

**Содержание**

[Введение 4](#_Toc165727209)

[1. Анализ предметной области 5](#_Toc165727210)

[1.1 Описание предметной области 5](#_Toc165727211)

[1.2 Постановка задачи 8](#_Toc165727212)

[2. Концептуальное и логическое проектирование 9](#_Toc165727213)

[2.1 Сущности предметной области 9](#_Toc165727214)

[1. Обоснование выбора концепций и механизмов ООП, необходимых для моделирования предметной области 12](#_Toc165727215)

[2. Проектирование и описание пользовательского интерфейса 15](#_Toc165727216)

[3. Описание результатов 18](#_Toc165727217)

[Заключение 19](#_Toc165727218)

[Список использованной литературы 20](#_Toc165727219)

[Приложение 1 3](#_Toc165727220)

# Введение

В современном мире розничная торговля играет значительную роль в экономике. Она обеспечивает население товарами и услугами, способствуя удовлетворению повседневных потребностей. Однако для успешного функционирования розничных торговых предприятий необходимо эффективно управлять продажами товаров. В этом контексте проектирование и реализация базы данных для учёта продажи товаров становится актуальной задачей.

Целью данной курсовой работы является проектирование и реализация базы данных для учёта продажи товаров в продуктовом магазине. В ходе работы будут рассмотрены особенности розничной торговли, требования к информационным системам и методы проектирования баз данных. Также будет проведён анализ существующих подходов к проектированию и реализации баз данных, выбраны подходящие инструменты и технологии для решения поставленной задачи.

Для достижения поставленной цели необходимо решить следующие задачи:

* Изучить особенности розничной торговли и требования к информационным системам.
* Разработать модель базы данных для учёта продажи товаров в продуктовом магазине.
* Выбрать подходящие инструменты и технологии для реализации базы данных.
* Реализовать спроектированную базу данных и протестировать ее.

# 1. Анализ предметной области

## 1.1 Описание предметной области

Существуют заводы, производящие продукты, продаваемые в магазине. Продукты от производителей в магазин доставляют компании-поставщики. Под поставкой понимается партия товара, привезенная в одном грузовом автомобиле. При поставке товаров в магазин составляются товарные накладные для описания поставленных товаров.

В накладной ТОРГ-12 предусматриваются все необходимые поля для оформления поставки.

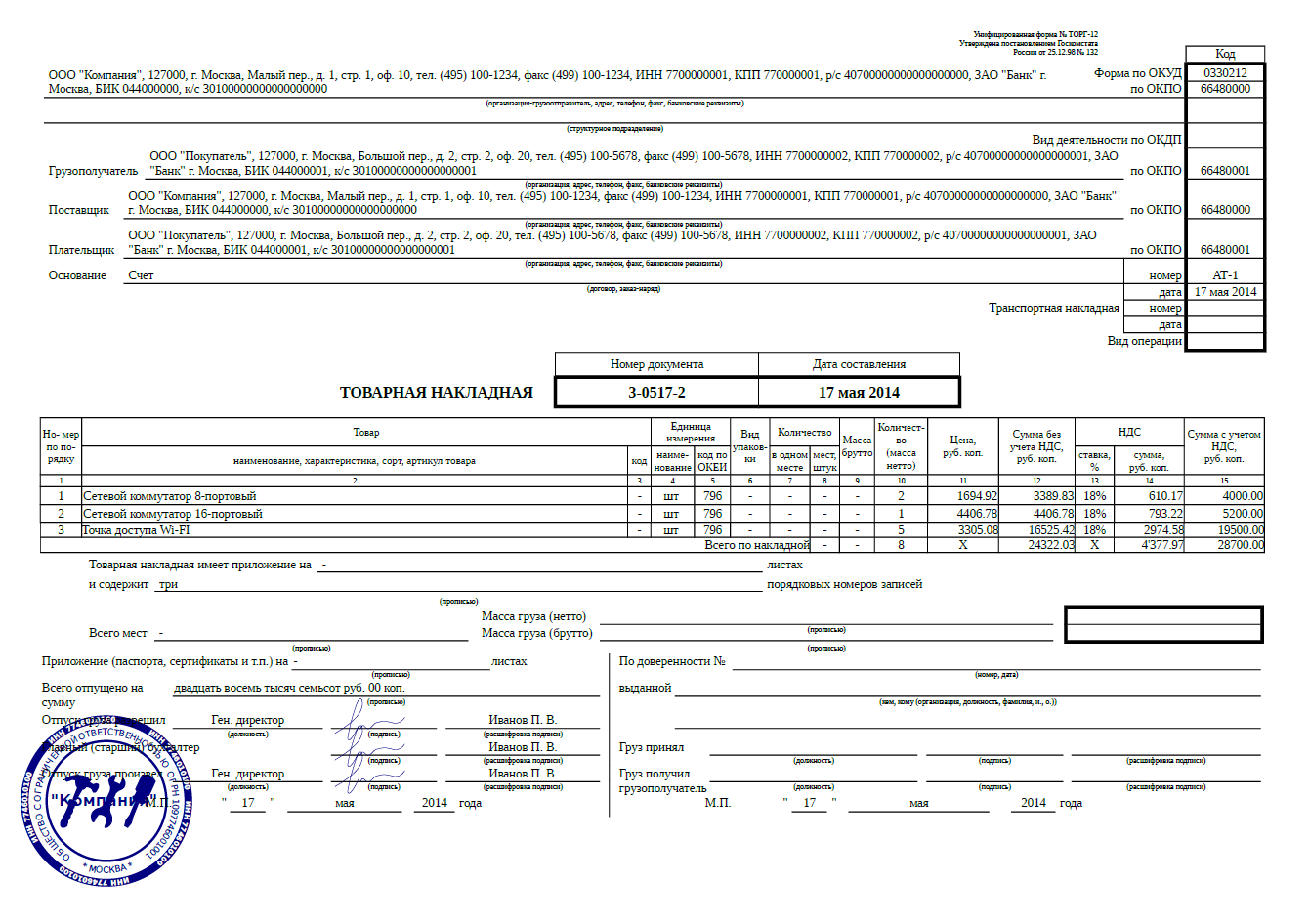


Рисунок 1 – пример заполненной накладной ТОРГ-12

В данной накладной учитываются следующие данные:

