**МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ**

**КЫРГЫЗСКОЙ РЕСПУБЛИКИ**

**КЫРГЫЗСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ им. И.Раззакова**

**ФАКУЛЬТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ**

Кафедра: **Программное обеспечение компьютерных систем**

Курс «ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ С++»

**СРС**

**Тема: «Сбербанк»**

Выполнил: студент группы ПИ-3-21

Табылдиева Шоола

Проверил: Мусабаев Э.Б.

**Бишкек 2024**

Оглавление

[Цели работы 2](#_Toc164764656)

[C++ 2](#_Toc164764657)

[Задачи самостоятельной работы 2](#_Toc164764658)

[Решение 2](#_Toc164764659)

[Вывод 6](#_Toc164764660)

# Цели работы

Целью объектно-ориентированного программирования (ООП) является создание программных систем, которые легко понимать, модифицировать и поддерживать. В контексте решения задач ООП целью является разработка эффективного и структурированного программного решения, которое отражает структуру и взаимодействие объектов в реальном мире, упрощает разработку, обеспечивает повторное использование кода и улучшает его читаемость. Основные принципы ООП, такие как инкапсуляция, наследование и полиморфизм, используются для создания моделей, которые отображают реальные объекты и их взаимодействие, что делает программу более понятной и легкой для сопровождения.

# C++

**C++** — это компилируемый, статически типизированный язык программирования, который является расширением языка C. Он поддерживает различные парадигмы программирования, включая процедурное, объектно-ориентированное и обобщенное программирование, что делает его мощным инструментом для разработки разнообразных приложений, от небольших скриптовых программ до крупного системного уровня.

C++ для решения задач ООП:

1. Поддержка объектно-ориентированного программирования: C++ обладает мощными средствами для создания классов и объектов, что позволяет разработчикам эффективно моделировать реальные объекты и их взаимодействие.
2. Инкапсуляция: C++ поддерживает инкапсуляцию, позволяя скрывать детали реализации объектов от пользователей классов. Это способствует разделению ответственности и упрощает разработку сложных систем.
3. Наследование: Одним из основных принципов ООП является наследование, которое позволяет создавать новые классы на основе существующих. C++ обеспечивает механизмы наследования, позволяющие создавать иерархии классов для повторного использования кода и создания иерархических структур.
4. Полиморфизм: C++ поддерживает полиморфизм, что означает, что одинаково названные методы могут выполнять различные действия в зависимости от типа объекта, к которому они применяются. Это способствует созданию гибких и расширяемых программ.
5. Эффективность: C++ является компилируемым языком, что означает, что код компилируется в машинный код, что обеспечивает высокую производительность и эффективность исполнения программ.

