комплексность и многоуровневую структуру, которая обеспечивает согласованность и целостность всего приложения. Такой визуальный инструмент, как диаграмма классов, значительно упрощает процесс проектирования и разработки программного обеспечения, позволяя разработчикам и заинтересованным сторонам легко видеть, как различные компоненты системы связаны и взаимодействуют между собой.

В конечном итоге, диаграмма классов является не просто иллюстрацией, а настоящим инструментом, который способствует лучшему пониманию структуры приложения и служит основой для его дальнейшего развития и масштабирования.

Рисунок 3.1. Диаграмма классов приложения.

Описание каждого класса [9]:

**1. Класс Employee**

**Назначение:** Класс Employee предназначен для представления информации о сотрудниках промышленного предприятия.

**Атрибуты:**

* **name** (тип: String):
  + **Назначение**: хранит имя сотрудника.
  + **Обоснование выбора типа**: String позволяет легко хранить и обрабатывать текстовую информацию.
  + **Доступ**: приватный, так как не должен изменяться напрямую извне класса.
* **position** (тип: String):
  + **Назначение**: хранит должность сотрудника.
  + **Обоснование выбора типа**: также текстовая информация, которая легко представима и обрабатываема как строка.
  + **Доступ**: приватный, чтобы защитить данные.

**Методы:**

* **Конструкторы**:
  + Employee()
    - **Назначение**: создает объект Employee с пустыми значениями для name и position.
  + Employee(String name, String position)
    - **Назначение**: создает объект Employee с указанными значениями для name и position.
* **Методы доступа**:
  + String getName()
    - **Назначение**: возвращает имя сотрудника.
    - **Возвращаемое значение**: String
      * имя сотрудника.
  + String getPosition()
    - **Назначение**: возвращает должность сотрудника.
    - **Возвращаемое значение**: String
      * должность сотрудника.

**2. Класс Product**

**Назначение:** Класс Product предназначен для представления информации о продуктах, производимых предприятием.

**Атрибуты:**

* **name** (тип: String):
  + **Назначение**: хранит название продукта.
  + **Обоснование выбора типа**: текстовая информация, легко хранится в строковом формате.
  + **Доступ**: приватный, чтобы предотвратить неконтролируемое изменение.
* **price** (тип: double):
  + **Назначение**: хранит цену продукта.
  + **Обоснование выбора типа**: double подходящий тип для работы с денежными значениями, позволяющий учитывать десятичные дроби.
  + **Доступ**: приватный, для защиты от некорректного изменения.

**Методы:**

* **Конструкторы**:
  + Product()
    - **Назначение**: создает объект Product с пустым названием и нулевой ценой.
  + Product(String name, double price)
    - **Назначение**: создает объект Product с указанными значениями для name и price.
* **Методы доступа**:
  + String getName()
    - **Назначение**: возвращает название продукта.
    - **Возвращаемое значение**: String
      * название продукта.
  + double getPrice()
    - **Назначение**: возвращает цену продукта.
    - **Возвращаемое значение**: double
      * цена продукта.

**3. Класс Enterprise**

**Назначение:** Класс Enterprise предназначен для представления общей информации о предприятии, включая имя, адрес, количество сотрудников и годовой доход.

**Атрибуты:**

* **name** (тип: String):
  + **Назначение**: хранит имя предприятия.
  + **Обоснование выбора типа**: хранение текстовой информации в виде строки.
  + **Доступ**: приватный.
* **address** (тип: String):
  + **Назначение**: хранит адрес предприятия.
  + **Обоснование выбора типа**: также текстовая информация.
  + **Доступ**: приватный.
* **numberOfEmployees** (тип: int):
  + **Назначение**: хранит количество сотрудников в предприятии.
  + **Обоснование выбора типа**: количество может быть представлено целым числом.
  + **Доступ**: приватный.
* **annualRevenue** (тип: double):
  + **Назначение**: хранит годовой доход предприятия.
  + **Обоснование выбора типа**: представляет собой денежную сумму.
  + **Доступ**: приватный.

**Методы:**

* **Конструктор**:
  + Enterprise(String name, String address, int numberOfEmployees, double annualRevenue)
    - **Назначение**: создает объект Enterprise с указанными значениями для всех атрибутов.
* **Методы доступа и изменения**:
  + String getName(), String getAddress(), int getNumberOfEmployees(), double getAnnualRevenue()
    - **Назначение**: возвращают соответствующие значения атрибутов.
    - **Возвращаемые значения**: соответствующие типы данных.
  + void setName(String name), void setAddress(String address), void setNumberOfEmployees(int numberOfEmployees), void setAnnualRevenue(double annualRevenue)
    - **Назначение**: устанавливают значения для соответствующих атрибутов.

