ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

***«*САНКТ-ПЕТЕРБУРГСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

**ПЕТРА ВЕЛИКОГО»**

**Институт компьютерных наук и кибербезопасности**

**Отчет по курсовой работе**

|  |
| --- |
| Куликова Ангелина  Денискина Анна  Дымерец Софья  Еникеева Мария |

*(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 алгоритмизация |

*(наименование дисциплины)*

**Оценка:**

Преподаватель: */*Хольгер Эспинола Ривера /

Обучающиеся: / Куликова Ангелина

Денискина Анна

Дымерец Софья

Еникеева Мария/

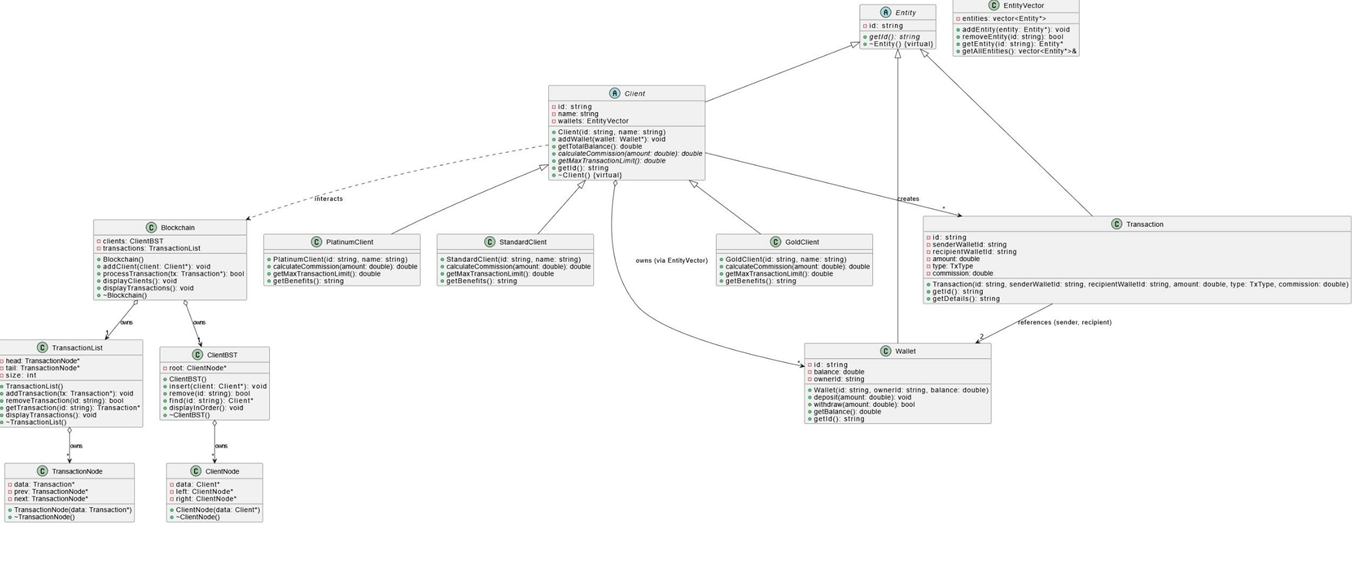
Дата: 07.06.25

**ПОСТАНОВКА ЗАДАЧ**

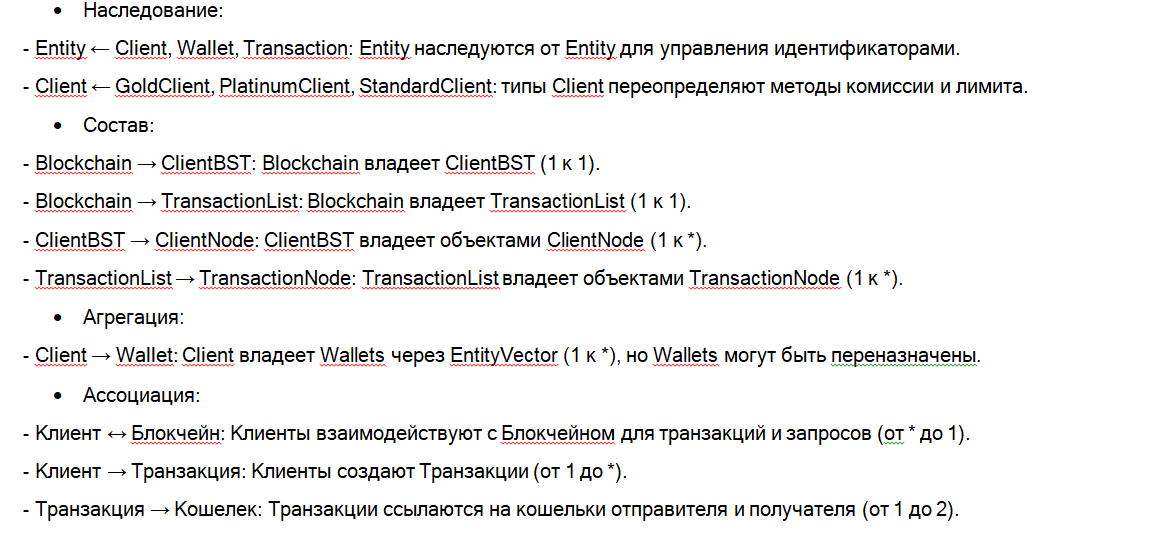
Реализовать систему, которая имитирует креативную и очень упрощенную версию операций финансовых транзакций BlockChain между кошельками Клиентов. Необходимо выполнять операции депозитов и снятий. Реализовать систему на C++, следуя парадигме ООП, используя классы, методы, конструкторы, деструкторы и отношения между классами, такие как наследование, ассоциация, агрегация или композиция, когда это необходимо. Использовать структуры данных поиска по двоичному дереву и двухсвязного списка. Подготовить необходимые методы, которые могут создавать отчеты о списке клиентов и списке транзакций, содержащихся в BlockChain. Написать отчёт по выполненной работе.

**ИСХОДНЫЕ ДАННЫЕ**

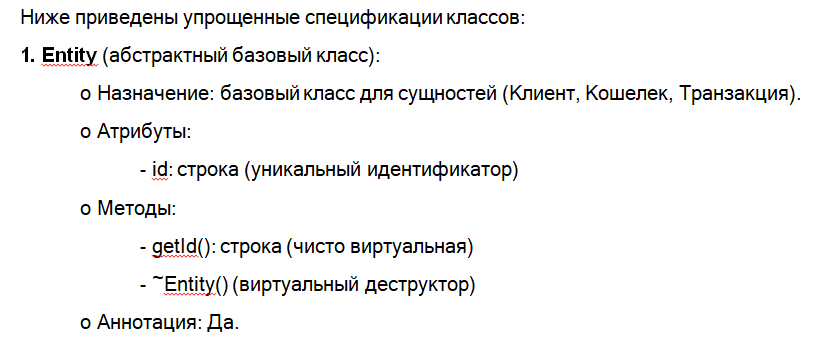
1. UML-ДИАГРАММА



1. Описание отношений



1. Описание классов, атрибутов и методов



**ОПИСАНИЕ КАЖДОГО РЕАЛИЗОВАННОГО МЕТОДА**

1. КЛАСС ENTITY

**метод:** Entity(){};

**параметры:** Отсутствуют

**возвращаемое значение:** Не возвращает значение (конструктор)

**описание предназначения метода:** Конструктор класса Entity, который используется для инициализации объекта данного класса. Обычно служит для установки значений по умолчанию при создании объекта.

