Министерство образования и науки Кыргызской Республики

Кыргызский государственный технический университет

им. И.Раззакова

Институт информационных технологий

Кафедра «Программное обеспечение компьютерных систем»

Направление: «Программная инженерия»

Дисциплина: «Объектно-Ориентированное Программирование»

**Курсовой проект**

**Тема: «Финансовый учёт»**

Выполнил: студент группы ПИ-3-21

Икрамов Абдул Муним

Проверил: Мусабаев Э.Б

**Бишкек 2024**

Оглавление

[**Введение** 2](#_Toc168063974)

[**Описание классов** 3](#_Toc168063975)

[**Диаграмма классов** 6](#_Toc168063976)

[**Листинг программы** 6](#_Toc168063977)

[**Тесты** 13](#_Toc168063978)

[**Заключение** 17](#_Toc168063979)

# 

# **Введение**

# В современном мире управление личными финансами становится все более важным аспектом нашей повседневной жизни. Отслеживание доходов, расходов и учет долгов помогает нам понять, как мы распоряжаемся нашими средствами, и принимать осознанные финансовые решения. Для облегчения этого процесса была разработана программа для управления финансами.

# Эта программа предназначена для помощи пользователям в эффективном управлении и контроле своих финансов. Она предоставляет удобный интерфейс для ввода и отслеживания финансовых операций, таких как доходы, расходы и долги. Пользователь может легко добавлять новые транзакции, просматривать текущий баланс и анализировать свои финансовые потоки.

# Программа обеспечивает сохранение истории транзакций в файл, что позволяет пользователям возвращаться к своим финансовым данным в любое время и делать анализ своих финансовых действий. Кроме того, она позволяет отслеживать долги и устанавливать сроки их погашения, обеспечивая более полный контроль над финансами.

# Введение этой программы в повседневную жизнь может помочь пользователям лучше управлять своими финансами, выстраивать более здоровые финансовые привычки и достигать своих финансовых целей.

# **Описание классов**

**Класс Date:**

Описание: Представляет дату в формате год-месяц-день.

Поля:

year: год

month: месяц

day: день

Методы:

getYear(): возвращает год

getMonth(): возвращает месяц

getDay(): возвращает день

**Класс Category:**

Описание: Представляет категорию расходов или доходов.

Поля:

name: название категории

Методы:

getName(): возвращает название категории

**Класс Expense:**

Описание: Представляет расход.

Поля:

amount: сумма расхода

date: дата расхода (объект класса Date)

category: категория расхода (объект класса Category)

description: описание расхода

Методы:

getAmount(): возвращает сумму расхода

getDate(): возвращает дату расхода

getCategory(): возвращает категорию расхода

getDescription(): возвращает описание расхода

**Класс Transaction:**

Описание: Представляет транзакцию.

Поля:

amount: сумма транзакции

date: дата транзакции (объект класса Date)

category: категория транзакции (объект класса Category)

Методы:

getAmount(): возвращает сумму транзакции

getDate(): возвращает дату транзакции

getCategory(): возвращает категорию транзакции

**Класс Budget:**

Описание: Представляет бюджет и управляет транзакциями.

Поля:

balance: текущий баланс

transactions: список транзакций (вектор объектов класса Transaction)

Методы:

addTransaction(): добавляет транзакцию и обновляет баланс

getBalance(): возвращает текущий баланс

setTransactions(): устанавливает список транзакций

addExpense(): добавляет расход и обновляет баланс

addIncome(): добавляет доход и обновляет баланс

getTransactionHistory(): возвращает историю транзакций

**Класс DebtManager:**

Описание: Управляет долгами.

Поля:

debts: список долгов (структура Debt, содержащая сумму и дату погашения)

Методы:

addDebt(): добавляет долг в список

getDebts(): возвращает список долгов

**Класс ReportManager:**

Описание: Управляет сохранением и загрузкой данных из файла.