**4. Класс ManufacturingEnterprise**

**Назначение:**

Класс ManufacturingEnterprise представляет собой конкретный тип предприятия, который включает информацию о сотрудниках и продуктах.

**Атрибуты:**

* **employees** (тип: List<Employee>):
  + **Назначение**: хранит список сотрудников предприятия.
  + **Обоснование выбора типа**: List позволяет динамически управлять списком сотрудников, добавлять новых и манипулировать данными.
  + **Доступ**: приватный.
* **products** (тип: List<Product>):
  + **Назначение**: хранит список продуктов, производимых предприятием.
  + **Обоснование выбора типа**: аналогично списку сотрудников, это обеспечит гибкость управления.
  + **Доступ**: приватный.

**Методы:**

* **Конструкторы**:
  + ManufacturingEnterprise()
    - **Назначение**: создает объект ManufacturingEnterprise с пустыми списками сотрудников и продуктов.
  + ManufacturingEnterprise(String name, String address, int numberOfEmployees, double annualRevenue)
    - **Назначение**: создает объект с указанными значениями для атрибутов суперкласса и пустыми списками.
* **Методы добавления**:
  + void addEmployee(Employee employee)
    - **Назначение**: добавляет нового сотрудника в список employees.
    - **Входные параметры**: Employee employee
      * объект сотрудника.
    - **Алгоритм реализации**:
      1. Проверить, что employee не равен null.
      2. Если проверка прошла успешно, добавить сотрудника в список employees.
  + void addProduct(Product product)
    - **Назначение**: добавляет новый продукт в список products.
    - **Входные параметры**: Product product
      * объект продукта.
    - **Алгоритм реализации**:
      1. Проверить, что product не равен null.
      2. Если проверка прошла успешно, добавить продукт в список products.
* **Методы доступа**:
  + List<Employee> getEmployees()
    - **Назначение**: возвращает список сотрудников предприятия.
    - **Возвращаемое значение**: List<Employee>.
  + List<Product> getProducts()
    - **Назначение**: возвращает список продуктов предприятия.
    - **Возвращаемое значение**: List<Product>.

**5.Класс HelloApplication**

**Назначение:** Класс HelloApplication является точкой входа в приложение на JavaFX. Он отвечает за первоначальную настройку и отображение пользовательского интерфейса.

**Методы:**

* **start(Stage primaryStage)**:
  + **Назначение**: инициализирует и отображает графический интерфейс приложения.
  + **Входные параметры**:
    - Stage primaryStage: основной контейнер для сцены приложения.
  + **Алгоритм реализации**:
    1. Использует FXMLLoader для загрузки файла разметки интерфейса (hello-view.fxml).
    2. Создает Scene с загруженным корневым элементом.
    3. Устанавливает заголовок окна.
    4. Отображает окно.
  + **Исключения**: Throws Exception для обработки любых ошибок при загрузке интерфейса.
* **main(String[] args)**:
  + **Назначение**: метод, запускающий приложение на JavaFX.
  + **Алгоритм реализации**: вызывает метод launch(args), который отвечает за запуск JavaFX приложения.
  1. **Класс MainController**

**Назначение:** Класс MainController управляет взаимодействием между пользовательским интерфейсом и бизнес-логикой приложения. Он обрабатывает события и обновляет интерфейс в ответ на действия пользователя.

**Методы:**

* **private void clearFields()**:
  + **Назначение**: очищает текстовые поля на форме.
  + **Алгоритм реализации**:
    - Вызывает метод clear() на каждом текстовом поле, чтобы удалить его содержимое.
* **private void showAlert(String title, String content)**:
  + **Назначение**: отображает предупреждающее сообщение пользователю.
  + **Входные параметры**:
    - String title: заголовок окна предупреждения.
    - String content: текст сообщения.
  + **Алгоритм реализации**:
    1. Создает объект Alert типа WARNING.
    2. Устанавливает заголовок и содержание.
    3. Показывает окно предупреждения и ждет, пока пользователь его закроет.
* **private void saveEnterprises()**:
  + **Назначение**: сохраняет данные о предприятиях в файл.
  + **Алгоритм реализации**:
    1. Использует ObjectOutputStream и FileOutputStream для создания потока для записи в файл enterprises.dat.
    2. Записывает список enterprises в файл.
    3. Обрабатывает возможные исключения IOException и выводит стек ошибок.

