Курсовая работа

Дисциплина: Основы Программирования и Алгоритмизации

Тема: Упрощённая симуляция системы блокчейн-транзакций

Github: <https://github.com/chelik010/algoritm-5130203-40001/tree/master/kursovaya_final_working>

Преподаватель: Хольгер Эспинола Ривера

Студенты: Сузоков Александр, Полярус Лидия

Группа: 5130203/40001

Дата: 2025

# Постановка задачи

Цель курсовой работы — реализовать упрощённую систему имитации транзакций в блокчейн-сети. В рамках системы необходимо реализовать хранение клиентов, кошельков и транзакций, используя структуры данных: двоичное дерево поиска и двусвязный список. Клиенты должны иметь тип (стандартный, платиновый, золотой), влиющий на комиссии и лимиты транзакций. Все данные должны сохраняться в файлы, поддерживать добавление, просмотр и обработку транзакций, а также взаимодействие между кошельками.

# Исходные данные задачи

Программа работает с файлами:

- Clients.txt — содержит информацию о клиентах и их кошельках.

- Blockchain\_transactions.txt — содержит информацию о совершенных транзакциях.

Пользователь взаимодействует через меню, где может добавлять клиентов, создавать транзакции, выводить информацию.

# Определение терминов

Клиент — пользователь системы с уникальным ID, именем и кошельками.

Кошелёк — объект, содержащий баланс и принадлежащий клиенту.

Транзакция — операция перевода средств между двумя кошельками.

BlockChain — основной управляющий класс, хранящий всех клиентов и транзакции.

ClientBST — бинарное дерево клиентов, упорядоченное по балансу.

TransactionList — двусвязный список всех транзакций.

Комиссия — сумма, удерживаемая системой при транзакции (зависит от типа клиента).

# Описание реализованных методов

## Класс Client (абстрактный)

Метод: getTotalBalance()

- Параметры: отсутствуют

- Возвращаемое значение: double

- Назначение: возвращает общий баланс по всем кошелькам клиента

- Реализация:

double Client::getTotalBalance() const {  
 double total = 0.0;  
 for (Wallet\* wallet : wallets.getAllEntities()) {  
 total += wallet->getBalance();  
 }  
 return total;  
}

Метод: calculateCommission(double amount) — виртуальный

- Назначение: вычисляет комиссию в зависимости от типа клиента

## Класс Wallet

Метод: deposit(double amount)

- Параметры: amount — сумма для пополнения

- Возвращает: void

- Назначение: добавляет средства на баланс кошелька

void Wallet::deposit(double amount) {  
 balance += amount;  
}

Метод: withdraw(double amount)

- Назначение: снимает средства с баланса, если достаточно средств

bool Wallet::withdraw(double amount) {  
 if (balance >= amount) {  
 balance -= amount;  
 return true;  
 }  
 return false;  
}

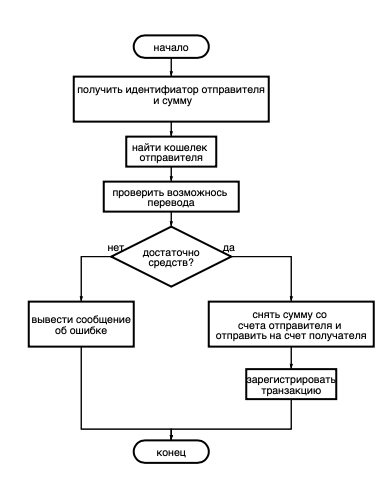
## Класс Blockchain

Метод: processTransaction(Transaction\* tx)

- Назначение: осуществляет перевод между кошельками с учётом комиссии

bool Blockchain::processTransaction(Transaction\* tx) {  
 Wallet\* sender = findWallet(tx->getSenderWalletId());  
 Wallet\* receiver = findWallet(tx->getReceiveWalletId());  
 double totalAmount = tx->getAmount() + tx->getCommission();  
  
 if (sender && receiver && sender->withdraw(totalAmount)) {  
 receiver->deposit(tx->getAmount());  
 transactions.addTransaction(tx);  
 return true;  
 }  
 return false;  
}

# Блок-схема перевода средств

Блок-схема иллюстрирует последовательность действий при выполнении перевода между кошельками клиента.

# Описание реализованных методов

## Класс Wallet

Класс представляет кошелёк клиента, используется для хранения баланса и идентификатора владельца.

Методы:

1. deposit(double amount) — добавляет средства на кошелёк.

void Wallet::deposit(double amount) {  
 balance += amount;  
}

2. withdraw(double amount) — снимает средства с кошелька, если достаточно баланса.

bool Wallet::withdraw(double amount) {  
 if (balance >= amount) {  
 balance -= amount;  
 return true;  
 }  
 return false;  
}

3. getBalance() — возвращает текущий баланс.

double Wallet::getBalance() const {  
 return balance;  
}

## Класс Transaction

Представляет транзакцию между двумя кошельками. Содержит ID, сумму, комиссию и тип.

Методы:

1. getDetails() — возвращает строку с информацией о транзакции.

std::string Transaction::getDetails() const {  
 return "Transaction " + id + ": " + std::to\_string(amount) + " from " + senderWalletId + " to " + receiveWalletId;  
}

## Метод processTransaction в Blockchain

Обрабатывает перевод средств с одного кошелька на другой с учётом комиссии отправителя.

bool Blockchain::processTransaction(Transaction\* tx) {  
 Client\* senderClient = clients.find(tx->getSenderWalletId());  
 Wallet\* senderWallet = senderClient->getWallet(tx->getSenderWalletId());  
 if (!senderWallet || senderWallet->getBalance() < tx->getAmount() + tx->getCommission()) return false;  
 senderWallet->withdraw(tx->getAmount() + tx->getCommission());  
  
 Client\* receiverClient = clients.find(tx->getReceiveWalletId());  
 Wallet\* receiverWallet = receiverClient->getWallet(tx->getReceiveWalletId());  
 if (!receiverWallet) return false;  
 receiverWallet->deposit(tx->getAmount());  
  
 transactions.addTransaction(tx);  
 return true;  
}

# Тестирование

Программа была протестирована с помощью сценариев:

- Добавление клиентов разных типов

- Создание кошельков с начальными балансами

- Проведение транзакций с учётом комиссии

- Проверка отказа при недостаточном балансе

- Проверка корректной записи в файлы

Результаты тестов подтверждают корректность расчёта комиссий, обработку лимитов и правильную запись транзакций в файл.

# Заключение

В рамках курсовой работы было:

- Изучено: принципы ООП, работа с файлами, структуры данных.

- Освоено: использование бинарного дерева и списка, работа с динамической памятью.

- Реализовано: система клиентов и транзакций с наследованием, полиморфизмом и обработкой ошибок.

Система успешно выполняет поставленные задачи и готова к дальнейшему расширению (например, поддержка дополнительных типов транзакций или более сложных комиссий).