Поля:

filename: имя файла для сохранения/загрузки

Методы:

saveData(): сохраняет данные в файл

loadData(): загружает данные из файла

# **Диаграмма классов**

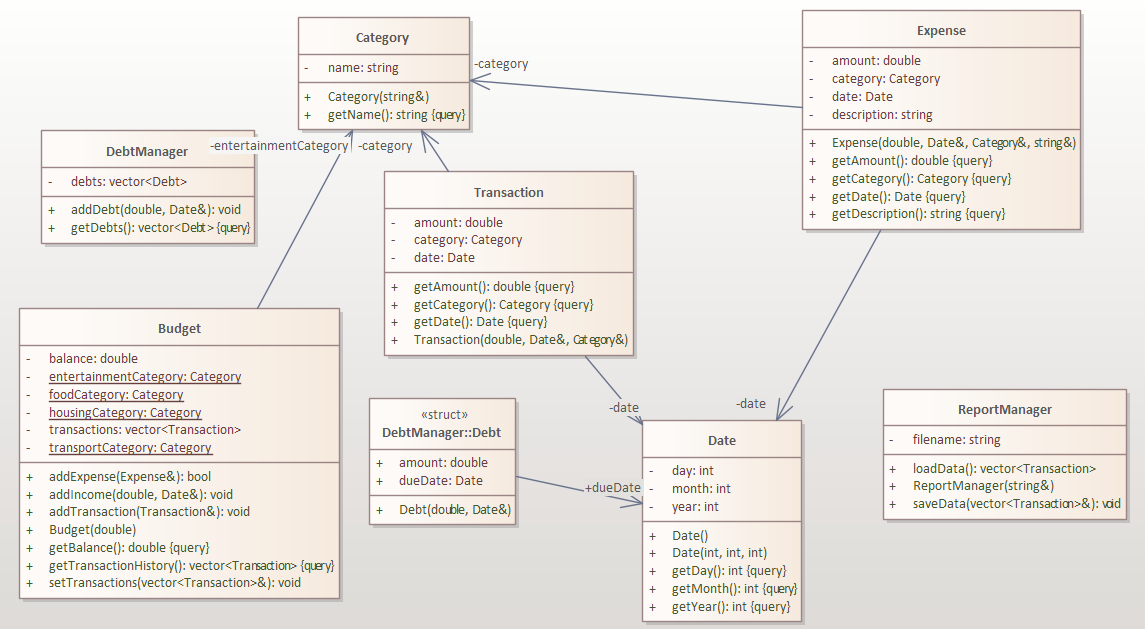


Рисунок 1. Диаграмма классов

# **Листинг программы**

#include <iostream>

#include <fstream>

#include <string>

#include <vector>

#include <ctime>

#include <limits>

using namespace std;

class Date {

private:

int year;

int month;

int day;

public:

Date() : year(0), month(0), day(0) {}

Date(int year, int month, int day) : year(year), month(month), day(day) {}

int getYear() const { return year; }

int getMonth() const { return month; }

int getDay() const { return day; }

};

class Category {

private:

string name;

public:

Category(const string& name) : name(name) {}

string getName() const { return name; }

};

class Expense {

private:

double amount;

Date date;

Category category;

string description;

public:

Expense(double amount, const Date& date, const Category& category, const string& description)

: amount(amount), date(date), category(category), description(description) {}

double getAmount() const { return amount; }

Date getDate() const { return date; }

Category getCategory() const { return category; }

string getDescription() const { return description; }

};

class Transaction {

private:

double amount;

Date date;

Category category;

public:

Transaction(double amount, const Date& date, const Category& category)

: amount(amount), date(date), category(category) {}

double getAmount() const { return amount; }

Date getDate() const { return date; }

Category getCategory() const { return category; }

};

class Budget {

private:

double balance;

vector<Transaction> transactions;

static Category foodCategory;

static Category transportCategory;

static Category housingCategory;

static Category entertainmentCategory;

public:

Budget(double initialBalance) : balance(initialBalance) {}

void addTransaction(const Transaction& transaction) {

transactions.push\_back(transaction);

balance += transaction.getAmount();

}

double getBalance() const { return balance; }

void setTransactions(const vector<Transaction>& transactions) {

this->transactions = transactions;

}

bool addExpense(const Expense& expense) {

if (balance < expense.getAmount()) {

cout << "Ошибка: Недостаточно средств на счете!" << endl;

return false;

}

Transaction expenseTransaction(-expense.getAmount(), expense.getDate(), expense.getCategory());

addTransaction(expenseTransaction);

return true;

}

void addIncome(double amount, const Date& date) {

Transaction incomeTransaction(amount, date, Category("Доход"));

addTransaction(incomeTransaction);

}

vector<Transaction> getTransactionHistory() const {

return transactions;

}

};

Category Budget::foodCategory("Продукты");

Category Budget::transportCategory("Транспорт");

Category Budget::housingCategory("Жилье");