3.2 Алгоритм реализации методов

Эти методы включают добавление сотрудников и продуктов. Мы также рассмотрим методы, ответственные за очистку полей и отображение предупреждений.

Эти методы обеспечивают обработку информации в приложении, позволяя добавлять новых сотрудников и продукты, управлять пользовательским интерфейсом. Они обеспечивают основную функциональность, необходимую для работы с данными о предприятиях.

**Класс ManufacturingEnterprise**

**Методы, выполняющие обработку информации:**

* **void addEmployee(Employee employee)**:
  1. Принимает объект Employee как параметр.
  2. Проверяет, не является ли employee null.
  3. Если объект валиден, добавляет employee в список employees, что увеличивает количество сотрудников предприятия.
* **void addProduct(Product product)**:
  1. Принимает объект Product как параметр.
  2. Проверяет, не является ли product null.
  3. Если объект валиден, добавляет product в список products, что увеличивает ассортимент продукции предприятия.
* **List<Employee> getEmployees()**:
  1. Возвращает текущий список сотрудников employees.
* **List<Product> getProducts()**:
  1. Возвращает текущий список продуктов products.

**2. Класс MainController**

**Методы, выполняющие обработку информации:**

* **private void clearFields()**:
  1. Вызывает метод clear() для каждого текстового поля формы (например, nameField, addressField, и т.д.).
  2. Очищает все поля ввода, что позволяет пользователю начать ввод новых данных без остатков предыдущих значений.
  3. Эффективно подготавливает форму для нового ввода, улучшая пользовательский интерфейс.
* **private void showAlert(String title, String content)**:
  1. Создает объект Alert с предупреждающим типом.
  2. Устанавливает заголовок и содержимое предупреждения, переданные в параметры.
  3. Вызывает showAndWait(), что отображает предупреждение и останавливает выполнение кода до закрытия окна пользователем.
  4. Помогает информировать пользователя о возможных ошибках или предупреждениях, обеспечивая лучшую интерактивность.
* **private void saveEnterprises()**:
  1. Использует ObjectOutputStream для записи текущего состояния списка предприятий (enterprises) в файл enterprises.dat.
  2. Оборачивает поток в блок try-with-resources, что гарантирует автоматическое закрытие потока после завершения записи.
  3. Вызывает метод writeObject(), чтобы сериализовать список предприятий и сохранить его в файл.
  4. Обрабатывает возможные исключения ввода-вывода, выводя стек вызовов в случае ошибок и обеспечивая стабильность приложения.

ЗАКЛЮЧЕНИЕ

В ходе реализации курсового проекта «Сведения о промышленных предприятиях» была разработана функциональная программа, сочетающая в себе удобный графический интерфейс и инструментарий для работы с данными о предприятиях. Приложение позволяет пользователям эффективно заполнять, хранить и просматривать информацию о различных промышленных компаниях, что является важным аспектом для повышения качества управления и принятия решений в реальном времени.

Были подробно рассмотрены основные компоненты приложения, включая архитектуру кода, диаграммы классов и интерфейсные элементы. Проект продемонстрировал, как использование языка Java и платформы JavaFX может значительно упростить разработку сложных приложений. Также важным результатом является возможность дальнейшего улучшения приложения: планируется добавить функции редактирования и удаления записей, расширить функционал по работе с данными, а также интеграцию с базами данных для надежного хранения информации.

Таким образом, данный курсовой проект не только достиг поставленных целей, но и заложил основу для будущих усовершенствований, что подтверждает его актуальность и значимость в современном мире управления промышленными предприятиями.

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1. Бурков А. В., Сидорцов Д. Н. Разработка мобильных приложений: учебное пособие. М.: Издательство "Диалог-МИФИ", 2020. 300 с.
2. Гридин В. И., Емельянов А. И. Проектирование и развитие баз данных. М.: Издательство Юрайт, 2021. 280 с.
3. Князев И. А. Разработка веб-приложений с использованием технологий HTML5 и CSS3. СПб.: Питер, 2022. 320 с.
4. Ильин П. А., Пантелеев В. А. Программирование на Python для предпринимателей. Екатеринбург: Издательство Уральского университета, 2021. 240 с.
5. Петрова М. Ю. Изучение и применение подходов к разработке web-приложений для бизнеса. Казань: Казанский федеральный университет, 2023.264 с.
6. Куликов А. И. Современные методы автоматизации процессов предприятия. М.: Наука, 2022. 312 с.
7. Панфилова Н. В. Информационные технологии в управлении промышленными предприятиями. М.: Инфра-М, 2021. 224 с.
8. Федорова Т. А. Основы проектирования интерфейсов для мобильных приложений. СПб.: БХВ-Петербург, 2023. 240 с.
9. Смирнов А. Л., Кузнецов С. С. Основы работы с RESTful API: практическое руководство. М.: Эксмо, 2020. 192 с.
10. Лебедев А. Н. Создание и управление базами данных для мобильных приложений. М.: ВАК, 2022. 350 с.

