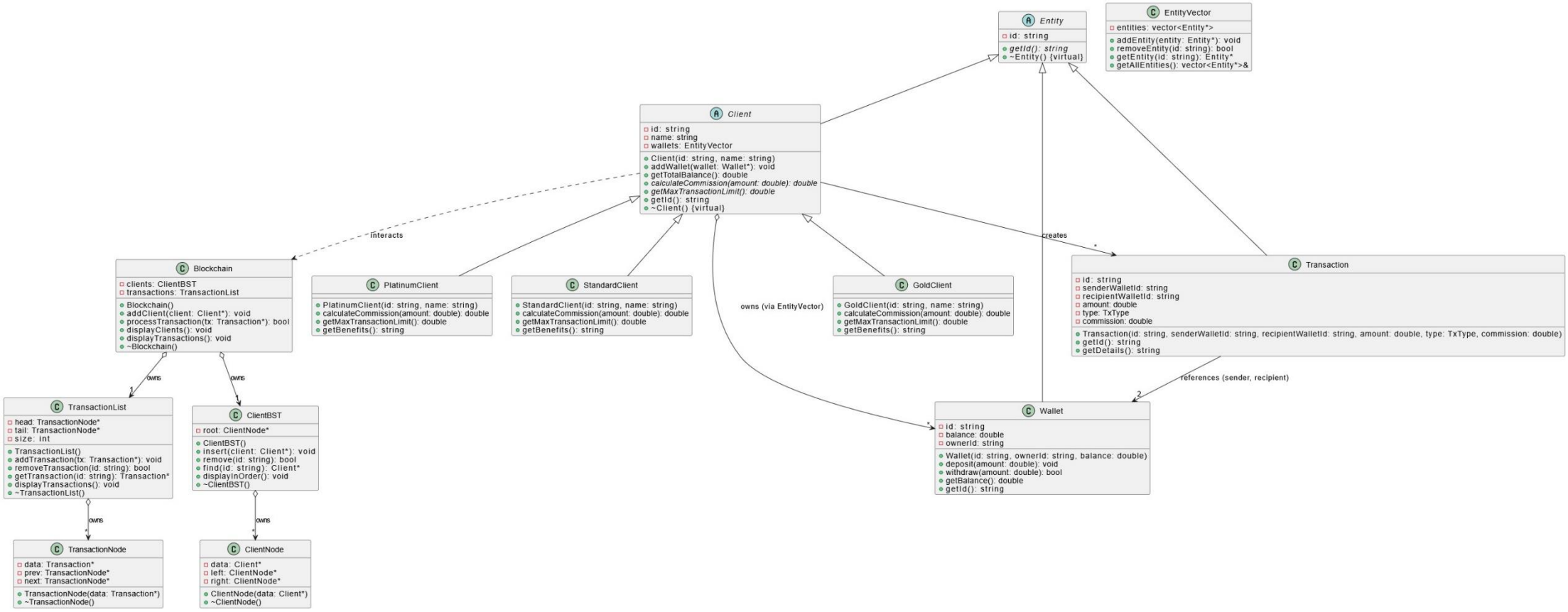
Курсовая Работа – Лаборатория Основы Программирования и Алгоритмизации

УПРОЩЕННАЯ СИМУЛЯЦИЯ СИСТЕМЫ БЛОКЧЕЙН-ТРАНЗАКЦИЙ

Преподаватель: Хольгер Эспинола Ривера

UML-ДИАГРАММА:



1. Описание отношений

- Наследование:

- Entity ← Client, Wallet, Transaction: Entity наследуются от Entity для управления идентификаторами.

- Client ← GoldClient, PlatinumClient, StandardClient: типы Client переопределяют методы комиссии и лимита.

- Состав:

- Blockchain → ClientBST: Blockchain владеет ClientBST (1 к 1).

- Blockchain → TransactionList: Blockchain владеет TransactionList (1 к 1).

- ClientBST → ClientNode: ClientBST владеет объектами ClientNode (1 к *).

- TransactionList → TransactionNode: TransactionList владеет объектами TransactionNode (1 к *).

- Агрегация:

- Client → Wallet: Client владеет Wallets через EntityVector (1 к *), но Wallets могут быть переназначены.

