ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«САНКТ-ПЕТЕРБУРГСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ПЕТРА ВЕЛИКОГО»
Институт компьютерных наук и кибербезопасности

Отчет по курсовой работе

Моторин Гордей Павлович

(Ф.И.О. обучающегося)

1 курс, 5130203/40002

(номер курса обучения и учебной группы)

02.03.03 Математическое обеспечение и администрирование информационных систем

(направление подготовки (код и наименование))

Основы программирования и алгоритмизации

(наименование дисциплины)

Оценка:

Преподаватель: Эспинола Ривера Хольгер Элиас

Обучающийся: Моторин Г. П.

Дата: 07.06.25

Постановка задачи:

Требовалось разработать упрощённую систему имитации блокчейна, которая:

- Управляет клиентами (разных типов) и их кошельками.
- Обрабатывает транзакции между кошельками с учётом комиссий и лимитов.
- Хранит данные в структурах: двоичное дерево поиска (для клиентов) и двусвязный список (для транзакций).

Исходные данные задачи:

- Клиенты: GoldClient, PlatinumClient, StandardClient (разные комиссии и лимиты).
- Кошельки: принадлежат клиентам, хранят баланс.
- Транзакции: переводы между кошельками с комиссией.

Термины предметной области:

- **Блокчейн** — система учёта транзакций.
- **Клиент** — пользователь системы (владелец кошельков).
- **Кошелёк** — контейнер для средств с уникальным ID.
- **Транзакция** — операция перевода средств между кошельками.
- **Комиссия** — плата за перевод (зависит от типа клиента).

Описание реализованных методов:

1. Класс Entity (Базовый абстрактный класс)

1. Entity(const string& id)
    - Параметры: id — уникальный идентификатор сущности.
    - Возвращаемое значение: Нет.
    - Назначение: Конструктор, инициализирует ID сущности.
    - Реализация: Принимает строку id и сохраняет её в поле класса.

Код:
```
Entity(const string& id) {
    this->id = id;
}
```

2. virtual ~Entity()
    - Параметры: Нет.
    - Возвращаемое значение: Нет.
    - Назначение: Виртуальный деструктор для корректного удаления производных классов.
    - Реализация: Пустой деструктор, но объявлен виртуальным для безопасного удаления объектов через указатель на базовый класс.

Код:
```
virtual ~Entity() {}
```

3. virtual string getId()
    - Параметры: Нет.
    - Возвращаемое значение: string — ID сущности.
    - Назначение: Возвращает уникальный идентификатор.
    - Реализация: Возвращает сохранённое значение поля id.

Код:
```
virtual string getId() {
    return this->id;
}
```

2. Класс EntityVector (Управление коллекцией сущностей)

1. void addEntity(Entity* entity)
    - Параметры: entity — указатель на добавляемую сущность.
    - Возвращаемое значение: Нет.
    - Назначение: Добавляет сущность в вектор.
    - Реализация: Использует vector::push_back() для добавления указателя в конец вектора.

Код: void addEntity(Entity* entity) {
  entities.push_back(entity);
}

2. bool removeEntity(const string& id)
    - Параметры: id — ID удаляемой сущности.
    - Возвращаемое значение: bool — true, если удаление успешно.
    - Назначение: Удаляет сущность по ID.
    - Реализация:
        - Ищет сущность в векторе по id с помощью std::find_if.
        - Если находит, удаляет её из вектора и возвращает true.
        - Если не находит, возвращает false.

Код: bool removeEntity(const string& id) {
  auto it = find_if(entities.begin(), entities.end(),[&id](Entity* e) { return e->getId() == id;
});
  if (it != entities.end()) {
    entities.erase(it);
    return true;
  }
  return false;
}

3. Entity* getEntity(const string& id)
    - Параметры: id — ID искомой сущности.
    - Возвращаемое значение: Entity* — указатель на сущность или nullptr.
    - Назначение: Поиск сущности по ID.
    - Реализация:
        - Перебирает вектор и сравнивает id каждой сущности с искомым.
        - Возвращает указатель на найденную сущность или nullptr.