Category Budget::entertainmentCategory("Развлечения");

void displayMenu() {

cout << "===== Система Финансового Управления =====" << endl;

cout << "1. Добавить Расход" << endl;

cout << "2. Добавить Доход" << endl;

cout << "3. Просмотреть Текущий Баланс" << endl;

cout << "4. Просмотреть Историю Транзакций" << endl;

cout << "5. Добавить Долг" << endl;

cout << "6. Просмотреть Информацию о Долгах" << endl;

cout << "7. Сохранить отчет в файл" << endl;

cout << "8. Выйти из программы" << endl;

cout << "===========================================" << endl;

}

void displayTransactionHistory(const vector<Transaction>& history) {

cout << "===== История Транзакций =====" << endl;

for (const auto& transaction : history) {

cout << "Дата: " << transaction.getDate().getYear() << "-" << transaction.getDate().getMonth() << "-" << transaction.getDate().getDay();

cout << ", Сумма: " << transaction.getAmount();

cout << ", Категория: " << transaction.getCategory().getName() << endl;

}

cout << "=============================" << endl;

}

Date getCurrentDate() {

time\_t now = time(0);

tm\* currentTime = localtime(&now);

int year = currentTime->tm\_year + 1900;

int month = currentTime->tm\_mon + 1;

int day = currentTime->tm\_mday;

return Date(year, month, day);

}

Date inputDueDate() {

Date currentDate = getCurrentDate();

Date dueDate;

do {

int year, month, day;

cout << "Введите дату погашения долга (год, месяц, день): ";

cin >> year >> month >> day;

dueDate = Date(year, month, day);

if (dueDate.getYear() < currentDate.getYear() ||

(dueDate.getYear() == currentDate.getYear() && dueDate.getMonth() < currentDate.getMonth()) ||

(dueDate.getYear() == currentDate.getYear() && dueDate.getMonth() == currentDate.getMonth() && dueDate.getDay() < currentDate.getDay())) {

cout << "Ошибка: Вы ввели дату, которая уже прошла. Пожалуйста, введите корректную дату." << endl;

}

} while (dueDate.getYear() < currentDate.getYear() ||

(dueDate.getYear() == currentDate.getYear() && dueDate.getMonth() < currentDate.getMonth()) ||

(dueDate.getYear() == currentDate.getYear() && dueDate.getMonth() == currentDate.getMonth() && dueDate.getDay() < currentDate.getDay()));

return dueDate;

}

template<typename T>

T inputNumber(const string& prompt) {

T value;

while (true) {

cout << prompt;

if (cin >> value) {

break;

} else {

cout << "Ошибка: Некорректный ввод. Пожалуйста, введите число." << endl;

cin.clear();

cin.ignore(numeric\_limits<streamsize>::max(), '\n');

}

}

return value;

}

class DebtManager {

private:

struct Debt {

double amount;

Date dueDate;

Debt(double amount, const Date& dueDate) : amount(amount), dueDate(dueDate) {}

};

vector<Debt> debts;

public:

void addDebt(double amount, const Date& dueDate) {

Debt newDebt(amount, dueDate);

debts.push\_back(newDebt);

}

vector<Debt> getDebts() const {

return debts;

}

};

class ReportManager {

private:

string filename;

public:

ReportManager(const string& filename) : filename(filename) {}

void saveData(const vector<Transaction>& transactions) {

ofstream file(filename);

if (file.is\_open()) {

for (const auto& transaction : transactions) {

file << transaction.getAmount() << " "

<< transaction.getDate().getYear() << " "

<< transaction.getDate().getMonth() << " "

<< transaction.getDate().getDay() << " "

<< transaction.getCategory().getName() << endl;

}

file.close();

cout << "Данные успешно сохранены в файле " << filename << endl;

} else {

cout << "Ошибка при открытии файла для сохранения данных." << endl;

}

}

vector<Transaction> loadData() {

vector<Transaction> transactions;

ifstream file(filename);

if (file.is\_open()) {

double amount;

int year, month, day;

string categoryName;

while (file >> amount >> year >> month >> day >> categoryName) {

Date date(year, month, day);

Category category(categoryName);

Transaction transaction(amount, date, category);

transactions.push\_back(transaction);

}

file.close();

cout << "Данные успешно загружены из файла " << filename << endl;

} else {

cout << "Ошибка при открытии файла для загрузки данных." << endl;

}

return transactions;