- Ассоциация:

- Клиент ↔ Блокчейн: Клиенты взаимодействуют с Блокчейном для транзакций и запросов (от * до 1).

- Клиент → Транзакция: Клиенты создают Транзакции (от 1 до *).

- Транзакция → Кошелек: Транзакции ссылаются на кошельки отправителя и получателя (от 1 до 2).

2. Примечания:

- Атрибуты: Частный (-), Публичный (+).

- Множественность: 1 (один), * (много).
- TxType: Enum в транзакции (TRANSFER).
- EntityVector управляет отношениями «один ко многим» (например, кошельки в Client).
- ClientBST упорядочивает клиентов по общему балансу.
- TransactionList использует двусвязный список для транзакций.

- **Описание классов, атрибутов и методов**

Ниже приведены упрощенные спецификации классов:

1. Entity (абстрактный базовый класс):

o Назначение: базовый класс для сущностей (Клиент, Кошелек, Транзакция).

o Атрибуты:

- id: строка (уникальный идентификатор)

o Методы:

- getId(): строка (чисто виртуальная)
- ~Entity() (виртуальный деструктор)

o Аннотация: Да.

2. EntityVector:

o Назначение: управляет отношением «один ко многим» для сущностей (например, кошельков, принадлежащих клиенту).

o Атрибуты:

- сущности: vector<Entity*> (хранит указатели сущностей)

o Методы:

- addEntity(entity: Entity*): void
- removeEntity(id: string): bool
- getEntity(id: string): Entity*
- getAllEntities(): vector<Entity*>&

3. Client (абстрактный базовый класс):

o Назначение: базовый класс для клиентских типов.

o Атрибуты:

- id: string (унаследовано от Entity)
- name: string
- wallets: EntityVector (хранит кошельки)

o Методы:

- Client(id: string, name: string)
- addWallet(wallet: Wallet*): void

- getTotalBalance(): double (сумма балансов кошельков)
- calculateCommission(amount: double): double (чисто виртуальный)
- getMaxTransactionLimit(): double (чисто виртуальный)
- getId(): string (переопределение)
- ~Client() (виртуальный деструктор)

4. GoldClient (наследуется от Client):

o Назначение: Премиум-клиенты с низкими комиссиями.

o Атрибуты: (Наследуется)

o Методы:

- GoldClient(id: string, name: string)
- calculateCommission(amount: double): double (например, 1%)
- getMaxTransactionLimit(): double (например, 10000)
- getBenefits(): string («Приоритетная обработка, низкие комиссии»)

5. PlatinumClient (Наследуется от Client):

o Назначение: клиенты среднего уровня с умеренными комиссиями.

o Атрибуты: (Наследуется)

o Методы:

- PlatinumClient(id: string, name: string)

- calculateCommission(amount: double): double (например, 2%)
- getMaxTransactionLimit(): double (например, 5000)
- getBenefits(): string («Бонусные вознаграждения, умеренные комиссии»)

6. StandardClient (Наследуется от Client):

o Назначение: Стандартные клиенты с более высокими комиссиями.

o Атрибуты: (Наследуется)

o Методы:

- StandardClient(id: string, name: string)
- calculateCommission(amount: double): double (например, 5%)
- getMaxTransactionLimit(): double (например, 1000)
- getBenefits(): string («Стандартный доступ»)

7. Wallet (Наследуется от Entity):

o Назначение: Хранит средства для транзакций.

o Атрибуты:

- id: string (унаследовано)
- balance: double
- ownerId: string (идентификатор клиента)

o Методы:

- Wallet(id: string, ownerId: string, balance: double)

- deposit(amount: double): void

- withdraw(amount: double): bool

- getBalance(): double

- getId(): string (переопределить)

8. Transaction (наследуется от Entity):

o Назначение: представляет собой перевод между двумя кошельками.

o Атрибуты:

- id: string (унаследовано)

- senderWalletId: string (ID кошелька отправителя)

- receiveWalletId: string (ID кошелька получателя)

- amount: double

- type: TxType (enum: {TRANSFER})

- commission: double

o Методы:

- Transaction(id: string, senderWalletId: string, receiveWalletId: string, amount: double, type: TxType, commission: double)

- getId(): string (переопределение)

