ВВЕДЕНИЕ

Объект исследования: технология создания программного продукта с помощью современных средств.

Предмет исследования: объектно-ориентированное приложение, с помощью которого можно производить управление счетами клиентами.

Методы исследования: в данной курсовой работе использовались методы технологии объектно-ориентированного программирования в интегрированной среде разработки Microsoft Visual Studio 2017 на языке программирования C++

Цель курсовой работы: разработка объектно-ориентированного приложения в интегрированной среде разработки Microsoft Visual Studio 2017 для управления счетами клиентами в банке.

Задачи курсовой работы:

* Разработка архитектуры приложения
* Разработка основного класса «Счёт»
* Разработка классов, реализующих взаимодействие с классом «Счёт»
* Разработка графического пользовательского интерфейса, основанного на стандартном консольном окне Windows.

Результаты курсовой работы: создана архитектура приложения с использованием современных инструментов и паттернов проектирования и реализована программная часть продукта.

АКТУАЛЬНОСТЬ ВЫБРАННОЙ ТЕМЫ

В настоящее время всё большую популярность приобретают онлайн-транзакции и оплата банковскими картами, таким образом качественные программы для управления счетами клиентами в банке с каждым днём становятся всё востребованнее. Сама идея создания правильно функционирующей системы не является сама по себе новой, однако примеров успешных реализаций довольно мало.

КОНКРЕТИЗАЦИЯ ЗАДАЧИ

Необходимо разработать программу, которая облегчила процесс управления собственными банковскими счетами. Она должна поддерживать различные виды счетов (расчётный, депозитный, кредитный), а также отображать счета в зависимости от доходности и количества операций по счёту.

ГРАНИЦА ПРИМЕНИМОСТИ ПРОГРАММЫ

Данное программное обеспечение имеет возможность применения банками и другими организациями, предоставляющие банковские услуги.

1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

В данной курсовой работе представлена программа, позволяющая управлять различными счетами (расчётный, депозитный, кредитный) клиентами в банке. Для её успешной реализации необходимо спроектировать такие её функциональные части, как открытие и закрытие новых счетов, перевод денег между счетами, вывод всех отсортированных по доходности или по количеству проведенных операций счетов на экран и базовый пользовательский интерфейс.

1. МЕТОДОЛОГИЯ РАЗРАБОТКИ ПРИЛОЖЕНИЯ

# 2.1 Выбор языка программирования для разработки приложения

В качестве основного языка для разработки данной программы является статически типизированный язык программирования C++, т.к. в нём поддерживается ООП, и обучение в этом семестре ведётся на примере именно этого языка.

# 2.2 Выбор инструментов для разработки графического интерфейса приложения

# 2.3 Выбор инструментов для разработки функциональной системы приложения.

2.3.1 QtCreator IDE

10

QtCreator IDE – кросс-платформенная интегрированная среда для

разработки приложений, использующих компоненты Qt. Включает в себя

удобный отладчик, визуальные средства для разработки интерфейса и

встроенный дизайнер и редактор интерфейсов.

2.3.2 GitHub

GitHub – бесплатный интернет-ресурс для разработки и хостинга ITпроектов, с встроенной системой контроля версий приложений.

GitHub был выбран как сервис по поддержанию контроля версий

приложения.

2.4 Разработка структуры приложения

Программа содержит следующие функции:

• Регистрация и удаление персональной адресной книги

• Добавление нового контакта

• Редактирование или добавление информации контакта

• Удаление контакта

• Сортировка списка контактов в алфавитном порядке

1. ПРОЕКТНАЯ ЧАСТЬ

# Описание функциональных требований к разрабатываемой программной системе.

Необходимо разработать программу, которая может быть использована для поддержания сведений об использовании клиентского счета в банке в актуальном состоянии. Для реализации необходимо:

* Разработать класс «User», который будет представлять клиента системы, с помощью которого можно будет проводить различные манипуляции с счетами в банке.
* Разработать абстрактный класс «Account», который будет представлять счёт в банке, и дочерние классы «TermAccount», «CreditAccount», «DepositAccount» и « CheckingAccount».
* Разработать класс «GUI», который будет отвечать за логику графического пользовательского интерфейса и передачу управления диалоговым окнам.
* Разработать абстрактный класс «Dialog», который отвечает за вывод информации на экран и взаимодействие с пользователем, и дочерние классы «MainDialog», «AddAcountDialog», «AccountDeletionDialog» и другие.