**описание реализации:** Метод представляет собой конструктор без параметров и без тела, то есть не выполняет никаких действий. Он позволяет создавать объект класса Entity с базовой инициализацией по умолчанию.

**программный код:** ****

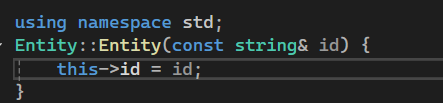
1. **метод:** Entity(const string& id)

**параметры:** const string& id — ссылка на строку, содержащую идентификатор сущности.

**возвращаемое значение:** Не возвращает значение (конструктор)

**описание предназначения метода:** Конструктор класса Entity, инициализирующий объект с заданным идентификатором.

**описание реализации:** Метод принимает строковый параметр id и устанавливает внутреннее поле объекта с этим значением. Обычно в теле конструктора происходит присвоение значения параметра в соответствующее поле класса (например, this->id = id).

**программный код:** ****

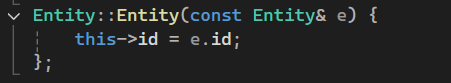
1. **метод:** Entity(const Entity& e);

**параметры:** const Entity& e — ссылка на объект другого класса Entity, который копируется.

**возвращаемое значение:** Не возвращает значение (конструктор)

**описание предназначения метода:** Конструктор копирования, создаёт новый объект Entity, копируя данные из существующего объекта e.

**описание реализации:** Метод получает ссылку на объект e, после чего копирует значения всех важных полей из объекта e в создаваемый объект. Обычно реализуется для правильного клонирования объекта, чтобы избежать ошибок с указателями или разделением ресурсов.

**программный код:** ****

1. **метод:** virtual ~Entity() {}

**параметры:** Отсутствуют

**возвращаемое значение:** Не возвращает значение

**описание предназначения метода:** виртуальный деструктор обеспечивает корректное удаление объектов производных классов через указатель базового класса.

**описание реализации:** Для пустого виртуального деструктора — метод не содержит тела, реализует механизм виртуального вызова деструктора.

**программный код:** ****

1. **метод:** virtual const string& getId() = 0;

**параметры:** Отсутствуют

**возвращаемое значение:** Возвращает константную ссылку на объект типа string.

**описание предназначения метода:** Метод предназначен для получения уникального идентификатора объекта в виде строки. Объявлен как чисто виртуальный, что требует обязательной реализации в производных классах.

**описание реализации:** Реализация метода отсутствует в базовом классе (он абстрактный). В производных классах должен быть переопределён для возврата внутреннего идентификатора объекта.

**программный код: **

1. КЛАСС ENTITYVECTOR
2. **метод:** EntityVector() {}

**параметры:** Отсутствуют

**возвращаемое значение:** нет (конструктор)

**описание предназначения метода:** Конструктор по умолчанию класса EntityVector; предназначен для создания объекта с начальными значениями.

**описание реализации:** Конструктор не принимает параметров и не выполняет никаких действий внутри (тело пустое {}). Обычно используется для инициализации объекта без настроек.

**программный код:** ****

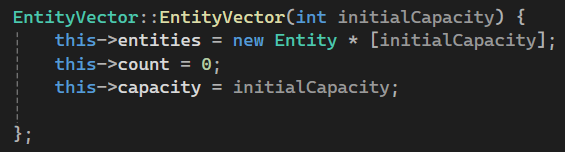
1. **метод:** EntityVector(int initialCapacity);

**параметры:**   int initialCapacity — начальная ёмкость (размер) внутреннего контейнера вектора.

**возвращаемое значение:** нет (конструктор)

**описание предназначения метода:** Конструктор предназначен для создания объекта EntityVector с заданным начальным размером ёмкости, что позволяет заранее выделить память под определённое количество элементов.

**описание реализации:** При вызове конструктор принимает значение initialCapacity и инициализирует внутренние структуры данных (например, массив или вектор) так, чтобы вместить указанное количество элементов без перераспределения памяти.

**программный код:** ****

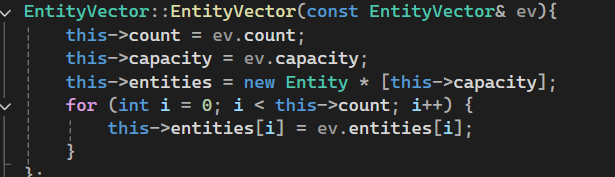
1. **метод:** EntityVector(const EntityVector& ev);

**параметры:** const EntityVector& ev — ссылка на существующий объект EntityVector, из которого происходит копирование.

**возвращаемое значение:** нет (конструктор)

**описание предназначения метода:** Конструктор копирования создаёт новый объект EntityVector как точную копию переданного объекта ev, включая копирование всех элементов и внутреннего состояния.

**описание реализации:** В конструкторе копирования выделяется новая память для хранения элементов такого же размера, как у исходного объекта ev. Затем производится копирование всех элементов из ev в новый объект, обеспечивая независимость двух объектов по памяти.

**программный код:** ****

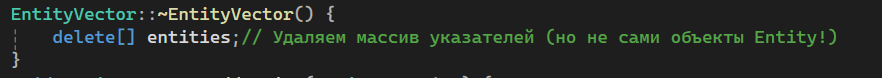
1. **метод:** ~EntityVector();

**параметры:** Отсутствуют

**возвращаемое значение:** нет

**описание предназначения метода:** Деструктор отвечает за корректное освобождение всех ресурсов, которые были выделены объектом EntityVector во время его жизни. Обычно это освобождение динамической памяти, выделенной под элементы контейнера.

**описание реализации:** При вызове деструктора память, выделенная под внутренний массив элементов, освобождается через оператор delete[]. После чего объект считается уничтоженным, предотвращая утечку памяти.

**программный код:** ****

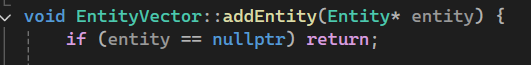
1. **метод:** void addEntity(Entity\* entity);

**параметры:**   Entity\* entity — указатель на объект типа Entity, который необходимо добавить.

**возвращаемое значение:** нет (конструктор)

**описание предназначения метода:**  Метод предназначен для добавления нового объекта Entity в коллекцию (вектор, массив или другой контейнер) внутри класса EntityVector.