}

};

int main() {

Date currentDate(2024, 5, 24);

Budget budget(1000.0);

DebtManager debtManager;

ReportManager reportManager("/Users/Abdulmunimikramov/Desktop/OOP/transactions.txt");

vector<Transaction> loadedTransactions = reportManager.loadData();

for (const auto& transaction : loadedTransactions) {

budget.addTransaction(transaction);

}

int choice;

do {

displayMenu();

cout << "Введите ваш выбор: ";

choice = inputNumber<int>("");

switch (choice) {

case 1: {

double amount = inputNumber<double>("Введите сумму расхода: ");

cout << "Выберите категорию расхода:" << endl;

cout << "1. Продукты" << endl;

cout << "2. Транспорт" << endl;

cout << "3. Жилье" << endl;

cout << "4. Развлечения" << endl;

int categoryChoice = inputNumber<int>("Введите номер категории: ");

Category selectedCategory("None");

switch (categoryChoice) {

case 1:

selectedCategory = Category("Продукты");

break;

case 2:

selectedCategory = Category("Транспорт");

break;

case 3:

selectedCategory = Category("Жилье");

break;

case 4:

selectedCategory = Category("Развлечения");

break;

default:

cout << "Некорректный выбор категории. Расход не добавлен." << endl;

continue;

}

if (!budget.addExpense(Expense(amount, currentDate, selectedCategory, "Расход")))

cout << "Операция отменена." << endl;

break;

}

case 2: {

double amount = inputNumber<double>("Введите сумму дохода: ");

budget.addIncome(amount, currentDate);

break;

}

case 3:

cout << "Текущий баланс: " << budget.getBalance() << endl;

break;

case 4:

budget.setTransactions(reportManager.loadData());

displayTransactionHistory(budget.getTransactionHistory());

break;

case 5: {

double amount = inputNumber<double>("Введите сумму долга: ");

Date dueDate = inputDueDate();

debtManager.addDebt(amount, dueDate);

cout << "Долг успешно добавлен." << endl;

break;

}

case 6: {

auto debts = debtManager.getDebts();

cout << "===== Информация о Долгах =====" << endl;

for (const auto& debt : debts) {

cout << "Сумма долга: " << debt.amount << ", Дата погашения: "

<< debt.dueDate.getYear() << "-" << debt.dueDate.getMonth() << "-" << debt.dueDate.getDay() << endl;

}

cout << "=============================" << endl;

break;

}

case 7:

reportManager.saveData(budget.getTransactionHistory());

cout << "Данные сохранены. Завершение программы. До свидания!" << endl;

break;

default:

cout << "Неверный выбор. Пожалуйста, введите правильную опцию." << endl;

}

} while (choice != 8);

return 0;

}

# **Тесты**

При запуске нас приветствует меню, где нам предлагается выбрать действие:

* Добавить Расход: Позволяет пользователю добавить информацию о расходе, включая сумму, категорию и описание.
* Добавить Доход: Позволяет пользователю добавить информацию о доходе, включая сумму.
* Просмотреть Текущий Баланс: Отображает текущий баланс.
* Просмотреть Историю Транзакций: Показывает пользователю историю всех транзакций.
* Добавить Долг: Позволяет пользователю добавить информацию о долге, включая сумму и дату погашения.
* Просмотреть Информацию о Долгах: Отображает информацию о всех текущих долгах.
* Сохранить отчет в файл: Сохраняет данные о транзакциях в файл "transactions.txt".
* Выйти из программы: Завершает выполнение программы.