# Описание разрабатываемой программной системы с точки зрения программиста.

Объектное представление системы: Клиент системы хранит данные о собственных счетах в банке, а также может их открывать и закрывать и переводить деньги между собственными счетами. Таким образом можно выделить два основополагающих объекта: «Пользователь» и «Счёт».

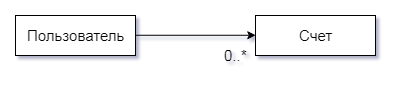
****

Рисунок 1. Диаграмма основных объектов

# Проектное решение и описание классов

Описание интерфейса «IAccountsOwner». Данный интерфейс определяет взаимодействие с объектом, который может выступать в роли хранилища счетов. Он обладает следующими публичными (public) методами:

* virtual void AddAccount(Account\*) = 0 — метод добавления нового счёта в контейнер.
* virtual void DeleteAccount(Account\*) = 0 — метод удаления счёта из контейнера.
* virtual std::set<Account\*>& GetAccounts() = 0 — метод, возвращающий набор счетов.

Описание класса «User». Данный класс представляет хранилище счетов клиента. Он реализует интерфейс «IAccountsOwner» для взаимодействия и обладает следующим собственным полем:

* std::set<Account\*> accounts — поле, которое хранит все счета клиента.
* void AddAccount(Account\*) override — переопределенный метод добавления нового счёта в контейнер.
* void DeleteAccount(Account\*) override — переопределенный метод удаления счёта из контейнера.
* std::set<Account\*>& GetAccounts() override — переопределенный метод, возвращающий набор счетов.

Описание абстрактного класса «Account». Данный класс является базовым для всех будущих счетов. Он реализует основной функционал для манипуляции со счетом и обладает следующими полями и методами:

* std::string number — переменная, хранящая номер счёта. Спецификатор доступа: protected.
* double money — переменная, хранящая информацию о количестве денег на счёте. Спецификатор доступа: protected.
* bool isClosed — переменная, хранящая информацию о состоянии счета (открыт или закрыт). Спецификатор доступа: protected.
* int numberOfOperations — переменная, хранящая информацию о количестве успешных операций. Спецификатор доступа: private.
* void PrintData() const — метод, выводящий полную информацию о счёте на экран. Спецификатор доступа: public.
* void PrintShortData() const — метод, выводящий краткую информацию о счёте на экран. Спецификатор доступа: public.
* virtual std::string GetData() const — метод, возвращающий полную информацию о счёте. Спецификатор доступа: public.
* virtual std::string GetShortData() const — метод, возвращающий краткую информацию о счёте. Спецификатор доступа: public.
* void AddMoney(const double& TRANSFERRED\_MONEY) — метод для добавления переведенных денег с другого счета на данный счет. Спецификатор доступа: public.
* void Close() — метод закрытия счёта. Спецификатор доступа: public.
* bool TransferMoney(Account\* receiver, const double& AMOUNT\_OF\_MONEY) — метод перевода денег на другой счет. Спецификатор доступа: public.
* bool IsClosed() const — метод, возвращающий текущее состояние счета. Спецификатор доступа: public.
* double GetMoney() const — метод, возвращающий количество денег на счете. Спецификатор доступа: public.
* int GetNumberOfOperations() const — метод, возвращающий количество операций выполненных над счётом. Спецификатор доступа: public.
* bool IsAvailableAmountOfMoney(const double& AMOUNT\_OF\_MONEY) — метод проверки доступного количества денег на счёте. Спецификатор доступа: private.
* void DecreaseMoney(const double& MONEY\_DECREMENT) — метод списания денег со счёта. Спецификатор доступа: private.