Код: Entity* getEntity(const string& id) {
  auto it = find_if(entities.begin(), entities.end(),[&id](Entity* e) { return e->getId() == id;
});
  return (it != entities.end()) ? *it : nullptr;
}

4. vector<Entity*>& getAllEntities()
    - Параметры: Нет.
    - Возвращаемое значение: vector<Entity*>& — ссылка на вектор всех сущностей.
    - Назначение: Возвращает все сущности.
    - Реализация: Просто возвращает ссылку на внутренний вектор entities.

Код: vector<Entity*>& getAllEntities() {
  return entities;
}

5. ~EntityVector()
    - Параметры: Нет.
    - Возвращаемое значение: Нет.
    - Назначение: Деструктор, освобождает память всех сущностей.
    - Реализация:

- В цикле проходит по вектору и удаляет каждый объект через delete.
- Очищает вектор.

Код: ~EntityVector() {
for (auto e : entities) {
delete e;
}
}

3. Класс Wallet (Кошелёк)

1. Wallet(const string& id, const string& ownerId, double balance)
   - Параметры:
     - id — ID кошелька.
     - ownerId — ID владельца.
     - balance — начальный баланс.
   - Возвращаемое значение: Нет.
   - Назначение: Конструктор кошелька.
   - Реализация:
     - Сохраняет переданные id, ownerId и balance в поля класса.

Код: Wallet(const string& id, const string& ownerId, double balance)
: Entity(id) {
this->ownerId = ownerId;
this->balance = balance;
}

2. void deposit(double amount)
   - Параметры: amount — сумма для зачисления.
   - Возвращаемое значение: Нет.
   - Назначение: Пополняет баланс.
   - Реализация:
     - Увеличивает поле balance на amount.

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

3. bool withdraw(double amount)
   - Параметры: amount — сумма для снятия.
   - Возвращаемое значение: bool — true, если снятие успешно.
   - Назначение: Снимает средства (если достаточно баланса).
   - Реализация:
     - Проверяет, что balance >= amount.
     - Если да — уменьшает balance и возвращает true.
     - Если нет — возвращает false.

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

4. double getBalance() const
   - Параметры: Нет.
   - Возвращаемое значение: double — текущий баланс.
   - Назначение: Возвращает баланс.
   - Реализация: Просто возвращает значение поля balance.

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

5. string getOwnerId() const
    - Параметры: Нет.
    - Возвращаемое значение: string — ID владельца.
    - Назначение: Возвращает владельца кошелька.
    - Реализация: Возвращает сохранённое значение ownerId.

Код: string getOwnerId() const { return ownerId; }

6. string getId() const
    - Параметры: Нет.
    - Возвращаемое значение: string — ID кошелька.
    - Назначение: Возвращает ID кошелька.
    - Реализация: Возвращает сохранённое значение id.

Код: string getId() const { return id; }

4. Класс Transaction (Транзакция)

1. Transaction(...)
    - Параметры:
        - id — ID транзакции.
        - senderId — ID кошелька отправителя.
        - receiverId — ID кошелька получателя.
        - amount — сумма.
        - type — тип транзакции.
        - commission — комиссия.
    - Возвращаемое значение: Нет.
    - Назначение: Конструктор транзакции.
    - Реализация:
        - Сохраняет все переданные параметры в соответствующие поля класса.

Код: Transaction(const string& id, const string& senderId, const string& receiverId, double amount, TxType type, double commission)
: Entity(id) {
this->senderWalletId = senderId;
this->receiveWalletId = receiverId;
this->amount = amount;
this->type = type;
this->commission = commission;
}

2. string getSenderWalletId() const
    - Параметры: Нет.
    - Возвращаемое значение: string — ID отправителя.
    - Назначение: Возвращает отправителя.
    - Реализация: Возвращает сохранённое значение senderWalletId.

Код: string getSenderWalletId() const { return senderWalletId; }

3. string getReceiverWalletId() const
    - Параметры: Нет.
    - Возвращаемое значение: string — ID получателя.
    - Назначение: Возвращает получателя.
    - Реализация: Возвращает сохранённое значение receiveWalletId.