ПРИЛОЖЕНИЕ 1  
 Листинг программы.

Enterprise

package org.example.courseworkalexander;

import java.io.Serializable;

public class Enterprise implements Serializable {

private static final long serialVersionUID = 1L; // Версия сериализации

private String name;

private String address;

private int numberOfEmployees;

private double annualRevenue;

public Enterprise(String name, String address, int numberOfEmployees, double annualRevenue) {

this.name = name;

this.address = address;

this.numberOfEmployees = numberOfEmployees;

this.annualRevenue = annualRevenue;

}

// Геттеры

public String getName() {

return name;

}

public String getAddress() {

return address;

}

public int getNumberOfEmployees() {

return numberOfEmployees;

}

public double getAnnualRevenue() {

return annualRevenue;

}

// Сеттеры

public void setName(String name) {

this.name = name;

}

public void setAddress(String address) {

this.address = address;

}

public void setNumberOfEmployees(int numberOfEmployees) {

this.numberOfEmployees = numberOfEmployees;

}

public void setAnnualRevenue(double annualRevenue) {

this.annualRevenue = annualRevenue;

}

}

MainController

private void clearFields() {

nameField.clear();

addressField.clear();

employeesField.clear();

revenueField.clear();

employeeNameField.clear();

employeePositionField.clear();

productNameField.clear();

productPriceField.clear();

}

private void showAlert(String title, String content) {

Alert alert = new Alert(Alert.AlertType.WARNING);

alert.setTitle(title);

alert.setHeaderText(null);

alert.setContentText(content);

alert.showAndWait();

}

private void saveEnterprises() {

try (ObjectOutputStream oos = new ObjectOutputStream(new FileOutputStream("enterprises.dat"))) {

oos.writeObject(new ArrayList<>(enterprises));

} catch (IOException e) {

e.printStackTrace();

}

}

}

ManufacturingEnterprise

package org.example.courseworkalexander;

import java.io.Serializable;

import java.util.ArrayList;

import java.util.List;

public class ManufacturingEnterprise extends Enterprise implements Serializable {

private static final long serialVersionUID = 1L; // Версия сериализации

private List<Employee> employees = new ArrayList<>();

private List<Product> products = new ArrayList<>();

// Конструктор по умолчанию

public ManufacturingEnterprise() {

super("", "", 0, 0.0); // Вызов конструктора суперкласса

this.employees = new ArrayList<>();

this.products = new ArrayList<>();

}

public ManufacturingEnterprise(String name, String address, int numberOfEmployees, double annualRevenue) {

super(name, address, numberOfEmployees, annualRevenue);

this.employees = new ArrayList<>();

this.products = new ArrayList<>();

}

public void addEmployee(Employee employee) {

employees.add(employee);

}

public void addProduct(Product product) {

products.add(product);

}

public List<Employee> getEmployees() {

return employees;

}

public List<Product> getProducts() {

return products;

}

}

Employee

package org.example.courseworkalexander;

import java.io.Serializable;

public class Employee implements Serializable {

private static final long serialVersionUID = 1L; // Версия сериализации

private String name;

private String position;

// Конструктор по умолчанию

public Employee() {

this.name = "";

this.position = "";

}

public Employee(String name, String position) {

this.name = name;

this.position = position;

}

public String getName() {

return name;

}

public String getPosition() {

return position;

}

}

Product

package org.example.courseworkalexander;

import java.io.Serializable;

public class Product implements Serializable {

private static final long serialVersionUID = 1L; // Версия сериализации

private String name;

private double price;

// Конструктор по умолчанию

public Product() {

this.name = "";

this.price = 0.0;

}

public Product(String name, double price) {

this.name = name;

this.price = price;

}

public String getName() {

return name;

}

public double getPrice() {

return price;