**описание реализации:** При вызове функция принимает указатель на сущность, проверяет наличие места в контейнере, при необходимости расширяет вместимость, затем добавляет переданный объект в конец внутренней структуры данных.

**программный код:** ****

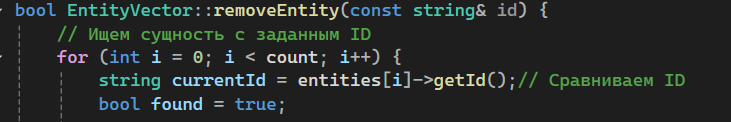
1. **метод:** bool removeEntity(const string& id);

**параметры:**   const string& id — константная ссылка на строку, содержащую уникальный идентификатор сущности, которую нужно удалить.

**возвращаемое значение:**   bool — возвращает true, если сущность с заданным идентификатором была найдена и успешно удалена; false — если объект с таким идентификатором отсутствует.

**описание предназначения метода:** Метод предназначен для удаления объекта Entity из коллекции по его уникальному идентификатору.

**описание реализации:** Функция ищет в контейнере объект с совпадающим id, при нахождении удаляет его из структуры данных (удаляет ссылку или указатель), сдвигает оставшиеся элементы (если требуется) и уменьшает счетчик элементов.

**программный код:** ****

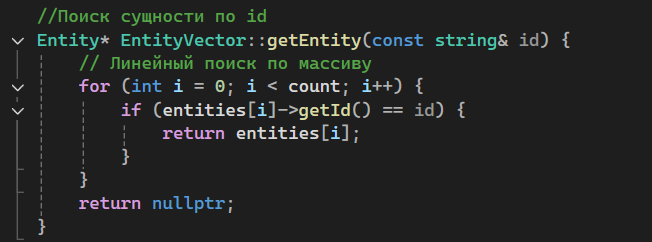
1. **метод:** Entity\* getEntity(const string& id);

**параметры:**  const string& id — константная ссылка на строку, которая содержит уникальный идентификатор искомой сущности.

**возвращаемое значение:**  Entity\* — указатель на найденный объект Entity с заданным идентификатором; nullptr, если объект с таким id отсутствует.

**описание предназначения метода:** Метод предназначен для поиска и получения указателя на сущность из коллекции по её уникальному идентификатору.

**описание реализации:** Функция выполняет перебор элементов коллекции, сравнивая идентификатор каждого объекта с заданным id. При совпадении возвращается указатель на соответствующий объект. Если объект не найден, возвращается nullptr.

**программный код:** ****

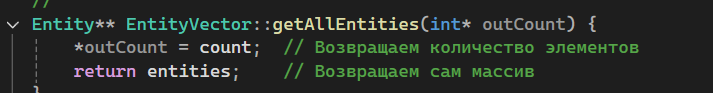
1. **метод:** Entity\*\* getAllEntities(int\* outCount);

**параметры:**   int\* outCount — указатель на целочисленную переменную, в которую функция запишет количество возвращаемых сущностей.

**возвращаемое значение**: Entity\*\* — указатель на массив указателей на объекты Entity.

**описание предназначения метода:** Функция предназначена для получения всех сущностей из коллекции вместе с информацией о количестве этих сущностей.

**описание реализации:** Метод возвращает массив указателей на все сущности, хранящиеся в коллекции. Одновременно в переменную, на которую указывает outCount, записывается число элементов в этом массиве.

**программный код:** ****

1. КЛАСС WALLET
2. **метод:** Wallet(const string& id, const string& ownerId, double balance);

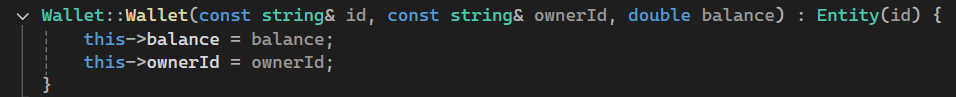
**параметры:**

- const string& id — константная ссылка на строку, идентификатор кошелька.   
  - const string& ownerId — константная ссылка на строку, идентификатор владельца кошелька.   
  - double balance — числовое значение с плавающей точкой, начальный баланс кошелька.

**возвращаемое значение**: нет (конструктор)

**описание предназначения метода:** Инициализация объекта Wallet с заданными значениями идентификатора, владельца и начального баланса.

**описание реализации:** Конструктор принимает три параметра и присваивает их соответствующим полям класса, создавая полностью инициализированный экземпляр Wallet.

**программный код:** ****

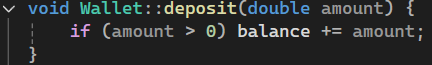
1. **метод:** void deposit(double amount);

**параметры:**   double amount — числовое значение с плавающей точкой, сумма для внесения на счёт.

**возвращаемое значение**: Метод класса, возвращающий ничего (void).

**описание предназначения метода:** Увеличение баланса кошелька на заданную сумму.

**описание реализации:** Метод принимает сумму amount и добавляет её к текущему значению баланса кошелька. Обычно перед добавлением проверяется корректность значения (например, сумма больше нуля).

**программный код:** ****

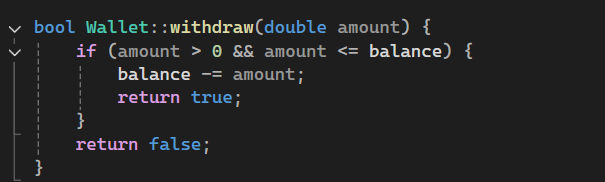
1. **метод:** bool withdraw(double amount);

**параметры:**   double amount — сумма в формате с плавающей точкой, которую нужно снять со счёта.

**возвращаемое значение**: Метод класса, возвращающий логическое значение (bool).

**описание предназначения метода:** Попытка уменьшить баланс кошелька на указанную сумму.

**описание реализации:** Метод проверяет, достаточно ли средств на счету для снятия суммы amount. Если баланс достаточен и сумма положительна, с баланса вычитается amount, и метод возвращает true (успешное снятие). Если средств недостаточно или сумма некорректна, баланс не меняется, метод возвращает false.

**программный код:** ****

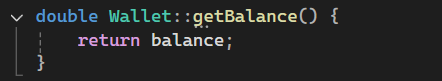
1. **метод:** double getBalance();

**параметры:**   Метод не принимает аргументов.

**возвращаемое значение**: Метод класса, возвращающий значение типа double

**описание предназначения метода:** Получить текущий баланс счёта или кошелька.

**описание реализации:** Метод возвращает числовое значение с плавающей точкой, соответствующее текущему состоянию баланса. Обычно не изменяет состояние объекта и доступен для чтения.