# Задачи самостоятельной работы

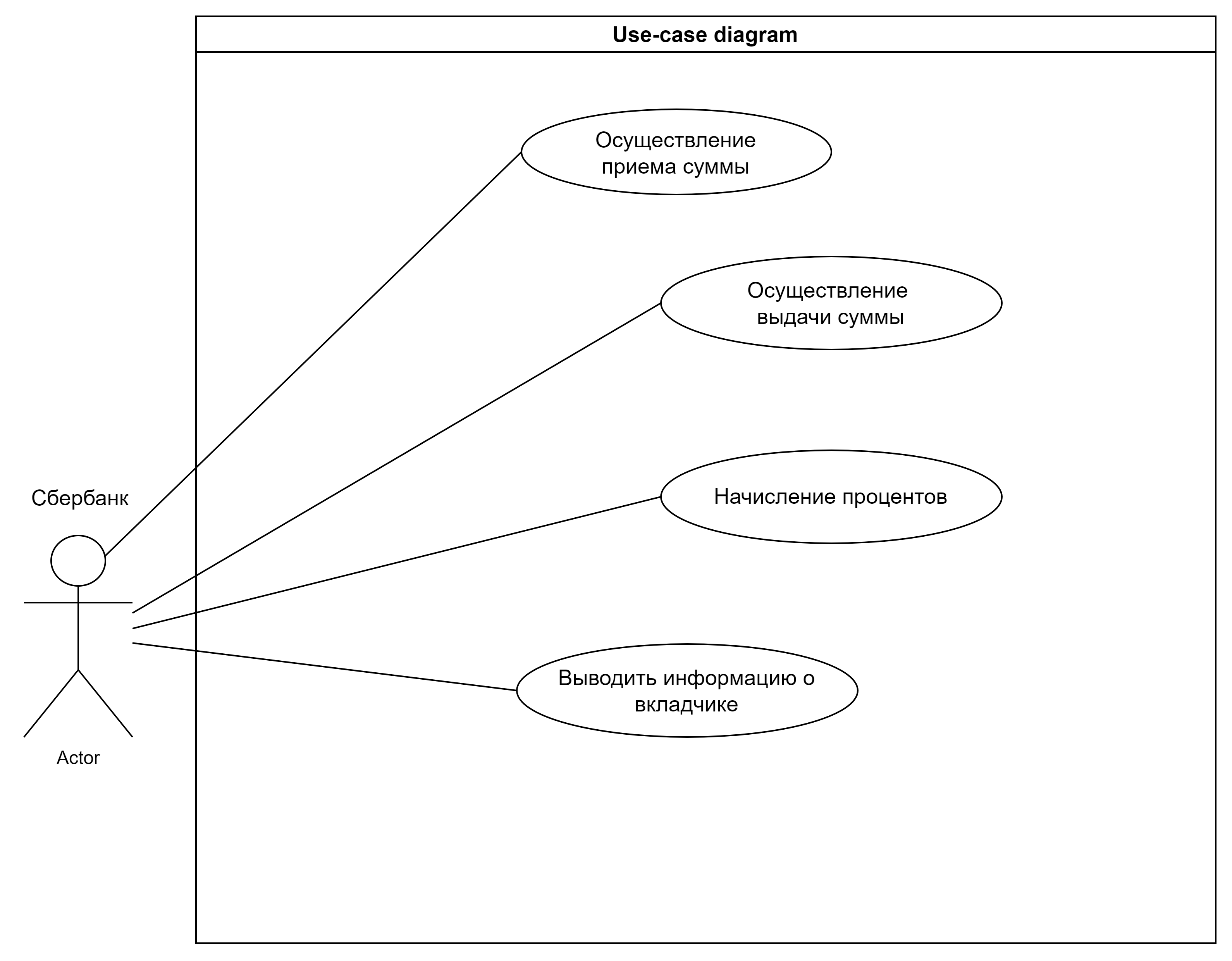
Реализовать программу с использованием принципов ООП:

Сбербанк. Сведения о вкладчиках банка: номер лицевого счета, категория вклада, паспортные данные, текущая сумма вклада, дата последней операции. Операции приема и выдачи любой суммы, автоматическое начисление процентов.

# Решение

Наша программа является консольным приложением. Написан был на языке C++.

**Use-Case диаграмма:**



Листинг кода:

#include <iostream>

#include <vector>

#include <string>

#include <ctime>

#include <Windows.h>

#include <thread>

using namespace std;

enum class DepositCategory {

ORDINARY,

VIP,

BUSINESS

};

class Depositor {

private:

string accountNumber;

DepositCategory depositCategory;

string passportData;

double currentAmount;

time\_t lastOperationDate;

public:

Depositor(string accNum, DepositCategory depCat, string passData, double currAmt) :

accountNumber(accNum), depositCategory(depCat), passportData(passData), currentAmount(currAmt) {

time(&lastOperationDate); // Устанавливаем текущую дату и время как дату последней операции

}

// Метод для приема суммы

void deposit(double amount) {

currentAmount += amount;

time(&lastOperationDate);

cout << "Сумма " << amount << " успешно зачислена на счет." << endl;

}

// Метод для выдачи суммы

void withdraw(double amount) {

if (amount <= currentAmount) {

currentAmount -= amount;

time(&lastOperationDate);

char buffer[80];

ctime\_s(buffer, sizeof(buffer), &lastOperationDate);

cout << "Дата последней операции: " << buffer;

cout << "Сумма " << amount << " успешно выдана со счета. Время: " << buffer <<endl;