* Поставщик
* Получатель
* Дата поставки
* Наименования товаров
* Количества товара
* Цена товара до и после добавления НДС

Каждый товар имеет фиксированный срок годности и наценку.

При покупке товара печатается чек с уникальным номером. В чеке перечислены товары, их количества, стоимости, суммарная цена покупки.

Также необходимо вести учёт списанных товаров. Товары с истекшим сроком годности необходимо списывать.

Срок годности считается истекшим, если в дату на товаре, указанную как крайнюю для употребления, он не был куплен до закрытия магазина.

При списании товаров формируется акт о списании товара, например, ТОРГ-16.

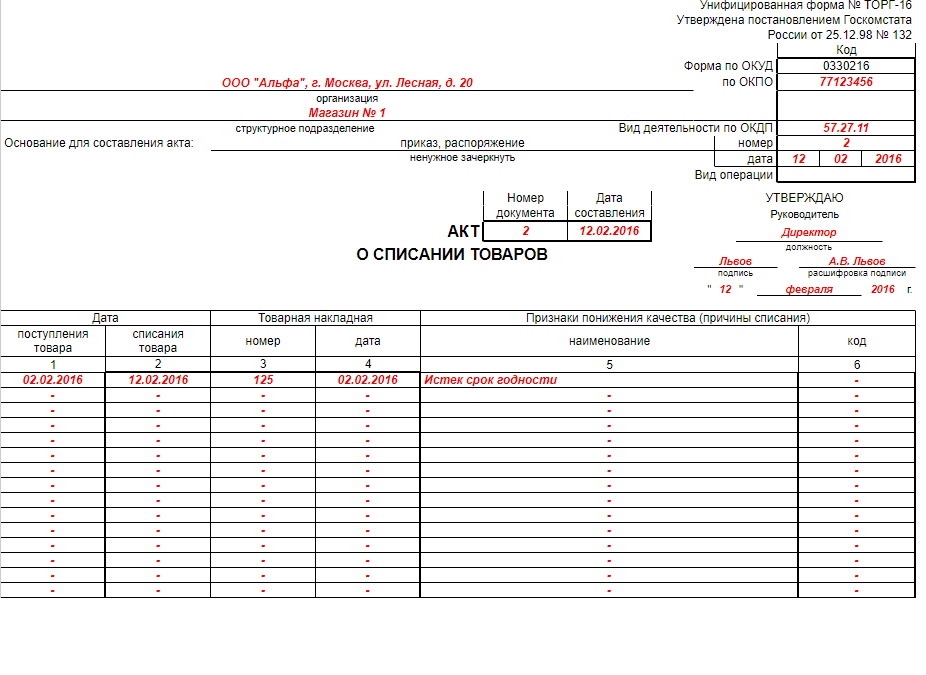


Рисунок 2 – пример заполненного акта о списании товаров ТОРГ-16. Лицевая сторона

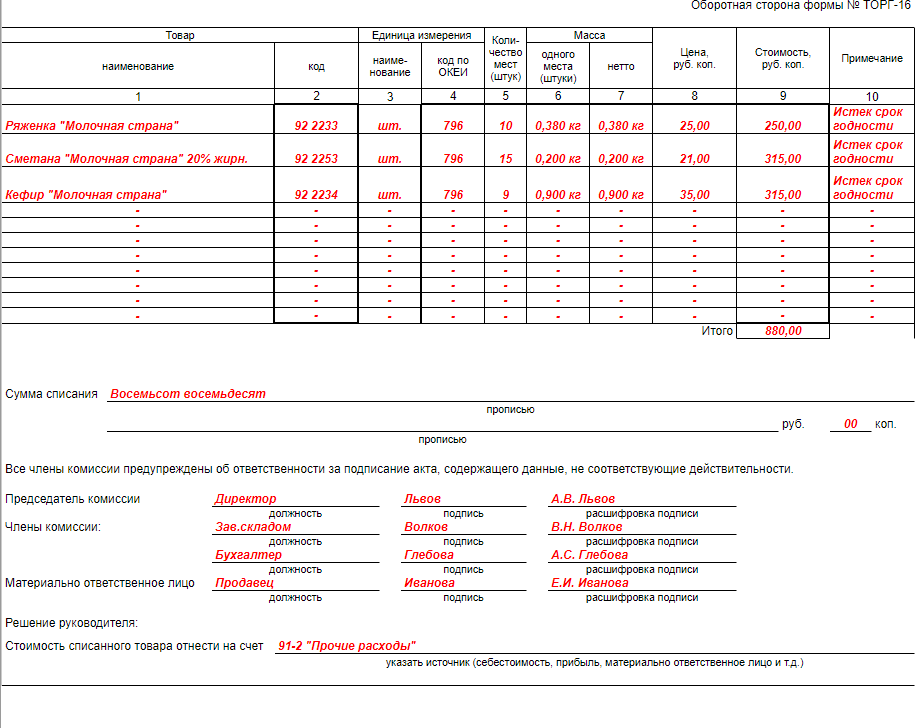


Рисунок 3 – пример заполненного акта о списании товаров ТОРГ-16. Оборотная сторона