**программный код:** ****

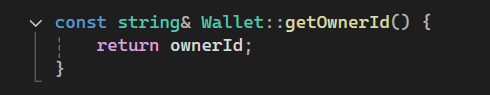
1. **метод:** const string& getOwnerId();

**параметры:**   Метод не принимает аргументов. переменную, в которую функция запишет количество возвращаемых сущностей.

**возвращаемое значение**: Метод класса, возвращающий константную ссылку на объект типа string.

**описание предназначения метода:** Получить уникальный идентификатор владельца (например, владельца счёта).

**описание реализации:** Метод возвращает ссылку на строку, которая содержит идентификатор владельца. При этом возвращаемое значение является константной ссылкой, что предотвращает изменение строки извне и повышает производительность за счёт отсутствия копирования.

**программный код:** ****

1. **метод:** const string& getId();

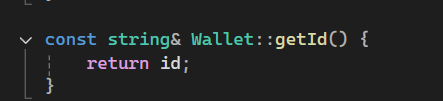
**параметры:**   нет

**возвращаемое значение**: константная ссылка на объект типа string

**описание предназначения метода:**  Метод предназначен для получения идентификатора (id) объекта в виде строки без возможности его изменения.

**описание реализации:** Метод возвращает ссылку на внутренний член класса (строку с идентификатором), при этом ссылка константная, что гарантирует невозможность модификации возвращаемой строки извне. Использование ссылки позволяет избежать копирования данных, что улучшает производительность.

**программный код:**



1. КЛАСС CLIENT
2. **метод:** Client(const string& id, const string& name)

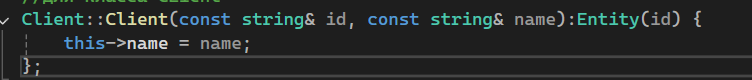
**параметры:**

- const string& id — константная ссылка на строку, идентификатор клиента   
  - const string& name — константная ссылка на строку, имя клиента

**возвращаемое значение**: отсутствует (конструктор)

**описание предназначения метода:**   Конструктор инициализирует объект класса Client, устанавливая значения идентификатора и имени клиента.

**описание реализации:**  Конструктор принимает две константные ссылки на строки и использует их для инициализации внутренних переменных класса, обычно через список инициализации. Использование ссылок предотвращает ненужное копирование строк при вызове конструктора.

**программный код:** 

1. **метод:** void addWallet(Wallet\* wallet);

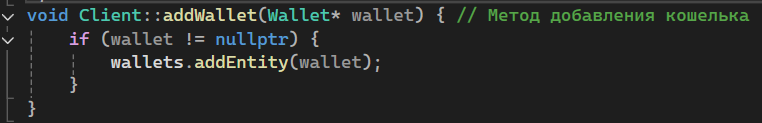
**параметры:**

Wallet\* wallet — указатель на объект класса Wallet, который необходимо добавить

**возвращаемое значение**: void (метод ничего не возвращает)

**описание предназначения метода:**   Метод предназначен для добавления нового объекта Wallet (кошелька) в коллекцию или список внутри класса. Обычно используется для управления множеством кошельков, связанных с клиентом или другим объектом.

**описание реализации:**   Метод принимает указатель на объект Wallet и добавляет его в внутреннюю структуру данных (например, в вектор или список). Такой подход позволяет работать с динамически созданными объектами и управлять ими через указатели.

**программный код:** ****

1. **метод:** double getTotalBalance();

**параметры:** отсутствуют

**возвращаемое значение**:  double — сумма балансов всех кошельков или счетов

**описание предназначения метода:**   Метод предназначен для вычисления и возврата общей суммы балансов всех объектов Wallet (кошельков), связанных с данным классом. Обычно используется для получения агрегированной информации о доступных средствах.

**описание реализации:**  Метод итерирует коллекцию указателей на Wallet, запрашивает у каждого текущий баланс и суммирует их. В конце возвращается итоговое значение типа double. Метод не модифицирует состояние объекта.

**программный код:** ****

1. **метод:** virtual double calculateCommission(double amount) = 0;

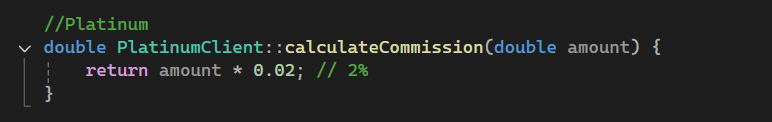
**параметры:** double amount — сумма, с которой нужно рассчитать комиссию

**возвращаемое значение**: double — величина комиссии для указанной суммы

**описание предназначения метода:**   Абстрактный (чисто виртуальный) метод предназначен для вычисления комиссии на основе переданной суммы. Определяется в базовом классе и должен быть реализован в производных классах с конкретной логикой расчёта комиссии.

**описание реализации:**   Метод объявлен как чисто виртуальный (= 0), значит базовый класс не содержит реализации, а указание типа virtual обеспечивает возможность переопределения. В производных классах реализуется конкретный алгоритм вычисления комиссии, который принимает сумму и возвращает значение комиссии.

**программный код:**



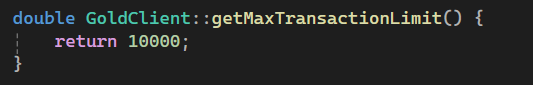
1. **метод:** virtual double getMaxTransactionLimit() = 0;

**параметры:** Метод не принимает параметров

**возвращаемое значение**: double — максимальный лимит одной транзакции

**описание предназначения метода:**   Чисто виртуальный метод предназначен для получения максимальной суммы, разрешённой для одной транзакции. Используется для определения ограничения на операции в производных классах с конкретными правилами лимитов.

**описание реализации:**   Метод объявлен как чисто виртуальный, что требует обязательного переопределения в наследниках. В базовом классе нет реализации, логика зависит от конкретной бизнес-модели, реализованной в потомках.

**программный код:** ****

1. **метод:** const string& getId();

**параметры:** Метод не принимает параметров

**возвращаемое значение**: Константная ссылка на объект типа string — идентификатор

**описание предназначения метода:**   Метод возвращает ссылку на строку, содержащую уникальный идентификатор объекта или сущности. Использование константной ссылки позволяет избежать копирования строки и защищает объект от изменения через возвращённый результат.

**описание реализации:**   Обычно метод возвращает ссылку на приватное поле класса, в котором хранится идентификатор. Метод объявляется без модификации состояния объекта (const-метод).

**программный код:**



1. **метод:** virtual ~Client() {}

**параметры:** Метод не принимает параметров

**возвращаемое значение**: Деструктор не возвращает значения

**описание предназначения метода:**   Виртуальный деструктор предназначен для корректного уничтожения объектов класса Client и его производных классов при удалении через указатель на базовый класс. Это гарантирует вызов правильного деструктора в иерархии наследования.