Код: string getReceiverWalletId() const { return receiveWalletId; }

4. double getAmount() const
    - Параметры: Нет.
    - Возвращаемое значение: double — сумма.
    - Назначение: Возвращает сумму перевода.
    - Реализация: Возвращает сохранённое значение amount.

Код: double getAmount() const { return amount; }

5. double getCommission() const
    - Параметры: Нет.
    - Возвращаемое значение: double — комиссия.
    - Назначение: Возвращает комиссию.
    - Реализация: Возвращает сохранённое значение commission.

Код: double getCommission() const { return commission; }

6. string getDetails() const
    - Параметры: Нет.
    - Возвращаемое значение: string — описание транзакции.
    - Назначение: Формирует строку с деталями транзакции.
    - Реализация:
        - Собирает строку из полей класса (id, senderId, receiverId, amount, commission).
        - Возвращает её в формате: "Transaction [id]: [amount] from [sender] to [receiver] (commission: [commission])".

```
Код: string getDetails() const {
   return "Transaction " + id + ": " + to_string(amount) + " from " +
      senderWalletId + " to " + receiveWalletId + " (commission: " +
      to_string(commission) + ")";
}
```

5. Класс Client (Абстрактный клиент)

1. Client(const string& id, const string& name)
    - Параметры:
        - id — ID клиента.
        - name — имя клиента.
    - Возвращаемое значение: Нет.
    - Назначение: Конструктор клиента.
    - Реализация:
        - Сохраняет id и name в поля класса.

```
Код: Client(const string& id, const string& name) : Entity(id) {
   this->name = name;
}
```

2. void addWallet(Wallet* wallet)
    - Параметры: wallet — указатель на кошелёк.
    - Возвращаемое значение: Нет.
    - Назначение: Добавляет кошелёк клиенту.
    - Реализация:
        - Передаёт указатель на кошелёк в EntityVector wallets через метод addEntity().

```
Код: void addWallet(Wallet* wallet) {
   wallets.addEntity(wallet);
}
```

3. double getTotalBalance()
    - Параметры: Нет.
    - Возвращаемое значение: double — общий баланс всех кошельков.
    - Назначение: Суммирует балансы кошельков.
    - Реализация:
        - Получает все кошельки через getAllEntities().
        - В цикле суммирует балансы (getBalance()).
        - Возвращает итоговую сумму.

Код: double getTotalBalance() {
double total = 0;
for (auto e : wallets.getAllEntities()) {
Wallet* w = dynamic_cast<Wallet*>(e);
if (w) total += w->getBalance();
}
return total;
}

4. virtual double calculateCommission(double amount) = 0
    - Параметры: amount — сумма.
    - Возвращаемое значение: double — размер комиссии.
    - Назначение: Расчёт комиссии (абстрактный метод).
    - Реализация: Определяется в дочерних классах (GoldClient, PlatinumClient, StandardClient).

Код: virtual double calculateCommission(double amount) const = 0;

5. virtual double getMaxTransactionLimit() = 0
    - Параметры: Нет.
    - Возвращаемое значение: double — максимальный лимит.
    - Назначение: Возвращает лимит (абстрактный метод).
    - Реализация: Определяется в дочерних классах.

Код: virtual double getMaxTransactionLimit() const = 0;

6. string getName() const
    - Параметры: Нет.
    - Возвращаемое значение: string — имя клиента.
    - Назначение: Возвращает имя.
    - Реализация: Возвращает сохранённое значение name.

Код: string getName() const { return name; }

7. EntityVector& getWallets()
    - Параметры: Нет.
    - Возвращаемое значение: EntityVector& — ссылка на кошельки.
    - Назначение: Возвращает кошельки клиента.
    - Реализация: Возвращает ссылку на внутренний EntityVector wallets.

Код: EntityVector& getWallets() { return wallets; }

6. Классы GoldClient, PlatinumClient, StandardClient

1. calculateCommission(double amount)
    - Параметры: amount — сумма транзакции.
    - Возвращаемое значение: double — комиссия.
    - Назначение: Рассчитывает комиссию.
    - Реализация:
        - GoldClient: amount * 0.03 (3%).
        - PlatinumClient: amount * 0.05 (5%).
        - StandardClient: amount * 0.10 (10%).