В этой форме указаны следующие данные:

* Дата составления акта о списании товаров
* Дата поставки и списания товаров
* Товарная накладная, по которой сверялась поставка товара
* Наименования товаров
* Количество товаров
* Масса товаров
* Суммарная стоимость списанных товаров

## 1.2 Постановка задачи

Таким образом, исходя из описания предметной области, можно сформировать список требований к разрабатываемой базе данных:

* Необходимо хранить информацию обо всех поставщиках, с которыми работает магазин
* Необходимо вести учёт поставок
* При учёте поставок необходимо сохранять информацию о дате поставки, количестве и стоимости определенного товара
* Необходимо вести учёт списанных товаров
* При учёте списанных товаров необходимо сохранять информацию о поставке, с которой поступил списываемый товар, количестве и стоимости списываемого товара
* Необходимо вести учёт покупок товара
* При учёте покупок сохранять информацию о поставках, с которыми прибыли купленные товары, количестве и стоимости купленных товаров

К дополнительным требованиям можно отнести наличие в базе данных триггеров, призванных упростить ее заполнение.

# 2. Концептуальное и логическое проектирование

## 2.1 Сущности предметной области

Сущность предметной области — это абстракция реального объекта, группы однотипных объектов или концептуального понятия предметной области, характеризуемая набором существенных характеристик (данных, атрибутов), связанных с проектируемой программной системой.

В рассматриваемой предметной области можно выделить следующие сущности:

* Поставщик
* Поставка
* Товар
* Продажа
* Списание товара

Это основные объекты и абстракции реального мира, необходимые для реализации проектируемой базы данных. Каждая сущность содержит особый набор атрибутов. Определим атрибуты для каждой из перечисленных сущностей.

Поставщик характеризуется своим местоположением и наименованием компании. Остальные характеристики поставщика в рамках рассматриваемой предметной области не важны, так что именно эти характеристики и будут являться атрибутами сущности.

Товар характеризуется своим названием, массой, видом упаковки (например, банка либо бутылка для напитков), сроком годности. Так же в числе атрибутов будет содержаться наценка магазина на товар, которую определяет магазин.

Поставка будет содержать информацию о количестве поставленных товаров, а также дату.

Продажа будет характеризоваться датой, временем и количеством проданных товаров.

В сущности «Списание товара» свойствами будут являться дата списания и количество списанного товара.

Основываясь на описаниях предметной области и сущностей построим ER-диаграмму в нотации Чена для визуализации концептуальной модели.

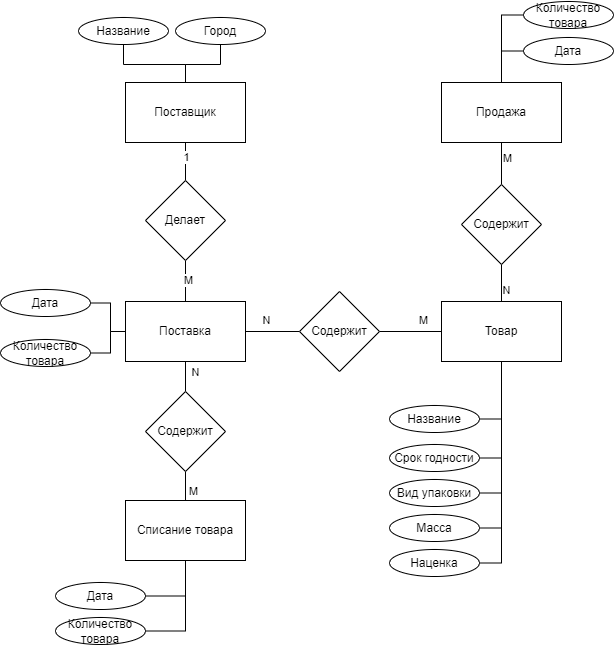


Рисунок 4 – ER-Диаграмма в нотации Чена.