**описание реализации:**   Определён как виртуальный пустой деструктор, что позволяет производным классам переопределять его для добавления своей логики очистки ресурсов, а при удалении объекта через базовый указатель вызывается именно нужный деструктор.

**программный код:**



5. КЛАСС GoldClient

1. **метод:** GoldClient(const string& id, const string& name) : Client(id, name) {}

**параметры:**

const string& id – константная ссылка на строку – ID клиента

const string& name – константная ссылка на строку – имя клиента

**возвращаемое значение**: отсутствует (это конструктор)

**описание предназначения метода:** конструктор нужен чтобы инициализировать объект с помощью ссылок на значения идентификатора и имени клиента

**описание реализации:** Метод принимает две константные строки по ссылке, что позволяет инициализировать объект без лишнего копирования строк

**программный код:**

GoldClient(const string& id, const string& name) : Client(id, name) {}

1. **метод:** double calculateCommission(double amount);

**параметры:** double amount – сумма, от которой вычисляется комиссия

**возвращаемое значение:** double – сумма комиссии

**описание предназначения метода:** метод нужен для того, чтобы вычислить сумму комиссии от изначальной суммы (она составляет 1%)

**описание реализации:** искомый параметр умножается на 0,01, что позволяет рассчитать сумму комиссии

**программный код:**

double GoldClient::calculateCommission(double amount) {

return amount \* 0.01; // 1%

}

1. **метод:** double getMaxTransactionLimit();

**параметры:** отсутствуют

**возвращаемое значение:** double – максимально возможная сумма транзакции

**описание предназначения метода:** метод нужен для получения лимита одной транзакции, что в дальнейшем используется для ограничения операций для разных клиентов.

**описание реализации:** метод возвращает значение, равное лимиту одной транзакции (в данном случае это 10000)

**программный код:**

double GoldClient::getMaxTransactionLimit() {

return 10000;}

1. **метод:** string getBenefits();

**параметры:** отсутствуют

**возвращаемое значение:** string – строка, описывающая выгоды данного тарифа

**описание предназначения метода:** с помощью этого метода клиент может узнать о выгодах данного тарифа

**описание реализации:** метод возвращает строку, описывающую выгоду тарифа (в данном случае это “Приоритетная обработка, низкие комиссии”)

**программный код:**

string GoldClient::getBenefits() {

return “Приоритетная обработка, низкие комиссии”;

}

6. КЛАСС PlatinumClient

1. **метод:** PlatinumClient(const string& id, const string& name) : Client(id, name) {}

**параметры:**

const string& id – константная ссылка на строку – ID клиента

const string& name – константная ссылка на строку – имя клиента

**возвращаемое значение**: отсутствует (это конструктор)

**описание предназначения метода:** конструктор нужен чтобы инициализировать объект с помощью ссылок на значения идентификатора и имени клиента

**описание реализации:** Метод принимает две константные строки по ссылке, что позволяет инициализировать объект без лишнего копирования строк

**программный код:**

PlatinumClient(const string& id, const string& name) : Client(id, name) {}

1. **метод:** double calculateCommission(double amount);

**параметры:** double amount – сумма, от которой вычисляется комиссия

**возвращаемое значение:** double – сумма комиссии

**описание предназначения метода:** метод нужен для того, чтобы вычислить сумму комиссии от изначальной суммы (она составляет 2%)

**описание реализации:** искомый параметр умножается на 0,02, что позволяет рассчитать сумму комиссии

**программный код:**

double PlatinumClient::calculateCommission(double amount) {

return amount \* 0.02; // 2%

}

1. **метод:** double getMaxTransactionLimit();

**параметры:** отсутствуют

**возвращаемое значение:** double – максимально возможная сумма одной транзакции

**описание предназначения метода:** метод нужен для получения лимита одной транзакции, что в дальнейшем используется для ограничения операций для разных клиентов.

**описание реализации:** метод возвращает значение, равное лимиту одной транзакции (в данном случае это 5000)

**программный код:**

double PlatinumClient::getMaxTransactionLimit() {

return 5000;}

1. **метод:** string getBenefits();

**параметры:** отсутствуют

**возвращаемое значение:** string – строка, описывающая выгоды данного тарифа

**описание предназначения метода:** с помощью этого метода клиент может узнать о выгодах данного тарифа

**описание реализации:** метод возвращает строку, описывающую выгоду тарифа (в данном случае это “ Бонусные вознаграждения, умеренные комиссии”)

**программный код:**

string PlatinumClient::getBenefits() {

return “ Бонусные вознаграждения, умеренные комиссии”;

}

7. КЛАСС PlatinumClient

1. **метод:** StandartClient(const string& id, const string& name) : Client(id, name) {}

**параметры:**

const string& id – константная ссылка на строку – ID клиента

const string& name – константная ссылка на строку – имя клиента

**возвращаемое значение**: отсутствует (это конструктор)

**описание предназначения метода:** конструктор нужен чтобы инициализировать объект с помощью ссылок на значения идентификатора и имени клиента

**описание реализации:** Метод принимает две константные строки по ссылке, что позволяет инициализировать объект без лишнего копирования строк

**программный код:**

StandartClient(const string& id, const string& name) : Client(id, name) {}

1. **метод:** double calculateCommission(double amount);

**параметры:** double amount – сумма, от которой вычисляется комиссия

**возвращаемое значение:** double – сумма комиссии

**описание предназначения метода:** метод нужен для того, чтобы вычислить сумму комиссии от изначальной суммы (она составляет 5%)

**описание реализации:** искомый параметр умножается на 0,05, что позволяет рассчитать сумму комиссии

**программный код:**

double Standartlient::calculateCommission(double amount) {

return amount \* 0.05; // 5%

}

1. **метод:** double getMaxTransactionLimit();

**параметры:** отсутствуют

**возвращаемое значение:** double – максимально возможная сумма одной транзакции

**описание предназначения метода:** метод нужен для получения лимита одной транзакции, что в дальнейшем используется для ограничения операций для разных клиентов.

**описание реализации:** метод возвращает значение, равное лимиту одной транзакции (в данном случае это 1000)

**программный код:**

double StandartClient::getMaxTransactionLimit() {

return 1000;}

1. **метод:** string getBenefits();

**параметры:** отсутствуют

**возвращаемое значение:** string – строка, описывающая выгоды данного тарифа

**описание предназначения метода:** с помощью этого метода клиент может узнать о выгодах данного тарифа

**описание реализации:** метод возвращает строку, описывающую выгоду тарифа (в данном случае это “Стандартный доступ”)

**программный код:**

string StandartClient::getBenefits() {

return “Стандартный доступ”;

}

8. КЛАСС ClientNode

- узел в двоичном дереве поиска клиентов

1. **метод:** ClientNode(Client\* data) : data(data), left(nullptr), right(nullptr)

**параметры:** Client\* data – указатель на объект класса Client

**возвращаемое значение:** отсутствует (это конструктор)

**описание предназначения метода:** Конструктор инициализирует указатель на клиента и устанавливает указатели на левых и правых потомков в nullptr.