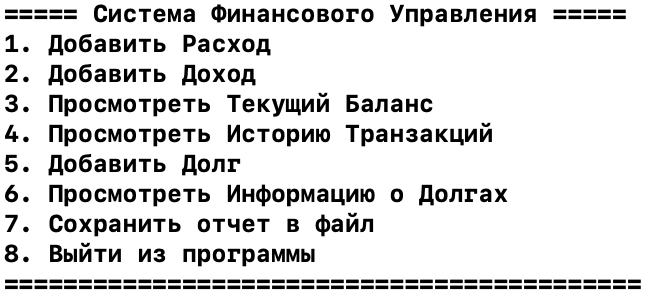


Рисунок 2. Выбор действий

Все данные сохраняются в одном файле transactions.txt

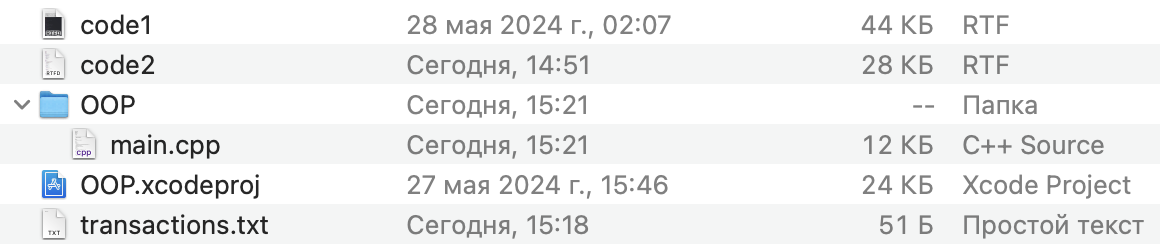


Рисунок 3. Хранимые данные

В пункте "Добавить Расход" пользователь может взаимодействовать с программой, чтобы добавить информацию о своем расходе. При выборе этого пункта пользователю предлагается ввести сумму расхода и выбрать категорию из предложенных опций:

Продукты

Транспорт

Жилье

Развлечения

После ввода суммы расхода и выбора категории, программа добавляет этот расход в текущий баланс и отображает сообщение об успешном добавлении, если на счете достаточно средств. Если же средств недостаточно, программа выведет сообщение об ошибке и расход не будет добавлен.Таким образом, путем взаимодействия с этим пунктом

меню пользователь может записать свои расходы, чтобы вести учет своих финансов.

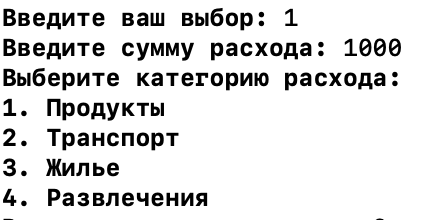




Рисунок 5. Меню Расхода

При выборе опции "Добавить Доход" пользователь может взаимодействовать с программой, чтобы добавить информацию о своем доходе. После выбора этой опции программа запрашивает у пользователя ввести сумму дохода. После ввода суммы дохода программа автоматически записывает этот доход в текущий баланс.

Программа не запрашивает у пользователя выбор категории для дохода, поскольку все доходы предполагаются как общие доходы. Поэтому после ввода суммы дохода, она сразу добавляется к общему балансу.

Этот пункт меню предоставляет пользователю простой и быстрый способ отслеживать и учитывать свои доходы.





Рисунок 6. Меню Дохода

Когда пользователь выбирает опцию "Просмотреть Текущий Баланс", программа отображает текущий баланс. Этот баланс представляет собой сумму денег на счете после всех совершенных транзакций, включая доходы и расходы.

После выбора этой опции программа выводит текущий баланс на экран, предоставляя пользователю информацию о том, сколько денег у него на счете в данный момент. Это позволяет пользователям быстро оценить свое финансовое положение и принять решение о дальнейших действиях.



Рисунок 7. Меню просмотреть текущий баланс

При выборе опции "Просмотреть Историю Транзакций" программа отображает пользователю историю всех транзакций, которые были совершены. Это включает в себя как доходы, так и расходы, которые были добавлены в систему.

Программа выводит на экран дату каждой транзакции, ее сумму и категорию. Это позволяет пользователям получить обзор своих финансовых операций за определенный период времени и отследить, куда уходят и откуда приходят их деньги.

После вывода истории транзакций пользователю снова предоставляется меню, где он может выбрать следующее действие, включая добавление новых транзакций, просмотр текущего баланса и т. д.

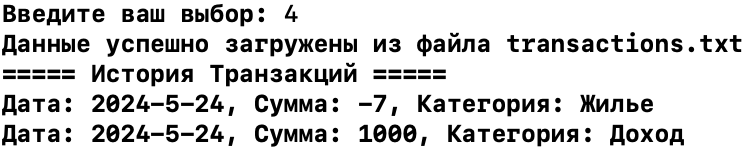


Рисунок 8. Меню Просмотреть Историю Транзакций

Пункт меню "Добавить Долг" предоставляет возможность пользователю добавить информацию о долге. После выбора этой опции программа запрашивает у пользователя ввести сумму долга и дату погашения.

После ввода данных программа автоматически добавляет этот долг в список управления долгами. Пользователь получит сообщение об успешном добавлении долга. Данный функционал может быть полезен для отслеживания сроков погашения долгов и контроля финансового состояния.