Разрыв

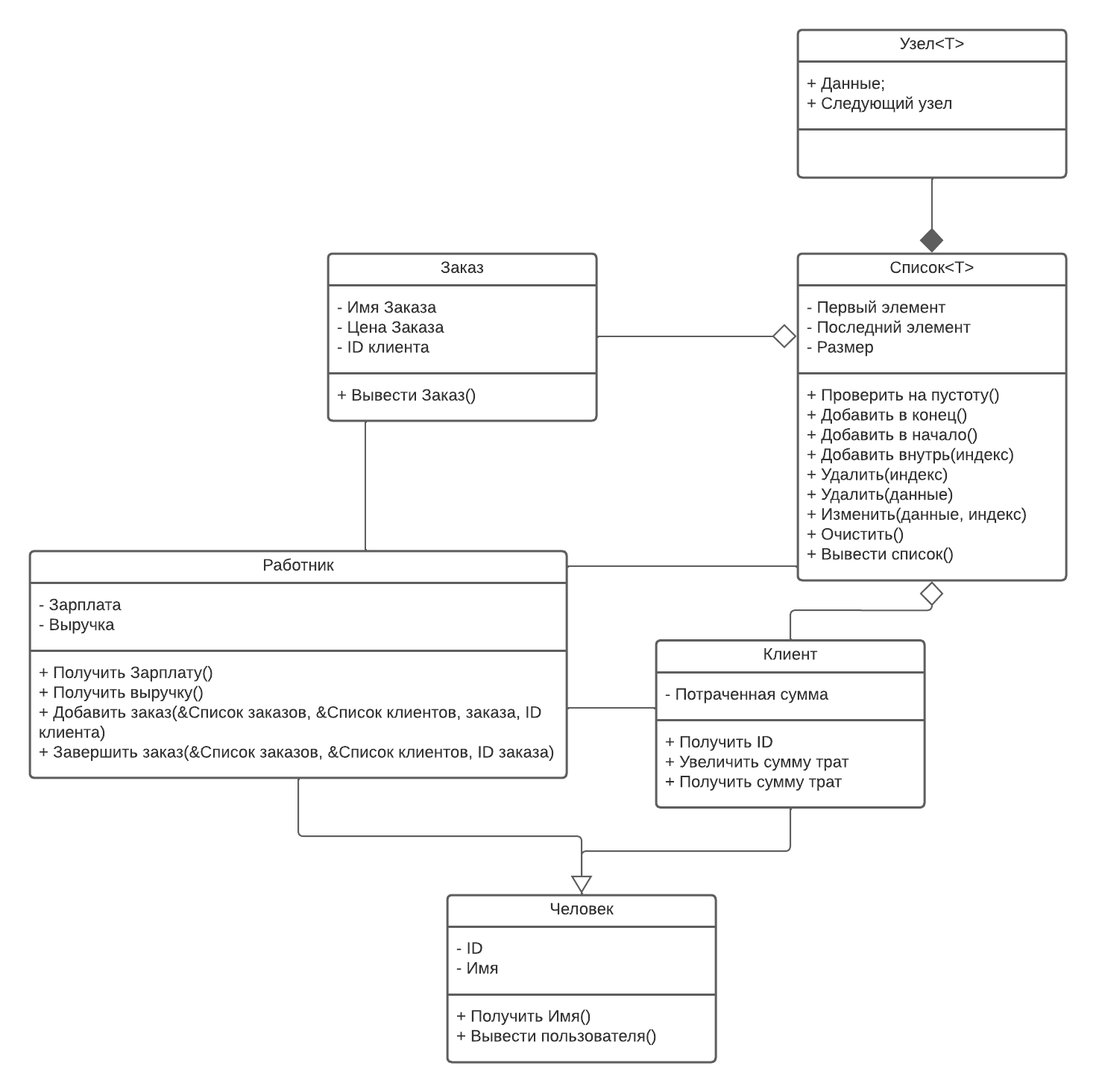


Рисунок 1 – Логическая диаграмма классов

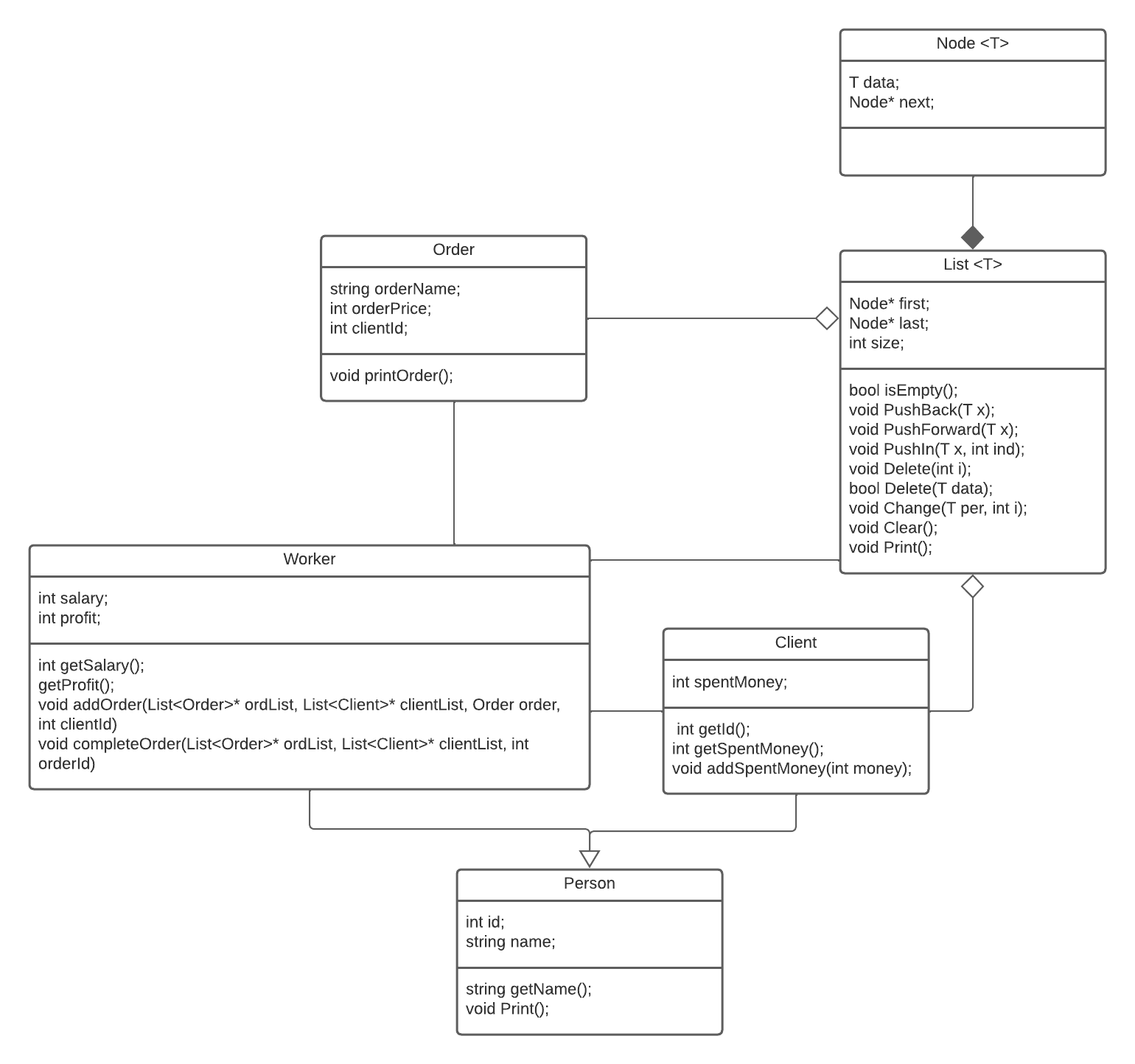


Рисунок 2 – Физическая диаграмма классов

# Обоснование выбора концепций и механизмов ООП, необходимых для моделирования предметной области

Данные о заказах и клиентах будут храниться в связных списках. Они позволяют выделить необходимый объем памяти во время выполнения программы, что дает возможность организовать корректную работу системы. Связные списки бывают следующих разновидностей:

По данным о других узлах списки подразделяются на:

* Односвязные. Списки, в узлах которых хранятся данные и указатель только на следующий узел. В таких списках последний элемент должен указывать на NULL, чтобы его можно было идентифицировать при обходе.
* Двусвязные. Списки, в узлах которых помимо данных хранятся указатели как на следующий, так и на предыдущий узел списка. Позволяют идти по списку не только с начала, но и с конца.

Так же списки делятся на линейные и кольцевые. В линейных списках последний элемент указывает на NULL, и элементы идут друг за другом в линейном порядке. В кольцевом списке нету последнего элемента, в результате обхода по такому списку происходит возврат в первый элемент, так называемую «голову» списка.

Для реализации поставленной задачи будет достаточно использования линейного односвязного списка.

Так же необходимо применение следующих механизмов ООП:

* различные виды конструкторов;
* наследование;
* инкапсуляция;
* полиморфизм;
* шаблоны классов;
* исключения;
* перегрузка операторов;
* дружественные функции;
* RTTI.

При разработке программы были созданы классы, работа с которыми во многом облегчается конструкторами. Конструктор класса — это специальная функция, которая автоматически вызывается сразу после создания объекта этого класса. Он не имеет типа возвращаемого значения и должен называться также, как класс, в котором он находится. Конструкторы повсеместно используются при реализации программ в парадигме ООП. Перегрузка конструкторов позволяет создать множество конструкторов, которые будут принимать разные входные значения, от чего может различаться и их реализация. Все классы в программе имеют свои конструкторы для более удобного создания объектов.

В программе так же используется наследование. Наследование - это механизм создания нового класса на основе уже существующего. При этом к существующему классу могут быть добавлены новые элементы (данные и функции), либо существующие функции могут быть изменены. Наследование содействует повторному использованию атрибутов и методов класса, а значит, делает процесс разработки ПО более эффективным. При реализации поставленной задачи использован суперкласс Person, от которого наследуются классы Worker и Client.