}

}

HelloAplication

package org.example.courseworkalexander;

import javafx.application.Application;

import javafx.fxml.FXMLLoader;

import javafx.scene.Parent;

import javafx.scene.Scene;

import javafx.stage.Stage;

import java.util.Objects;

public class HelloApplication extends Application {

@Override

public void start(Stage primaryStage) throws Exception{

Parent root = FXMLLoader.load(Objects.requireNonNull(getClass().getResource("/org/example/courseworkalexander/hello-view.fxml")));

Scene scene = new Scene(root);

primaryStage.setTitle("Сведения о промышленных предприятиях");

primaryStage.setScene(scene);

primaryStage.show();

}

public static void main(String[] args) {

launch(args);

}

}

FXML

<?xml version="1.0" encoding="UTF-8"?>

<?import javafx.geometry.Insets?>

<?import javafx.scene.control.Button?>

<?import javafx.scene.control.Label?>

<?import javafx.scene.control.TableColumn?>

<?import javafx.scene.control.TableView?>

<?import javafx.scene.control.TextField?>

<?import javafx.scene.layout.HBox?>

<?import javafx.scene.layout.VBox?>

<VBox maxHeight="-Infinity" maxWidth="-Infinity"

minHeight="-Infinity" minWidth="-Infinity" prefHeight="500.0" prefWidth="600.0" spacing="20.0"

xmlns="http://javafx.com/javafx/23.0.1" xmlns:fx="http://javafx.com/fxml/1"