Описание абстрактного класса «TermAccount». Данный класс является дочерним к классу «Account» и базовым для всех будущих срочных счетов (например, кредитный или депозитный). Он реализует дополнительный функционал для манипуляции со срочным счетом и обладает следующими полями и методами:

* float interestRate — переменная, хранящая процентную ставку по счету. Спецификатор доступа: protected.
* short int term — переменная, хранящая срок счёта. Спецификатор доступа: protected.
* double termMoney — переменная, хранящая сумму (депозита или кредита) счёта. Спецификатор доступа: protected.
* TermAccount(const double& MONEY, const float& INTEREST\_RATE, const short int& TERM, const double& TERM\_MONEY) — конструктор срочного счета. Спецификатор доступа: public.
* virtual std::string GetData() const — переопределенный метод, возвращающий полную информацию о счёте. Спецификатор доступа: public.
* virtual std::string GetShortData() const — переопределенный метод, возвращающий краткую информацию о счёте. Спецификатор доступа: public.

Описание классов «CreditAccount» и «DepositAccount». Данные классы является дочерними класса «TermAccount». Они реализуют разное поведение, однако переопределяют одни и те же методы базового класса, а именно:

* std::string GetData() const override— переопределенный метод, возвращающий полную информацию о счёте. Спецификатор доступа: public.
* std::string GetShortData() const override— переопределенный метод, возвращающий краткую информацию о счёте. Спецификатор доступа: public.

Так же, как и методы, их конструкторы одинаковы:

* CreditAccount (const double& MONEY, const float& INTEREST\_RATE, const short int& TERM, const double& TERM\_MONEY) — конструктор кредитного счета. Спецификатор доступа: public.
* DepositAccount (const double& MONEY, const float& INTEREST\_RATE, const short int& TERM, const double& TERM\_MONEY) — конструктор депозитного счета. Спецификатор доступа: public.

Описание класса «CheckingAccount». Данный класс является дочерним к классу «Account». Он реализует поведение расчетного счета и переопределяет следующие методы базового класса:

* std::string GetData() const override — метод, возвращающий полную информацию о счёте. Спецификатор доступа: public.
* std::string GetShortData() const override — метод, возвращающий краткую информацию о счёте. Спецификатор доступа: public.

Описание интерфейса «IDialogManager». Данный интерфейс определяет взаимодействие с менеджером диалогов. Он обладает следующими публичными (public) методами:

* virtual void ShowDialog(class Dialog\*) = 0 — метод вывода нового диалога на экран.
* virtual void CloseDialog() = 0 — метод закрытия текущего диалога.
* virtual class IAccountsOwner\* GetAccountOwner() = 0 — метод, возвращающий владельца счетов (текущего клиента).

Описание класса «GUI». Данный класс отвечает за графический пользовательский интерфейс. Он реализует интерфейс «IDialogManager» для взаимодействия и обладает следующими полями и методами:

* GUI(IAccountsOwner\* owner) — конструктор класса GUI. Спецификатор доступа: public.
* IAccountsOwner\* accountsOwner — переменная, хранящая указатель на клиента. Спецификатор доступа: private.
* std::stack<Dialog\*> dialogStack — переменная, хранящая стек диалогов. Спецификатор доступа:private.
* void ShowDialog(Dialog\*) override — переопределенный метод вывода нового диалога на экран. Спецификатор доступа: public.
* void CloseDialog() override — переопределенный метод закрытия текущего диалога. Спецификатор доступа: public.
* IAccountsOwner\* GetAccountOwner() override — переопределенный метод, возвращающий владельца счетов (текущего клиента). Спецификатор доступа: public.