После добавления долга пользователю снова предоставляется меню для выбора следующего действия.

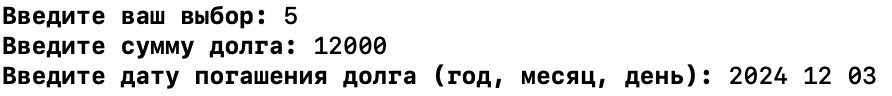


Рисунок 9. Меню Добавить Долг

Пункт меню "Просмотреть Информацию о Долгах" позволяет пользователю просмотреть информацию о текущих долгах. После выбора этой опции программа отображает список всех текущих долгов, включая сумму долга и дату погашения.

Пользователь получит подробную информацию о каждом долге, что поможет ему следить за своими обязательствами и контролировать свои финансы. После отображения информации о долгах пользователю снова предоставляется меню для выбора следующего действия.

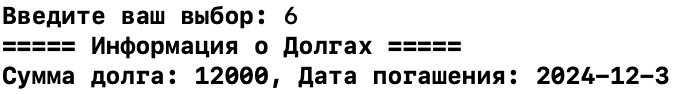


Рисунок 10. Меню Просмотреть Информацию о Долгах

При выборе опции "Сохранить отчет в файл" программа сохраняет данные о всех транзакциях в файл с названием "transactions.txt". Это позволяет пользователю сохранить историю своих финансовых операций для последующего просмотра или анализа.

Программа выводит сообщение о успешном сохранении данных в файл, если операция прошла успешно. В случае возникновения ошибки при сохранении, пользователь получит соответствующее сообщение об ошибке.

Этот функционал полезен для того, чтобы иметь возможность в будущем вернуться к истории своих транзакций или передать данные о финансах другим людям или программам для анализа.

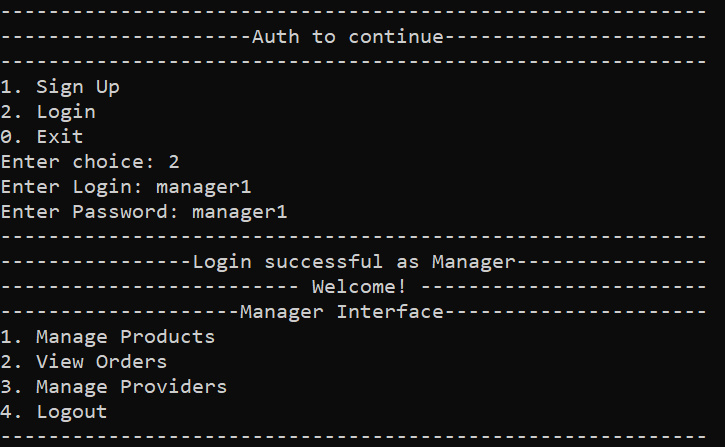


Рисунок 11. Меню Сохранить отчет в файл

При выборе опции "Выйти из программы" программа завершает свою работу, выходя из цикла обработки пользовательского ввода в главной функции main(). После этого программа завершает выполнение, и управление возвращается операционной системе.

Это действие позволяет пользователю завершить работу с программой после выполнения всех необходимых операций, предоставляя удобный способ выхода из приложения.

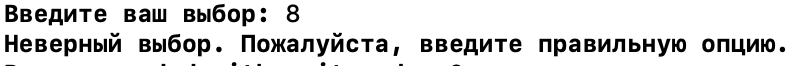


Рисунок 12. Меню завершения программы

# 

# **Заключение**

Программа для управления финансами предоставляет пользователю удобные инструменты для отслеживания и управления своими финансами. Она включает в себя функции добавления расходов и доходов, учета долгов, просмотра истории транзакций, а также сохранения данных в файл для последующего анализа.

Путем взаимодействия с меню пользователь может легко выполнять различные финансовые операции, получать информацию о текущем балансе, а также отслеживать свои долги. Программа предоставляет простой и понятный интерфейс, который делает управление финансами более эффективным и удобным.

Завершение работы программы происходит по желанию пользователя, который может выбрать опцию выхода из приложения из главного меню. Это позволяет пользователям легко завершать работу с программой, когда все необходимые операции выполнены.

В целом, программа представляет собой полезный инструмент для тех, кто стремится эффективно управлять своими финансами и поддерживать их в порядке.

https://github.com/AbdulMunimIkramov/OOP\_KURS