fx:controller="org.example.courseworkalexander.MainController">

<padding>

<Insets topRightBottomLeft="10"/>

</padding>

<children>

<Label text="Промышленные предприятия"/>

<TableView fx:id="enterpriseTable" VBox.vgrow="ALWAYS">

<columns>

<TableColumn fx:id="nameColumn" text="Название"/>

<TableColumn fx:id="addressColumn" text="Адрес"/>

<TableColumn fx:id="employeesColumn" text="Количество сотрудников"/>

<TableColumn fx:id="revenueColumn" text="Годовой доход"/>

</columns>

</TableView>

<HBox spacing="10">

<children>

<TextField fx:id="nameField" promptText="Название"/>

<TextField fx:id="addressField" promptText="Адрес"/>

<TextField fx:id="employeesField" promptText="Сотрудники"/>

<TextField fx:id="revenueField" promptText="Доход"/>

<Button fx:id="addButton" text="Добавить" onAction="#handleAddButton"/>

<Button fx:id="editButton" text="Редактировать" onAction="#handleEditButton"/>

<Button fx:id="deleteButton" text="Удалить" onAction="#handleDeleteButton"/>

</children>

</HBox>

<HBox spacing="10">

<children>

<Label text="Добавить сотрудника"/>

<TextField fx:id="employeeNameField" promptText="Имя сотрудника"/>

<TextField fx:id="employeePositionField" promptText="Должность"/>

<Button text="Добавить сотрудника" onAction="#handleAddEmployeeButton"/>

</children>

</HBox>

<HBox spacing="10">

<children>

<Label text="Добавить продукт"/>

<TextField fx:id="productNameField" promptText="Название продукта"/>

<TextField fx:id="productPriceField" promptText="Цена"/>

<Button text="Добавить продукт" onAction="#handleAddProductButton"/>

</children>

</HBox>

<HBox spacing="10">

<children>

<Button text="Просмотреть сотрудников" onAction="#handleViewEmployeesButton"/>

<Button text="Просмотреть продукты" onAction="#handleViewProductsButton"/>

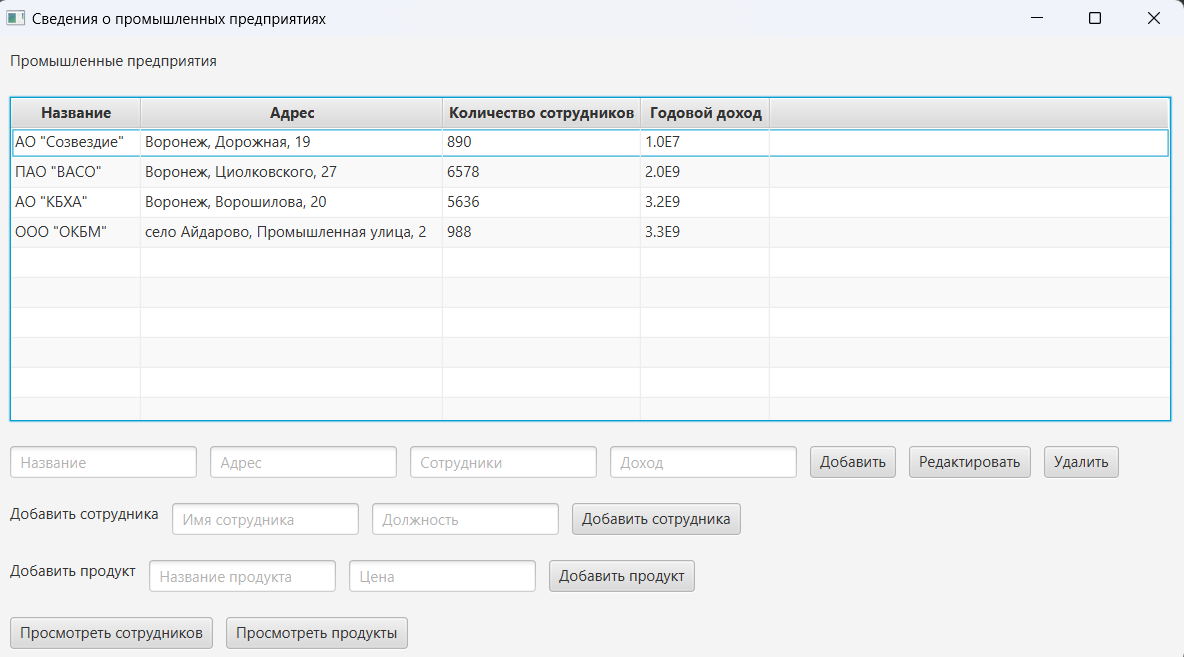
</children>

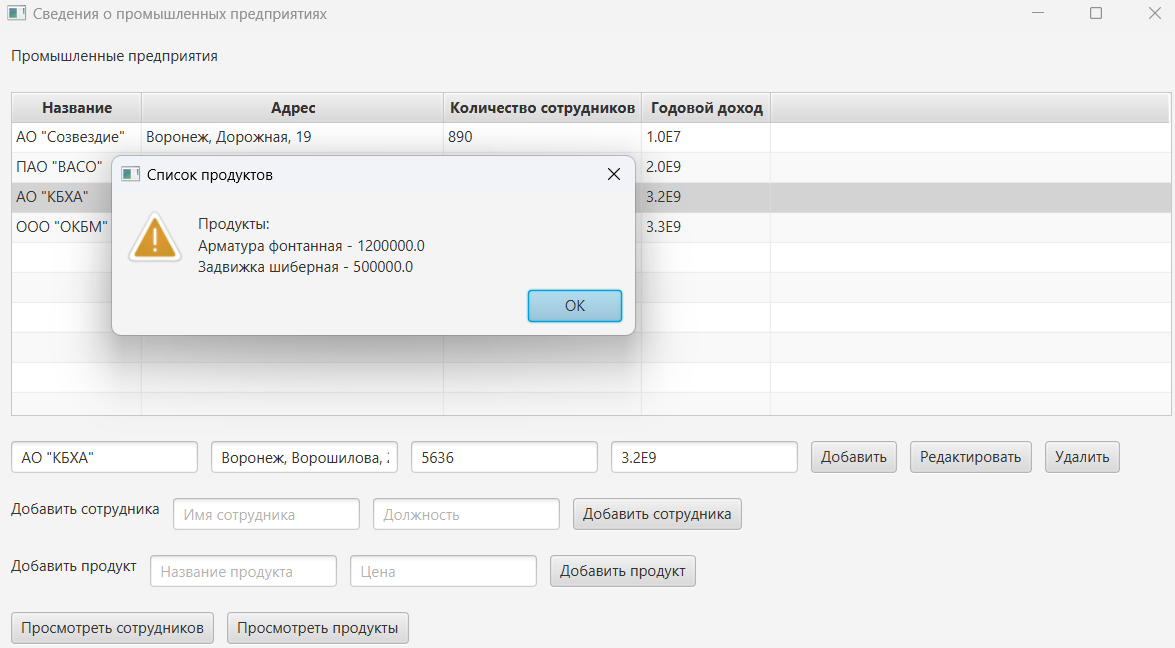
</HBox>

</children>

</VBox>

ПРИЛОЖЕНИЕ 2

 Скриншоты рабочих окон программы в различных режимах работы.

Интерфейс программы при запуске

Список продуктов на предприятии АО «КБХА»