Код: double calculateCommission(double amount) const {
return amount * x; // x% комиссия
}

2. getMaxTransactionLimit()
    - Параметры: Нет.
    - Возвращаемое значение: double — максимальный лимит.
    - Назначение: Возвращает лимит транзакции.
    - Реализация:
        - GoldClient: 10000.0.
        - PlatinumClient: 5000.0.

- StandardClient: 1000.0.

Код: double getMaxTransactionLimit() const {
return x;
}

7. Класс Blockchain (Управление системой)

Методы:

1. void addClient(Client* client)
   - Параметры: client — указатель на клиента.
   - Возвращаемое значение: Нет.
   - Назначение: Добавляет клиента в систему.
   - Реализация:
     - Передаёт клиента в ClientBST clients через метод insert().

Код: void addClient(Client* client) {
clients.insert(client);
}

2. bool processTransaction(Transaction* tx)
   - Параметры: tx — указатель на транзакцию.
   - Возвращаемое значение: bool — успех обработки.
   - Назначение: Проверяет и выполняет транзакцию.
   - Реализация:
     - Находит кошельки отправителя и получателя.
     - Проверяет лимиты и баланс.
     - Если всё в порядке — списывает сумму с комиссией и зачисляет получателю.
     - Добавляет транзакцию в историю.

Код:
```
bool processTransaction(Transaction* tx) {
    // Получаем кошельки отправителя и получателя
    Wallet* senderWallet = nullptr;
    Wallet* receiverWallet = nullptr;
    Client* senderClient = nullptr;
    Client* receiverClient = nullptr;
    // Получаем всех клиентов из BST
    vector<Client*> allClients = clients.getAllClients();
    // Находим клиентов и их кошельки
    for (size_t i = 0; i < allClients.size(); i++) {
        Client* client = allClients[i];
        Entity* wallet = client->getWallets().getEntity(tx->getSenderWalletId());
        if (wallet) {
            senderWallet = dynamic_cast<Wallet*>(wallet);
            senderClient = client;
        }
        wallet = client->getWallets().getEntity(tx->getReceiverWalletId());
        if (wallet) {
            receiverWallet = dynamic_cast<Wallet*>(wallet);
            receiverClient = client;
        }
    }
    if (!senderWallet || !receiverWallet || !senderClient || !receiverClient) {
        return false;
    }
    // Проверяем лимит транзакции
    if (tx->getAmount() > senderClient->getMaxTransactionLimit()) {
```

```
        return false;
    }
    // Рассчитываем комиссию
    double commission = senderClient->calculateCommission(tx->getAmount());
    double totalAmount = tx->getAmount() + commission;
    // Проверяем баланс
    if (senderWallet->getBalance() < totalAmount) {
        return false;
    }
    // Выполняем транзакцию
    senderWallet->withdraw(totalAmount);
    receiverWallet->deposit(tx->getAmount());
    // Добавляем транзакцию в историю
    transactions.addTransaction(tx);
    return true;
}
```

3. void displayClients()
    - Параметры: Нет.
    - Возвращаемое значение: Нет.
    - Назначение: Выводит список клиентов.
    - Реализация:
        - Использует метод displayInOrder() из ClientBST.

Код:
```
void displayClients() const {
    clients.displayInOrder();
}
```

4. void displayTransactions()
    - Параметры: Нет.
    - Возвращаемое значение: Нет.
    - Назначение: Выводит список транзакций.
    - Реализация:
        - Использует метод displayTransactions() из TransactionList.

Код:
```
void displayTransactions() const {
    transactions.displayTransactions();
}
```

5. void saveDataToFiles()
    - Параметры: Нет.
    - Возвращаемое значение: Нет.
    - Назначение: Сохраняет данные в файлы.
    - Реализация:
        - Открывает файлы Clients.txt и Blockchain_transactions.txt.
        - Записывает данные в текстовом формате.

Код:
```
void saveDataToFiles() {
    saveClientsToFile("Clients.txt");
    saveTransactionsToFile("Blockchain_transactions.txt");
}
```

6. void loadDataFromFiles()
    - Параметры: Нет.
    - Возвращаемое значение: Нет.
    - Назначение: Загружает данные из файлов.
    - Реализация:
        - Читает файлы и восстанавливает объекты (Client, Wallet, Transaction).