В создании объектно-ориентированных программ есть множество плюсов, один из которых – возможность программировать отдельные модули одной программы, что позволяет эффективно распределить работу между большой командой разработчиков. Однако, при написании подобных модулей, зачастую разработчик хочет оградить часть своего кода от неразумного использования, в чем помогает инкапсуляция. Инкапсуляция – это сокрытие от пользователя реализации компонентов программы. Чтобы избежать нежелательных изменений там, где они не были предусмотрены разработчиком, информация закрывается от использования, и доступ к ней можно получить только через реализованные ранее методы. В нашей программе инкапсуляция будет защищать приватные поля классов от нежелательных изменений.

Полиморфизм – еще одна особенность ООП. Он заключается в том, что можно использовать одно и то же имя функции с разными входными значениями, что позволяет изменять реализацию в зависимости от переданных параметров. На принципе полиморфизма в классе Person реализуем полиморфную функцию вывода информации об объекте в консоль и назовем ее Print, чтобы переопределить ее в дочерних классах и обращаться к ней из шаблона класса List без ошибок компиляции.

Возникают ситуации, когда необходимо проделать одни и те же действия с данными разных типов, что в давние времена означало бы, что нам придется писать реализацию программы для каждого типа данных отдельно. Эту проблему можно решить использованием шаблонов. Шаблон – это общая реализация для абстрактного типа данных. В поставленной задаче используются связные списки клиентов и заказов, что делает целесообразным создание шаблона класса, реализующего список. Это существенно уменьшит объем кода, а также в дальнейшем вносить правки в такой код будет на порядок легче. В нашей программе шаблоном класса является класс List, создающий связный список из экземпляров любого класса.

Зачастую, при написании программ возникает необходимость отлавливать ошибки, возникшие в коде. Ошибки, возникающие во время выполнения, можно отлавливать благодаря блоку try. В программе используем блок try, чтобы в нем анализировать наличие ошибок. В нашей программе такими ошибками будут неверный ID клиента или заказа. В случае возникновения ошибки выбросим исключение с помощью throw, и обработаем его в блоке catch.

Перегрузка операторов в C++ позволяет определить новое поведение для стандартных операторов языка, таких как +, -, \*, /, =, == и других. Это позволяет работать с пользовательскими типами данных так же, как с встроенными типами. В программе с помощью перегрузки операторов реализуем вывод объектов класса в поток.