}

else {

cout << "Ошибка: недостаточно средств на счете." << endl;

}

}

//начисления процентов

void applyInterest(double rate) {

double interest = currentAmount \* rate / 100.0;

currentAmount += interest;

time(&lastOperationDate);

char buffer[80];

ctime\_s(buffer, sizeof(buffer), &lastOperationDate);

cout << "Проценты в размере " << interest << " успешно начислены. Время: " << buffer << endl;

}

// Метод для вывода информации о вкладчике

void printInfo() {

cout << "Номер счета: " << accountNumber << endl;

cout << "Категория вклада: ";

switch (depositCategory) {

case DepositCategory::VIP:

cout << "VIP" << endl;

break;

case DepositCategory::BUSINESS:

cout << "Бизнес" << endl;

break;

default:

cout << "Обычный" << endl;

break;

}

cout << "Паспортные данные: " << passportData << endl;

cout << "Текущая сумма вклада: " << currentAmount << endl;

char buffer[80];

ctime\_s(buffer, sizeof(buffer), &lastOperationDate);

cout << "Дата последней операции: " << buffer;

}

};

int main() {

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

// Создаем объекты вкладчиков

Depositor depositor1("123789", DepositCategory::ORDINARY, "1234567890", 5000.0);

Depositor depositor2("984321", DepositCategory::VIP, "0987654321", 10000.0);

cout << "Информация о вкладчике 1:" << endl;

depositor1.printInfo();

cout << endl;

cout << "Информация о вкладчике 2:" << endl;

depositor2.printInfo();

cout << endl << endl;

depositor1.deposit(1000.0);

std::this\_thread::sleep\_for(std::chrono::seconds(10));

depositor2.withdraw(2000.0);

std::this\_thread::sleep\_for(std::chrono::seconds(12));

depositor1.applyInterest(5.0);

cout << "Информация о вкладчике 1:" << endl;

depositor1.printInfo();

cout << endl;

cout << "Информация о вкладчике 2:" << endl;

depositor2.printInfo();

return 0;

}

Результат выполнения программы (рис.1):

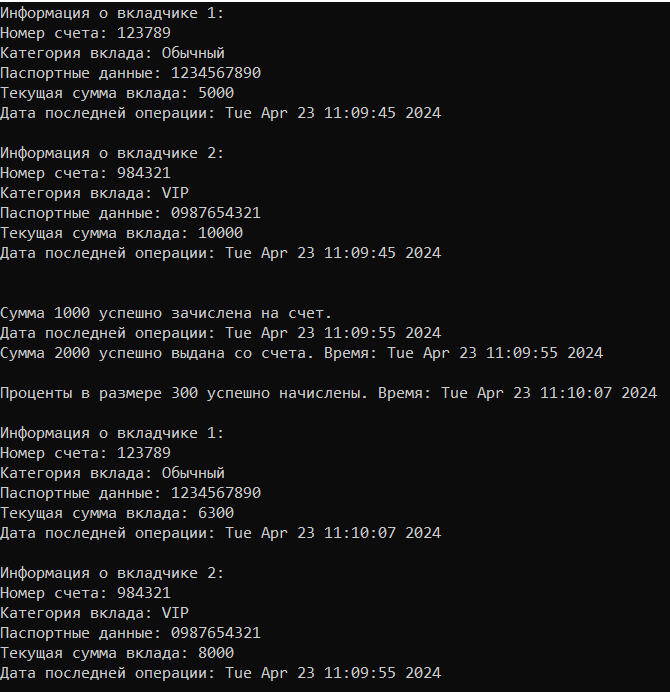


Рисунок 1 Результат выполнения программы

# Вывод

В результате создания данной программы мы приобрели опыт в применении принципов ООП для решения конкретных задач, а также углубились в работу с языком C++, который обеспечивает высокую производительность и мощные инструменты для разработки программного обеспечения.