Код:
```
void loadDataFromFiles() {
```

```
        loadClientsFromFile("Clients.txt");
        loadTransactionsFromFile("Blockchain_transactions.txt");
    }
```

7. void saveClientsToFile(const char* filename)
    - Параметры: char* filename – указатель на название файла.
    - Возвращаемое значение: Нет.
    - Назначение: Сохранение клиентов в файл.
    - Реализация:

Код:
```
void saveClientsToFile(const char* filename) {
    FILE* file = fopen(filename, "w");
    if (!file) {
        perror("Failed to open clients file for writing");
        return;
    }
    vector<Client*> allClients = clients.getAllClients();
    for (Client* client : allClients) {
        // Сохраняем тип клиента, ID и имя
        const char* clientType = "";
        if (dynamic_cast<GoldClient*>(client)) {
            clientType = "Gold";
        }
        else if (dynamic_cast<PlatinumClient*>(client)) {
            clientType = "Platinum";
        }
        else if (dynamic_cast<StandardClient*>(client)) {
            clientType = "Standard";
        }
        fprintf(file, "%s %s %s\n", clientType, client->getId().c_str(), client-
>getName().c_str());
        // Сохраняем кошельки клиента
        vector<Entity*>& wallets = client->getWallets().getAllEntities();
        for (Entity* e : wallets) {
            Wallet* w = dynamic_cast<Wallet*>(e);
            if (w) {
                fprintf(file, "Wallet %s %.2f\n", w->getId().c_str(), w->getBalance());
            }
        }
        fprintf(file, "EndClient\n");
    }
    fclose(file);
}
```

8. void saveTransactionsToFile(const char* filename)
    - Параметры: char* filename – указатель на название файла.
    - Возвращаемое значение: Нет.
    - Назначение: Сохранение транзакций в файл.
    - Реализация:

Код:
```
void saveTransactionsToFile(const char* filename) {
    FILE* file = fopen(filename, "w");
    if (!file) {
        perror("Failed to open transactions file for writing");
        return;
    }
```

```
    TransactionNode* current = transactions.getHead();
    while (current) {
        Transaction* tx = current->data;
        fprintf(file, "%s %s %s %.2f %.2f\n",
            tx->getId().c_str(),
            tx->getSenderWalletId().c_str(),
            tx->getReceiverWalletId().c_str(),
            tx->getAmount(),
            tx->getCommission());
        current = current->next;
    }
    fclose(file);
}
```

9. void loadClientsFromFile(const char* filename)
    - Параметры: char* filename – указатель на название файла.
    - Возвращаемое значение: Нет.
    - Назначение: Загрузка клиентов из файла.
    - Реализация:

Код:

```
void loadClientsFromFile(const char* filename) {
    FILE* file = fopen(filename, "r");
    if (!file) {
        perror("Failed to open clients file for reading");
        return;
    }
    char clientType[20];
    char id[50];
    char name[100];
    Client* currentClient = nullptr;
    while (fscanf(file, "%19s", clientType) == 1) {
        if (strcmp(clientType, "Wallet") == 0) {
            // Чтение кошелька
            char walletId[50];
            double balance;
            if (fscanf(file, "%49s %lf", walletId, &balance) == 2 && currentClient) {
                currentClient->addWallet(new Wallet(walletId, currentClient->getId(), balance));
            }
        }
        else if (strcmp(clientType, "EndClient") == 0) {
            currentClient = nullptr;
        }
        else {
            // Чтение нового клиента
            if (fscanf(file, "%49s %99[^\n]", id, name) == 2) {
                if (strcmp(clientType, "Gold") == 0) {
                    currentClient = new GoldClient(id, name);
                }
                else if (strcmp(clientType, "Platinum") == 0) {
                    currentClient = new PlatinumClient(id, name);
                }
                else if (strcmp(clientType, "Standard") == 0) {
                    currentClient = new StandardClient(id, name);
```

```
            }
            if (currentClient) {
                clients.insert(currentClient);
            }
        }
    }
}
fclose(file);
}
```