В C++ дружественная функция является функцией, объявленной внутри класса, которая имеет доступ к приватным и защищенным членам этого класса. Дружественная функция может быть объявлена как внутри класса, так и вне его, с использованием ключевого слова friend. В программе реализуем с помощью дружественной функции вывод информации о заказе на экран.

RTTI (Run-Time Type Information) – это механизм, позволяющий определить тип данных переменной прямо во время выполнения программы. В C++ для динамической идентификации типов применяются операторы dynamic\_cast и typeid, для использования которых информацию о типах во время выполнения обычно необходимо добавить через опции компилятора при компиляции модуля. С помощью RTTI в программе при выводе информации из списка на экран будем определять тип данных элементов списка и выводить их соответствующим образом.

# Проектирование и описание пользовательского интерфейса

В ходе реализации программы было принято решение сделать интерфейс в консоли, чтобы сделать акцент на функциональной части библиотеки классов.

При запуске программы открывается меню, элементы которого позволяют в полной мере протестировать функции программы (Рисунок 3).

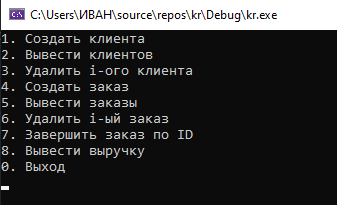


Рисунок 3 – Начальный экран

При выборе пункта 1 у пользователя запрашивается ввод информации о клиенте, такой как его имя и ID в системе (Рисунок 4).

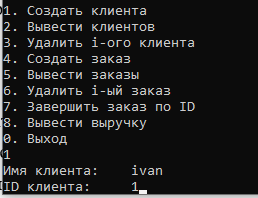


Рисунок 4 – Создание клиента

Можно запросить вывод всех клиентов списка, выбрав пункт номер 2. Выводится тип списка, далее имя и ID клиента, а после список его заказов. В случае отсутствия заказов, на месте списка возникает надпись «No orders» (Рисунок 5).

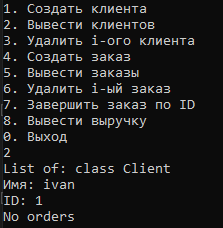


Рисунок 5 – Вывод списка клиентов

Пункт 3 меню отвечает за удаления из списка элемента с индексом i, при том, что элемент с индексом 0 считается первым (Рисунок 6).

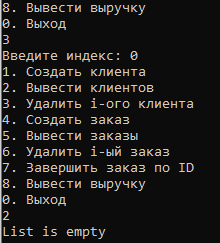


Рисунок 6 – Удаление из списка

Создание заказа происходит при выборе 4 пункта меню, и включает в себя привязку заказа к определенному клиенту (Рисунок 7).

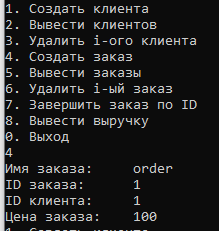


Рисунок 7 – Создание заказа

Завершение заказа осуществляется при выборе 7 пункта меню, и включает в себя удаление заказа из общего списка заказов, удаление из клиентского списка заказов и прибавки цены заказа к сумме выручки (Рисунок 8).

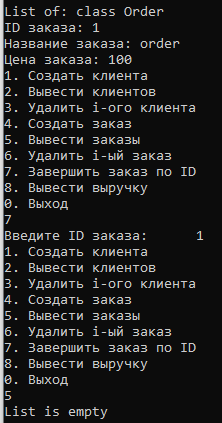


Рисунок 8 – Завершение заказа

После завершения заказа класс работника службы сохраняет сумму заработанных денег. Эту сумму можно увидеть, выбрав пункт 8 меню (Рисунок 9).

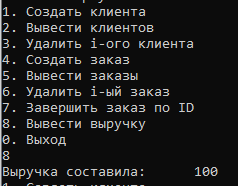


Рисунок 9 – Вывод выручки

# Описание результатов

В процессе создания моей курсовой работы я успешно разработал библиотеку классов, которая моделирует работу службы доставки. Кроме того, я разработал консольный интерфейс (Рисунок 3-9). Теперь полученную программу можно использовать для симуляции работы службы доставки.

В результате выполнения работы были достигнуты следующие цели:

1. выбраны методы решения;

2. выбраны структуры данных;

3. разработана структура программы;

4. разработаны алгоритмы решения для поставленной задачи;

5. реализована программа на языке С++;

6. выполнено описание пользовательского интерфейса.

# Заключение

В процессе разработки программы для курсовой работы мы изучили концепцию объектов и классов, а также ознакомились с методологией разработки классовых библиотек на языке C++. Мы освоили теоретические основы, которые мы применили на практике при написании программы. Таким образом, мы успешно достигли поставленной цели и выполнение задачи можно считать завершенным.