**описание реализации:** метод принимает указатель на объект типа Client, а затем инициализирует члены этого класса. data(data) – присваиваем данные из указателя данному объекту. left(nullptr), right(nullptr) – инициализируем указатели на левого и правого потомка как nullptr, что значит, что их на момент создания нет.

**программный код:**

ClientNode(Client\* data) : data(data), left(nullptr), right(nullptr) {}

1. **метод: ~ClientNode()**

**параметры:** отсутствуют

**возвращаемое значение:** отсутствует

**описание предназначения метода:** Деструктор класса ClientNode нужен для того, чтобы освобождать память при удалении объекта

**описание реализации:** мы освобождаем память, выделенную для объекта Client, на который указывает указатель data.

**программный код:**

~ClientNode() { delete data; }

9. КЛАСС Blockchain

1. **метод:** void addClient(Client\* client);

**параметры:** Client\* client – указатель на объект класса Client

**возвращаемое значение:** отсутствует

**описание предназначения метода:** метод нужен для того, чтобы добавить клиента в дерево клиентов

**описание реализации:** вызывается метод insert из класса ClientBST, чтобы вставить клиента в дерево.

**программный код:**

void Blockchain::addClient(Client\* client) {

clients.insert(client);

}

1. **метод:** bool processTransaction(Transaction\* tx);

**параметры:** указатель на объект класса Transaction

**возвращаемое значение:** true или false, в зависимости от того, можно ли провести данную операцию

**описание предназначения метода:** метод определяет, можно ли провести операцию, опираясь на кошельки отправителя и получателя, количества средств на счету отправителя, лимит транзакции отправителя. Затем метод проводит операцию, добавляя её в список.

**описание реализации:**

Wallet\* sender = dynamic\_cast<Wallet\*>(clients.find(tx->getSenderWalletId()));

Wallet\* receiver = dynamic\_cast<Wallet\*>(clients.find(tx->getReceiveWalletId()));

* находит кошельки отправителя и получателя по их id с помощью метода find из ClientBST

if (!sender || !receiver) return false;

* проверка: существуют ли оба кошелька

Client\* senderClient = dynamic\_cast<Client\*>(clients.find(sender->getOwnerId()));

if (!senderClient) return false;

* находит клиента отправителя и проверяет его существование

double commission = senderClient->calculateCommission(tx->getAmount());

double totalAmount = tx->getAmount() + commission;

if (sender->getBalance() < totalAmount) return false;

if (tx->getAmount() > senderClient->getMaxTransactionLimit()) return false;

* проверка: достаточно ли у отправителя средств с учетом лимита транзакций

sender->withdraw(totalAmount);

receiver->deposit(tx->getAmount());

transactions.addTransaction(tx);

return true;

* проводит транзакцию, добавляя её в список

1. **метод:** void displayClients();

**параметры:** отсутствуют

**возвращаемое значение:** отсутствуют

**описание предназначения метода:** выводит список всех клиентов

**описание реализации:** применяет метод displayInOrder из класса TransactionList ко всем клиентам

**программный код:**

void Blockchain::displayClients() {

clients.displayInOrder();

}

1. **метод:** void displayTransactions();

**параметры:** отсутствуют

**возвращаемое значение:** отсутствуют

**описание предназначения метода:** выводит список всех трнзакций

**описание реализации:** применяет метод displayTransaction из класса TransactionList ко всем транзакциям

**программный код:**

void Blockchain::displayTransactions() {

transactions.displayTransactions();

}

1. КЛАСС TRANSACTION
2. **метод:** Transaction(string senderWalletId, string receiveWalletId, double amount, TxType type, double commission)

**параметры:**

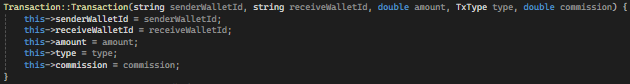
* string senderWalletId - идентификатор кошелька отправителя.
* string receiveWalletId - идентификатор кошелька получателя.
* double amount – сумма перевода.
* TxType type - тип транзакции.
* double commission – комиссия.

**возвращаемое значение**: отсутствует (конструктор)

**описание предназначения метода:** этот конструктор инициализирует объект класса Transaction, устанавливая значения полей.

**описание реализации:**  принимает входные параметры и присваивает их соответствующим полям объекта.

**программный код:**



1. **метод:** const string& getId()

**параметры:** отсутствуют

**возвращаемое значение**: const string& – константная ссылка на строку, содержащую идентификатор транзакции (id)

**описание предназначения метода:**   Метод предоставляет доступ к идентификатору транзакции (id) в режиме "только для чтения". Используется, когда нужно получить id транзакции без возможности его изменения.

**описание реализации:** Возвращает константную ссылку на поле id объекта класса Transaction. Использование const string& исключает копирование данных и гарантирует, что вызывающий код не сможет изменить id.

**программный код:**



1. **метод:** string getDetails()

**параметры:** отсутствуют

**возвращаемое значение**: string – строка, содержащая форматированное описание транзакции.

**описание предназначения метода:** геттер, выводит текстовое описание транзакции

**описание реализации:** Использует to\_string() для преобразования чисел (amount, commission) в строки.

**программный код:**

****

1. **метод:** string getSenderWalletId()

**параметры:** отсутствуют

**возвращаемое значение**: ID отправителя строкой

**описание предназначения метода:** вывод ID отправителя

**программный код:**



1. **метод:** string getReceiveWalletId()

**параметры:** отсутствуют

**возвращаемое значение**: ID получателя строкой

**описание предназначения метода:** вывод ID получателя

**программный код:**

****

1. **метод:** double getAmount()

**параметры:** отсутствуют

**возвращаемое значение:** сумма перевода

**описание предназначения метода:** вывод суммы перевода

**программный код:**

****

1. **метод:** double getCommission()

**параметры:** отсутствуют

**возвращаемое значение:** сумма комиссии

**описание предназначения метода:** вывод суммы комиссии

**программный код:**

****

1. КЛАСС TRANSACTIONLIST

**1. метод:** void addTransaction(Transaction\* tx)

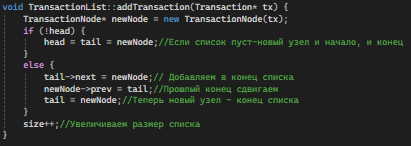
**параметры:**tx (тип Transaction\*) - указатель на добавляемую транзакцию

**возвращаемое значение**: Отсутствует (void)

**описание предназначения метода:** Метод добавляет новую транзакцию в конец связного списка

**описание реализации:** Создается новый узел списка (TransactionNode) с переданной транзакцией. Если список пуст (head == nullptr), новый узел становится единственным элементом (и головой, и хвостом). Если список не пуст, новый узел добавляется в конец.