10. void loadTransactionsFromFile(const char* filename)
    - Параметры: char* filename – указатель на название файла.
    - Возвращаемое значение: Нет.
    - Назначение: Загрузка транзакций из файла.
    - Реализация:

Код:
```
void loadTransactionsFromFile(const char* filename) {
    FILE* file = fopen(filename, "r");
    if (!file) {
        perror("Failed to open transactions file for reading");
        return;
    }
    char txId[50];
    char senderId[50];
    char receiverId[50];
    double amount;
    double commission;
    while (fscanf(file, "%49s %49s %49s %lf %lf",
        txId, senderId, receiverId, &amount, &commission) == 5) {
        Transaction* tx = new Transaction(txId, senderId, receiverId,
            amount, TxType::TRANSFER, commission);
        transactions.addTransaction(tx);
    }
    fclose(file);
}
```

8. Класс TransactionList (Двусвязный список транзакций)

1. TransactionList()
    - Параметры: Нет.
    - Возвращаемое значение: Нет.
    - Назначение: Конструктор, инициализирует пустой список.
    - Реализация:
        - Устанавливает head = nullptr, tail = nullptr, size = 0.

Код:
```
TransactionList() : head(nullptr), tail(nullptr), size(0) {}
```

2. void addTransaction(Transaction* tx)
    - Параметры: tx — указатель на транзакцию.
    - Возвращаемое значение: Нет.
    - Назначение: Добавляет транзакцию в конец списка.
    - Реализация:
        - Создает новый узел TransactionNode.
        - Если список пуст (head == nullptr), делает новый узел головой и хвостом.
        - Иначе добавляет узел в конец (tail->next = newNode, newNode->prev = tail).
        - Увеличивает size на 1.

Код: void addTransaction(Transaction* tx) {
```
    TransactionNode* newNode = new TransactionNode(tx);
    if (!head) {
        head = tail = newNode;
    }
    else {
        tail->next = newNode;
        newNode->prev = tail;
        tail = newNode;
    }
    size++;
}
```
3. bool removeTransaction(const string& id)
    - Параметры: id — ID транзакции для удаления.
    - Возвращаемое значение: bool — true, если удаление успешно.
    - Назначение: Удаляет транзакцию по ID.
    - Реализация:
        - Ищет узел с заданным id (перебор от head до tail).
        - Если находит:
            - Обновляет связи prev и next соседних узлов.
            - Если удаляемый узел — голова или хвост, обновляет head/tail.
            - Удаляет узел, уменьшает size, возвращает true.
        - Если не находит — возвращает false.

Код: bool removeTransaction(const string& id) {
```
    TransactionNode* current = head;
    while (current) {
        if (current->data->getId() == id) {
            if (current->prev) current->prev->next = current->next;
            if (current->next) current->next->prev = current->prev;
            if (current == head) head = current->next;
            if (current == tail) tail = current->prev;
            delete current;
            size--;
            return true;
        }
        current = current->next;
    }
    return false;
}
```
4. Transaction* getTransaction(const string& id)
    - Параметры: id — ID транзакции.
    - Возвращаемое значение: Transaction* — указатель на транзакцию или nullptr.
    - Назначение: Поиск транзакции по ID.
    - Реализация:
        - Перебирает узлы от head до tail, сравнивает id.
        - Возвращает найденную транзакцию или nullptr.

Код: Transaction* getTransaction(const string& id) {
```
    TransactionNode* current = head;
    while (current) {
        if (current->data->getId() == id) {
            return current->data;
```

```
        }
        current = current->next;
    }
    return nullptr;
}
```

5. void displayTransactions()
    - Параметры: Нет.
    - Возвращаемое значение: Нет.
    - Назначение: Выводит все транзакции в консоль.
    - Реализация:
        - Перебирает узлы, для каждого вызывает tx->getDetails() и выводит результат.

Код:
```
void displayTransactions() const {
    TransactionNode* current = head;
    while (current) {
        cout << current->data->getDetails() << endl;
        current = current->next;
    }
}
```

6. ~TransactionList()
    - Параметры: Нет.
    - Возвращаемое значение: Нет.
    - Назначение: Деструктор, освобождает память.
    - Реализация:
        - Последовательно удаляет все узлы, начиная с head.