- getDetails(): string

9. TransactionNode:

- o Назначение: Узел для двусвязного списка транзакций.
- o Атрибуты:
  - data: Transaction*
  - prev: TransactionNode*
  - next: TransactionNode*
- o Методы:
  - TransactionNode(data: Transaction*)
  - ~TransactionNode()

10. TransactionList:

- o Назначение: Двусвязный список для транзакций.
- o Атрибуты:
  - head: TransactionNode*
  - tail: TransactionNode*
  - size: int
- o Методы:
  - TransactionList()
  - addTransaction(tx: Transaction*): void

- removeTransaction(id: string): bool
- getTransaction(id: string): Transaction*
- displayTransactions(): void
- ~TransactionList()

11. ClientNode:

o Назначение: Узел для двоичного дерева поиска клиентов.

o Атрибуты:

- data: Client*
- left: ClientNode*
- right: ClientNode*

o Методы:

- ClientNode(data: Client*)
- ~ClientNode()

12. ClientBST:

o Назначение: Двоичное дерево поиска для клиентов, упорядоченное по балансу.

o Атрибуты:

- root: ClientNode*

o Методы:

- ClientBST()
- insert(client: Client*): void
- remove(id: string): bool
- find(id: string): Client*
- displayInOrder(): void
- ~ClientBST()

13. BlockChain:

o Назначение: Управление клиентами и транзакциями.

o Атрибуты:

- клиенты: ClientBST
- транзакции: TransactionList

o Методы:

- Blockchain()
- addClient(client: Client*): void
- processTransaction(tx: Transaction*): bool
- displayClients(): void
- displayTransactions(): void
- ~Blockchain()

- **Инструкции**

Реализуйте систему, которая имитирует креативную и очень упрощенную версию операций финансовых транзакций BlockChain между кошельками Клиентов. Ядром системы является BlockChain, который содержит информацию о Клиентах и транзакциях. Необходимо выполнять операции депозитов и снятий. Реализуйте систему на C++, следуя парадигме ООП, используя классы, методы, конструкторы, деструкторы и отношения между классами, такие как наследование, ассоциация, агрегация или композиция, когда это необходимо. Используйте структуры данных поиска по двоичному дереву и двухсвязного списка. Подготовьте необходимые методы, которые могут создавать отчеты о списке клиентов и списке транзакций, содержащихся в BlockChain. Если хотите, вы можете добавить или упростить идеи, выраженные в диаграмме UML и этом описании (не обязательно реализовывать всю схему — все классы, атрибуты или методы, но наиболее критические идеи и функции являются обязательными). Сохраните информацию о клиентах и информацию о транзакциях в 2 файлах: **Clients.txt** и **Blockchain_transactions.txt**, используя функциональные возможности для чтения и записи в файлы с помощью функций **fopen**, **fscanf**, **fprintf** и **fclose**, используемых в уроке 09 1-го семестра по курсу «Алгоритмизация и программирование». Задание должно быть выполнено в **группах по 4 студента**. Отчет должен быть написан в соответствии с инструкциями, данными преподавателем теории. Презентация состоится в день оценки лабораторной работы. Проект должен быть опубликован **в репозитории Github** лидера группы. В день презентации необходимо прийти с распечатанной версией отчет.

- **Комментарии**

- Реализация **Entity Vector** может быть основана на структуре данных и алгоритмах, определенных в **lab02_q3**. Определите один класс для каждой сущности, а затем реализуйте некоторый векторизованный класс этой сущности.

- Реализация классов **Client Node** и **Client BST** может быть основана на структуре данных и алгоритмах, определенных в **lab06_q1**. Код должен иметь некоторые адаптации, чтобы работать с объектами класса **Client**. Необходимо выбрать одно числовой атрибут из класса Client, который может быть полезен для применения инварианта процесса построения бинарного дерева.

- Реализация классов **Transaction Node** и **Transaction List** может быть основана на структуре данных и алгоритмах, определенных в **lab06_q3**. Та же идея **Double Linked List** для объектов класса **Circle** может быть реализована с учетом надлежащей структуры и отношений, которые имеет класс **Transaction**.

- Реализация функций чтения и записи в файлах может осуществляться с некоторыми адаптациями, следуя **week09** (Github репозитории **Algorithms-5130203**).