**программный код:**

****

**2. метод:** bool removeTransaction(const string& id)

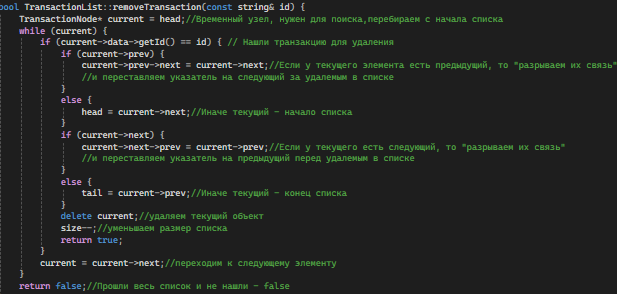
**параметры:** const string& id – константная ссылка на id

**возвращаемое значение:** bool - true если транзакция была найдена и удалена, false если не найдена

**описание предназначения метода:** Метод осуществляет поиск и удаление транзакции из связного списка по её идентификатору.

**описание реализации:** Начинаем с головы списка (head). Последовательно перебираем узлы, пока не найдем нужный ID. При нахождении: обновляем связи соседних узлов. Особые случаи: удаляемый узел - голова или хвост. Освобождаем память узла и уменьшаем счетчик элементов. Если узел не найден - возвращаем false

**программный код:**



3. **метод:** Transaction\* getTransaction(const string& id)

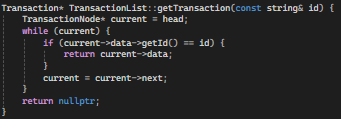
**параметры:** const string& id – константная ссылка на id типа string

**возвращаемое значение:** Transaction\* - указатель на найденную транзакцию

**описание предназначения метода:** Метод осуществляет поиск транзакции в связном списке по её уникальному идентификатору.

**описание реализации:** Метод последовательно проверяет все узлы списка, начиная с головного (head). Для каждого узла он сравнивает ID транзакции с заданным. При совпадении возвращает указатель на транзакцию, а если не находит - nullptr.

**программный код:**

****

1. **метод:** void displayTransactions()

**параметры:** нет

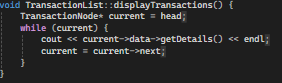
**возвращаемое значение:** нет, тип void

**описание предназначения метода:** Метод выводит на консоль подробную информацию о всех транзакциях, содержащихся в списке, в порядке их хранения (от головы к хвосту).

**описание реализации:** Начинает обход с головного элемента списка (head), переходит от узла к узлу через указатель next. Для каждого узла вызывает метод getDetails() хранимой транзакции.

Выводит полученную строку с информацией о транзакции на консоль. Процесс продолжается до достижения конца списка (пока current != nullptr)

**программный код:**

****

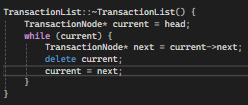
1. **метод:** ~TransactionList()

**параметры:** деструктор не принимает параметров

**возвращаемое значение:** деструктор не возвращает значений

**описание предназначения метода:** обеспечивает корректное удаление объекта класса TransactionList

**описание реализации:** Деструктор последовательно удаляет все узлы списка, начиная с головного элемента. Для каждого узла сохраняется указатель на следующий элемент, после чего текущий узел удаляется, и процесс переходит к следующему. Так продолжается, пока не будут удалены все элементы списка. Метод автоматически вызывается при уничтожении объекта, гарантируя освобождение занятой памяти.

**программный код: **

1. КЛАСС CLIENTBST
2. **метод:** void insertHelper(ClientNode\*& node, Client\* client)

**параметры:**

**-** node (тип ClientNode\*&) - ссылка на указатель текущего узла дерева

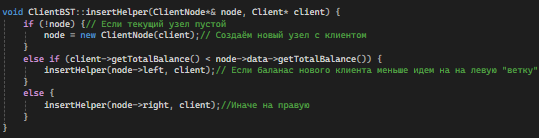
- client (тип Client\*) - указатель на добавляемого клиента

**возвращаемое значение:** Отсутствует (void)

**описание предназначения метода:** Метод рекурсивно вставляет нового клиента в бинарное дерево поиска

**описание реализации:** Если текущий узел пуст (nullptr), создается новый узел с клиентом. Если баланс добавляемого клиента меньше баланса в текущем узле: рекурсивный вызов для левого поддерева. Иначе (баланс больше или равен): рекурсивный вызов для правого поддерева. Процесс повторяется пока не найдется пустое место для вставки.

**программный код:**

****

1. **метод:** ClientNode\* findMin(ClientNode\* node)

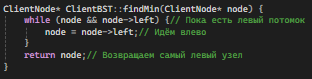
**параметры:** node (тип ClientNode\*) - указатель на корень поддерева для поиска

**возвращаемое значение:** ClientNode\* - указатель на узел с минимальным значением баланса

**описание предназначения метода:** Метод находит и возвращает клиента с минимальным общим балансом в заданном поддереве бинарного дерева поиска.

**описание реализации:** Начинает с переданного узла (node), последовательно переходит по левым потомкам (node->left), прекращает поиск при достижении узла без левого потомка и возвращает найденный узел с минимальным значением.

**программный код:**

****

1. **метод:** ClientNode\* removeHelper(ClientNode\* node, const string& id)

**параметры:**

**-** node (тип ClientNode\*) - текущий узел для обработки;

- id (тип const string&) - идентификатор клиента для удаления.

**возвращаемое значение:** ClientNode\* - указатель на модифицированное поддерево (после удаления)

**описание предназначения метода:** Рекурсивно удаляет клиента с указанным ID из BST, сохраняя свойства бинарного дерева поиска.

**описание реализации:**

**-** Базовый случай: если узел пуст, возвращает nullptr

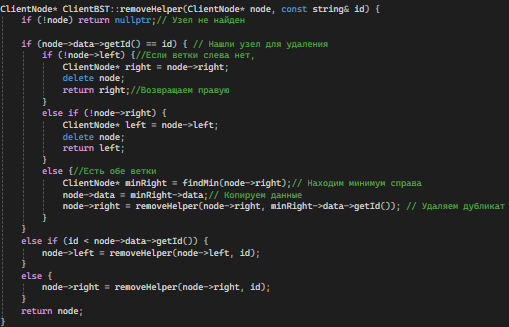
Поиск узла:

* Если ID совпадает - обрабатывает удаление
* Если ID меньше - рекурсивно ищет в левом поддереве
* Если ID больше - рекурсивно ищет в правом поддерева.

Удаление узла:

* Один потомок: заменяет удаляемый узел его потомком;
* Два потомка: находит минимальный элемент в правом поддереве, копирует его данные и рекурсивно удаляет дубликат.