# Список использованной литературы

1. Структуры данных: связный список | habr [Электронный ресурс]. Режим доступа: [https://habr.com/ru/articles/717572](https://habr.com/ru/articles/717572/) (Дата обращения: 01.06.2023)
2. Динамическая идентификация типа данных| wikipedia [Электронный ресурс]. Режим доступа:

<https://ru.wikipedia.org/wiki/Динамическая_идентификация_типа_данных> (Дата обращения: 01.06.2023)

1. Инкапсуляция в C ++ | GeeksforGeeks [Электронный ресурс]. Режим доступа: https://www.geeksforgeeks.org/encapsulation-in-cpp/ (Дата обращения: 01.06.2023)
2. Наследование и полиморфизм | academy.yandex [Электронный ресурс]. Режим доступа: https://academy.yandex.ru/handbook/cpp/article/inheritance-and-polymorphism (Дата обращения: 01.06.2023)
3. Дружественные функции и классы | metanit.com [Электронный ресурс]. Режим доступа: https://metanit.com/cpp/tutorial/5.5.php (Дата обращения: 10.05.2023)

# Приложение 1

**Листинг файла «kr.cpp»  
(обязательное)**

#include <iostream>

#include <string>

using namespace std;

class Exeption

{

public:

Exeption(int code, string desc) : code(code), ErrorDescription(desc) {}

int code;

string ErrorDescription;

};

class Order;

template <typename T>

class Node

{

public:

Node(T \_data) : data(\_data) {}

T data;

Node<T>\* next = nullptr;

};

template <typename T>

class List

{

private:

Node<T>\* first = nullptr;

Node<T>\* last = nullptr;

int size = 0;

public:

bool isEmpty()

{

return (first == nullptr);

}

int getSize()

{

return size;

}

void PushBack(T x)

{

Node<T>\* per = new Node<T>(x);

size++;

if (isEmpty())

{

first = per;

last = per;

return;

}

last->next = per;

last = per;

}

void PushForward(T x)

{

Node<T>\* per = new Node<T>(x);

size++;

if (isEmpty())

{

first = per;

last = per;

return;

}

per->next = first;

first = per;

}

void PushIn(T x, int ind)

{

try

{

Node<T>\* per = new Node<T>(x);

Node<T>\* el = first;

Node<T>\* elb = nullptr;

if (ind < 0 || ind >= size)

{

throw Exeption(101, "Некорректный ID");

return;

}

size++;

int k = 0;

if (ind == 0)

{

per->next = el;

first = per;

return;

}

while (k < ind)

{

k++;

elb = el;

el = el->next;

}

if (el->next == nullptr)

{

elb->next = per;

per->next = el;

last = el;

return;

}

else

{

elb->next = per;

per->next = el;

return;

}

}

catch (Exeption exeption)

{

cout << "Код ошибки: " << exeption.code << endl;

cout << exeption.ErrorDescription << endl;

}

}

void Delete(int i)

{

try

{

if (i < 0 || i >= size)

{

throw Exeption(101, "Некорректный ID");

return;

}

int k = 0;

Node<T>\* el = first;

Node<T>\* elb = nullptr;

Node<T>\* ela = nullptr;

size--;

if (i == 0)

{

el = first->next;

delete first;

first = el;

return;

}

while (k < i)

{

k++;

elb = el;

el = el->next;

ela = el->next;

}

if (ela == nullptr)

{

last = elb;

elb->next = nullptr;

delete el;

return;

}

else

{

elb->next = ela;

delete el;

}

}

catch (Exeption exeption)

{

cout << "Код ошибки: " << exeption.code << endl;

cout << exeption.ErrorDescription << endl;

}

}

bool DeleteWithId(int id)

{

Node<T>\* el = first;

Node<T>\* last = first;

if (isEmpty())

{

return 1;

}

if (el->data->getId() == id)

{

first = el->next;

delete el;

return 0;

}

while (el->next != nullptr)

{

if (el->data->getId() == id)

{

last->next = el->next;

delete el;

return 0;

}

last = el;

el = el->next;

}

return 1;

}

void Change(T per, int i)

{

try

{

int k = 0;

Node<T>\* el = first;

if (i < 0 || i >= size)

{

throw Exeption(101, "Некорректный ID");

return;

}

while (k < i)

{

k++;

el = el->next;

}

el->data = per;

return;

}

catch (Exeption exeption)

{

cout << "Код ошибки: " << exeption.code << endl;

cout << exeption.ErrorDescription << endl;

}

}

void Clear()

{

Node<T>\* el = first;

while (el->next != nullptr)

{

delete el;

}

}

void Print()

{

Node<T>\* el = first;

if (isEmpty())

{

cout << "List is empty" << endl;

return;

}

string typeName = typeid(el->data).name();

cout << "List of: " + typeName << endl;

while (el != nullptr)

{

if constexpr (is\_same\_v<T, Order\*>)

{

Order ord = \*(el->data);

printOrder(ord);

}

else

{

el->data->Print();

cout << el->data->GetType() << endl;

}

el = el->next;

}

}

T\* Find(int id)

{

Node<T>\* el = first;

Node<T>\* last = first;

if (isEmpty())

{

return nullptr;

}

if (el->data->getId() == id)

{

return &el->data;

}

while (el->next != nullptr)

{

if (el->data->getId() == id)

{

return &el->data;

}

last = el;

el = el->next;

}

return nullptr;

}

};

class Person

{

protected:

string name;

int id;

public:

Person(string name, int id) : name(name), id(id) {}

virtual int getId()

{

return id;

}

virtual string getName()

{

return name;

}

virtual void Print()

{

cout << "Имя: " << name << endl;

cout << "ID: " << getId() << endl;

}

virtual string GetType()

{

return "Person";

}

string GetName()

{

return name;

}

};

class Order

{

int id;

string orderName;

int orderPrice;

int clientId;

public:

Order() {}

Order(string orderName, int orderPrice, int id) :

id(id),

orderName(orderName),

orderPrice(orderPrice) {}

friend void printOrder(const Order& order);

int getId()

{

return id;

}

string GetType()

{

return "Order";

}

string getName()

{

return orderName;

}

int getPrice()

{

return orderPrice;

}

int getClient()

{

return clientId;

}

void setClient(int id)

{

clientId = id;