Код:
```
~TransactionList() {
    TransactionNode* current = head;
    while (current) {
        TransactionNode* next = current->next;
        delete current;
        current = next;
    }
}
```

9. Класс ClientBST (Двоичное дерево поиска клиентов)

1. ClientBST()
    - Параметры: Нет.
    - Возвращаемое значение: Нет.
    - Назначение: Конструктор, инициализирует пустое дерево.
    - Реализация:
        - Устанавливает root = nullptr.

Код:
```
ClientBST() : root(nullptr) {}
```

2. void insert(Client* client)
    - Параметры: client — указатель на клиента.
    - Возвращаемое значение: Нет.
    - Назначение: Добавляет клиента в дерево.
    - Реализация:
        - Рекурсивно ищет место для вставки, сравнивая client->getTotalBalance().
        - Если balance нового клиента меньше текущего узла — идёт влево, иначе — вправо.
        - Находит пустое место (nullptr) и создает новый ClientNode.

Код:
```
void insert(Client* client) {
```

```
    insertHelper(root, client);
}
```

3. bool remove(const string& id)
    - Параметры: id — ID клиента для удаления.
    - Возвращаемое значение: bool — true, если удаление успешно.
    - Назначение: Удаляет клиента по ID.
    - Реализация:
        - Рекурсивно ищет узел с заданным id.
        - Если у узла нет детей или только один — удаляет его напрямую.
        - Если два ребенка — заменяет удаляемый узел минимальным элементом из правого поддерева.

Код:
```
bool remove(const string& id) {
    int oldSize = countNodes(root);
    root = removeHelper(root, id);
    return countNodes(root) < oldSize;
}
```

4. Client* find(const string& id)
    - Параметры: id — ID клиента.
    - Возвращаемое значение: Client* — указатель на клиента или nullptr.
    - Назначение: Поиск клиента по ID.
    - Реализация:
        - Рекурсивно обходит дерево, сравнивая id.
        - Возвращает найденный объект или nullptr.

Код:
```
Client* find(const string& id) const {
    return findHelper(root, id);
}
```

5. void displayInOrder()
    - Параметры: Нет.
    - Возвращаемое значение: Нет.
    - Назначение: Выводит клиентов в отсортированном порядке (по балансу).
    - Реализация:
        - Рекурсивный обход: левое поддерево → текущий узел → правое поддерево.
        - Для каждого узла выводит client->getId() и client->getTotalBalance().

Код:
```
void displayInOrder() const {
    inOrderHelper(root);
}
```

6. vector<Client*> getAllClients()
    - Параметры: Нет.
    - Возвращаемое значение: vector<Client*> — вектор всех клиентов.
    - Назначение: Возвращает клиентов в порядке возрастания баланса.
    - Реализация:
        - Рекурсивно обходит дерево (как в displayInOrder), заполняя вектор.

Код:
```
vector<Client*> getAllClients() const {
    vector<Client*> result;
    getAllClientsHelper(root, result);
    return result;
}
```

7. ~ClientBST()
    - Параметры: Нет.
    - Возвращаемое значение: Нет.
    - Назначение: Деструктор, освобождает память.

- Реализация:
  - Рекурсивно удаляет все узлы (пост-порядок обхода: дети → корень).

Код: ~ClientBST() {
clearHelper(root);
}

Тестирование:

1. Загрузка данных о кошельках и клиентах с файла Clients.txt
2. Перевод между кошельками:
   - Успех: баланс ≥ суммы + комиссия.
   - Неудача: недостаточно средств.
3. Сохранение операций и изменненых данных на файлы Clients.txt и Blockchain_transactions.txt

Результат:

Clients after transactions:

Client c1: Alice, Balance: 4425

Client c2: Bob, Balance: 3475

Client c3: Charlie, Balance: 3000

Заключение:

- **Изучено:** Принципы ООП, работа с деревьями и списками.
- **Освоено:** Паттерны проектирования, обработка транзакций.
- **Реализовано:** Система блокчейна с клиентами, кошельками и транзакциями.