Возвращает обновленное поддерево

**программный код:**

1. **метод**: Client\*findHelper(ClientNode\* node, const string& id) const

**параметры:**

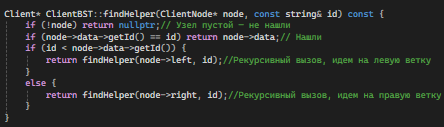
**-** указатель на текущий узел дерева node типа ClientNode\*

- константная ссылка на строку id типа const string&, содержащая идентификатор искомого клиента.

**возвращаемое значение:** возвращает указатель на найденный объект клиента Client\*

**описание предназначения метода:** для поиска клиента в бинарном дереве поиска по его уникальному идентификатору

**описание реализации:** Если идентификатор текущего узла совпадает с искомым, возвращается указатель на данные этого узла. Если искомый идентификатор меньше идентификатора текущего узла, метод рекурсивно вызывается для левого поддерева, в противном случае — для правого.

**программный код:**

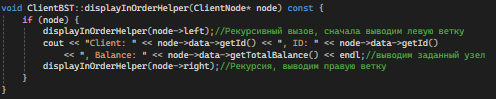
1. **метод:** void displayInOrderHelper(ClientNode\* node) const

**параметры:** указатель на текущий узел дерева node типа ClientNode\*

**возвращаемое значение:** нет, тип void

**описание предназначения метода:** для вывода информации о клиентах, хранящихся в бинарном дереве поиска, в отсортированном порядке по возрастанию.

**описание реализации:** Если переданный узел не пуст, он сначала рекурсивно вызывает себя для левого поддерева, что обеспечивает обработку всех узлов с меньшими значениями. Затем выводит информацию о текущем клиенте, включая его идентификатор и общий баланс. После этого рекурсивно обрабатывает правое поддерево, содержащее узлы с большими значениями.

**программный код:** ****

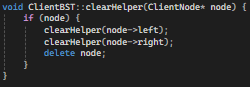
1. **метод:** void clearHelper(ClientNode\* node)

**параметры:** указатель на текущий узел дерева node типа ClientNode\*

**возвращаемое значение:** нет, тип void

**описание предназначения метода:** для рекурсивного удаления всех узлов бинарного дерева поиска, начиная с указанного узла.

**описание реализации:** Для действительного узла он сначала рекурсивно вызывает себя для левого поддерева, затем для правого поддерева, что гарантирует обработку всех дочерних узлов. После обработки обоих поддеревьев метод освобождает память текущего узла с помощью оператора delete.

**программный код:** ****

1. **метод:** void insert(Client\* client)

**параметры:** указатель на объект клиента client типа Client\*, который необходимо добавить в дерево.

**возвращаемое значение:** нет, тип void

**описание предназначения метода:** Этот метод служит публичным интерфейсом для вставки новых клиентов в бинарное дерево поиска.

**описание реализации:** работает путем вызова вспомогательной приватной функции insertHelper, передавая ей корень дерева (root) и указатель на добавляемого клиента.

**программный код:** ****

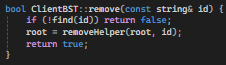
1. **метод:** bool remove(const string& id)

**параметры:** константная ссылка на строку id типа const string&, содержащая идентификатор клиента, которого необходимо удалить из дерева.

**возвращаемое значение:** bool: true если клиент с указанным идентификатором был найден и успешно удален, false если клиент с таким идентификатором не был найден в дереве.

**описание предназначения метода:** безопасный способ удаления клиента из бинарного дерева поиска, проверяя существование клиента перед удалением.

**описание реализации:** Если клиент не найден, сразу возвращается false. Если клиент существует, вызывается вспомогательный метод removeHelper, которому передается корень дерева и идентификатор для удаления. Результат работы removeHelper (модифицированное дерево) сохраняется в корне, после чего метод возвращает true, подтверждая успешное удаление

**программный код:** ****

1. **метод:** Client\* find(const string& id) const

**параметры:** константная ссылка на строку id типа const string&, содержащая id клиента, которого необходимо найти в дереве.

**возвращаемое значение:** указатель на объект клиента Client\*

**описание предназначения метода:** Этот метод предоставляет публичный интерфейс для поиска клиентов в бинарном дереве поиска по их уникальному идентификатору.

**описание реализации:** работает путем вызова приватного вспомогательного метода findHelper, передавая ему корень дерева (root) и искомый идентификатор клиента.

**программный код:** ****

1. **метод:** void displayInOrder() const

**параметры:** нет

**возвращаемое значение:** нет, тип void

**описание предназначения метода:** выводит информацию о клиентах в порядке возрастания их идентификаторов

**описание реализации:** простой публичный интерфейс, который вызывает приватный вспомогательный метод displayInOrderHelper, передавая ему корень дерева (root).

**программный код:**

****

1. **метод:** ~ClientBST()

**параметры:** нет

**возвращаемое значение:** нет

**описание предназначения метода:** освобождает всю память, выделенную под узлы бинарного дерева поиска.

**описание реализации:** вызывает вспомогательный метод clearHelper, передавая ему корень дерева (root).

**программный код:**

****

ЗАВЕРШЕНИЕ

1. Изучено

* Принципы работы блокчейн-систем и их упрощенной реализации в контексте финансовых транзакций.
* Основы объектно-ориентированного программирования (ООП) в C++, включая инкапсуляцию, наследование, полиморфизм и абстракцию.
* Структуры данных: бинарные деревья поиска (BST) для эффективного хранения и поиска клиентов, а также двунаправленные списки для хранения транзакций.
* Механизмы управления памятью в C++ (конструкторы, деструкторы, операторы new и delete).

2. Освоено

* Создание классов и их взаимодействие через ассоциацию, агрегацию и композицию.
* Реализация операций добавления, удаления и поиска данных в бинарном дереве (BST).
* Работа с двунаправленными списками для хранения и обработки транзакций.
* Формирование отчетов о клиентах и транзакциях с возможностью вывода в консоль.
* Обработка базовых финансовых операций: депозиты, снятия и переводы между кошельками.

3. Реализовано

* Класс Client – хранит информацию о клиенте (ID, баланс и список транзакций).
* Класс Transaction – описывает транзакцию (отправитель, получатель, сумма, комиссия, тип операции).
* Класс TransactionList – двунаправленный список для хранения истории транзакций.
* Класс ClientBST – бинарное дерево поиска для эффективного хранения и поиска клиентов.
* Блокчейн-система – управление транзакциями, проверка балансов, генерация отчетов.
* Методы отчетности – вывод списка клиентов и транзакций в удобном формате.

ИТОГ

Разработана упрощенная, но функциональная система, имитирующая блокчейн-транзакции с использованием ООП и эффективных структур данных. Проект демонстрирует понимание ключевых концепций C++ и их применение в реальных сценариях.