}

};

class Client : public Person

{

int spentMoney = 0;

public:

Client(string name, int id) : Person(name, id) {}

int getId() override

{

return id;

}

string getName() override

{

return name;

}

int getSpentMoney()

{

return spentMoney;

}

void addSpentMoney(int money)

{

spentMoney += money;

}

void Print() override

{

cout << "Имя: " << this->name << endl;

cout << "ID: " << getId() << endl;

cout << "Потрачено денег: " << getSpentMoney() << endl;

}

string GetType() override

{

return "Client";

}

friend ostream& operator<<(ostream& os, const Client& obj) {

os << "Client(" << obj.id << ")";

return os;

}

};

class Worker : public Person

{

protected:

int salary;

int profit = 0;

public:

int getId() override

{

return id;

}

Worker(string name, int id, int Salary) : Person(name, id)

{

salary = Salary;

}

string GetType() override

{

return "Client";

}

int GetSalary()

{

return salary;

}

int GetProfit()

{

return profit;

}

void Print() override

{

cout << "Имя: " << name << endl;

cout << "ID: " << getId() << endl;

cout << "Зарплата: " << salary << endl;

}

friend ostream& operator<<(ostream& os, const Worker& obj) {

os << "Worker(" << obj.id << ")";

return os;

}

void addOrder(List<Order\*>\* ordList, List<Person\*>\* clientList, Order\* order, int clientId)

{

Person\*\* per = clientList->Find(clientId);

try

{

if (per == nullptr)

{

throw Exeption(101, "Некорректный ID");

}

}

catch (Exeption ex)

{

cout << "Код ошибки: " << ex.code << endl;

cout << ex.ErrorDescription << endl;

return;

}

order->setClient(clientId);

ordList->PushBack(order);

}

void completeOrder(List<Order\*>\* ordList, List<Person\*>\* clientList, int orderId)

{

Order\* order = \*ordList->Find(orderId);

try

{

if (order == nullptr)

{

throw Exeption(101, "Некорректный ID");

}

}

catch (Exeption ex)

{

cout << "Код ошибки: " << ex.code << endl;

cout << ex.ErrorDescription << endl;

return;

}

int clientId = order->getClient();

Person\*\* per = clientList->Find(clientId);

Client client = \*(Client\*)(\*per);

(\* ((Client\*\*)per))->addSpentMoney(order->getPrice());

profit += order->getPrice();

ordList->DeleteWithId(orderId);

}

};

void printOrder(const Order& order)

{

cout << "ID заказа: " << order.id << endl;

cout << "Название заказа: " << order.orderName << endl;

cout << "Цена заказа: " << order.orderPrice << endl;

}

int main()

{

setlocale(LC\_ALL, "ru");

List <Order\*> orders;

List <Person\*> clients;

Worker admin("Admin", 1, 1000);

string input;

while (true)

{

cout << "1. Создать клиента" << endl;

cout << "2. Вывести клиентов" << endl;

cout << "3. Удалить i-ого клиента" << endl;

cout << "4. Создать заказ" << endl;

cout << "5. Вывести заказы" << endl;

cout << "6. Удалить i-ый заказ" << endl;

cout << "7. Завершить заказ по ID" << endl;

cout << "8. Вывести выручку" << endl;

cout << "0. Выход" << endl;

cin >> input;

if (input == "1")

{

string name;

cout << "Имя клиента:\t";

cin >> name;

string id;

cout << "ID клиента:\t";

cin >> id;

try

{

stoi(id);

}

catch (const std::exception& e) {

cout << "Ошибка: " << e.what() << endl;

continue;

}

Client\* cl = new Client(name, stoi(id));

clients.PushBack(cl);

}

else if (input == "2")

{

clients.Print();

}

else if (input == "3")

{

cout << "Введите индекс:\t";

string i;

cin >> i;

try

{

stoi(i);

}

catch (const std::exception& e) {

cout << "Ошибка: " << e.what() << endl;

continue;

}

clients.Delete(stoi(i));

}

else if (input == "4")

{

string name;

cout << "Имя заказа:\t";

cin >> name;

string id;

cout << "ID заказа:\t";

cin >> id;

try

{

stoi(id);

}

catch (const std::exception& e) {

cout << "Ошибка: " << e.what() << endl;

continue;

}

cout << "ID клиента:\t";

string idc;

cin >> idc;

try

{

stoi(idc);

}

catch (const std::exception& e) {

cout << "Ошибка: " << e.what() << endl;

continue;

}

string price;

cout << "Цена заказа:\t";

cin >> price;

try

{

stoi(price);

}

catch (const std::exception& e) {

cout << "Ошибка: " << e.what() << endl;

continue;

}

Order\* ord = new Order(name, stoi(price), stoi(id));

admin.addOrder(&orders, &clients, ord, stoi(idc));

}

else if (input == "5")

{

orders.Print();

}

else if (input == "6")

{

cout << "Введите индекс:\t";

string i;

cin >> i;

try

{

stoi(i);

}

catch (const std::exception& e) {

cout << "Ошибка: " << e.what() << endl;

continue;

}

orders.Delete(stoi(i));

}

else if (input == "7")

{

cout << "Введите ID заказа:\t";

string id;

cin >> id;

try

{

stoi(id);

}

catch (const std::exception& e) {

cout << "Ошибка: " << e.what() << endl;

continue;

}

admin.completeOrder(&orders, &clients, stoi(id));

}

else if (input == "8")

{

cout << "Выручка составила:\t" << admin.GetProfit() << endl;

}

else if (input == "0")

{

return 0;

}

}

}