Описание абстрактного класса «Dialog». Данный класс является базовым для всех будущих диалогов. Он реализует основной функционал для манипуляции с диалогом и обладает следующими полями и методами:

* Dialog(IDialogManager\* dialogManager) — конструктор диалога. Спецификатор доступа: public.
* bool isClose — переменная, хранящая текущее состояние диалога. Спецификатор доступа: protected.
* int backLine — переменная, хранящая номер строки выхода из диалога. Спецификатор доступа: protected.
* std::vector<std::string> data — переменная, хранящая данные для вывода на экран. Спецификатор доступа: protected.
* IDialogManager\* dialogManager — переменная, хранящая указатель на менеджер диалогов. Спецификатор доступа: protected.
* void Show() — метод, выводящий диалог на экран. Спецификатор доступа: public.
* void Close() — метод закрытия диалога. Спецификатор доступа: public.
* void SetColor(const Color text, const Color background) const — метод, устанавливающий цвет текста и заднего фона в консоли. Спецификатор доступа: protected.
* void SetSelectorColor() const — метод, устанавливающий цвет селектора в консоли. Спецификатор доступа: protected.
* void SetDefaultColor() const — метод, устанавливающий стандартный цвет текста и заднего фона в консоли. Спецификатор доступа: protected.
* virtual void HandleInput(int& currentLine, const char& INPUT) — метод, отвечающий за ввод в консоли. Спецификатор доступа: protected.
* virtual void PrintData(const int& CURRENT\_LINE) const — метод, выводящий данные на экран. Спецификатор доступа: protected.

Описание класса «MenuDialog». Данный класс является дочерним классом «Dialog» и реализует логику меню приложения. Он переопределяет лишь один метод:

* void HandleInput(int& currentLine, const char& INPUT) override — переопределенный метод, отвечающий за ввод в консоли. Спецификатор доступа: private.

Описание класса «AddAccountDialog». Данный класс является дочерним классом «Dialog» и реализует логику открытия нового счета. Он обладает следующими методами:

* void HandleInput(int& currentLine, const char& INPUT) override — переопределенный метод, отвечающий за ввод в консоли. Спецификатор доступа: private.
* void ClearWindow() const — метод, отвечающий за очистку окна консоли. Спецификатор доступа: private.
* void PassNewAccount(Account\* newAccount) const — метод, отвечающий за передачу нового аккаунта клиенту. Спецификатор доступа: private.

Также в нём определено перечисление «ELine», которое хранит номера строк в консоли. Оно обладает следующими элементами:

* CreditAccount
* DepositAccount
* CheckingAccount
* Back

Описание класса «AccountDeletionDialog». Данный класс является дочерним классом «Dialog» и реализует логику закрытия определенного счета. Он обладает следующими полями и методами:

* std::set<Account\*> userAccounts — переменная, хранящая указатели на счета клиента. Спецификатор доступа: private.
* int backLine — переменная, хранящая номер строки выхода из диалога. Спецификатор доступа: private.
* void HandleInput(int& currentLine, const char& INPUT) override — переопределенный метод, отвечающий за ввод в консоли. Спецификатор доступа: private.
* void SetBackLineId() —метод, устанавливающий новый номер строки выхода из диалога. Спецификатор доступа: private.

Описание класса «TransferMoneyDialog». Данный класс является дочерним классом «Dialog» и реализует логику перевода денег между счетами. Он обладает следующими полями и методами:

* std::set<Account\*> userAccounts — переменная, хранящая указатели на счета клиента. Спецификатор доступа: private.
* Account\* sender — переменная, хранящая указатель на отправителя. Спецификатор доступа: private.
* Account\* receiver — переменная, хранящая указатель на получателя. Спецификатор доступа: private.
* int backLine — переменная, хранящая номер строки выхода из диалога. Спецификатор доступа: private.
* void HandleInput(int& currentLine, const char& INPUT) override — переопределенный метод, отвечающий за ввод в консоли. Спецификатор доступа: private.
* void UpdateData() —метод, отвечающий за обновление данных в классе. Спецификатор доступа: private.
* Account\* GetAccountById(const int& id) — метод, возвращающий счёт в наборе по индексу. Спецификатор доступа: private.
* bool CanSendMoney() — метод, проверяющий возможность перевода денег. Спецификатор доступа: private.
* void SendMoney() — метод, отвечающий за отправку денег. Спецификатор доступа: private.

Описание класса «AccountsListDialog». Данный класс является дочерним классом «Dialog» и выводит все счета клиента на экран. Он переопределяет следующий метод:

* void HandleInput(int& currentLine, const char& INPUT) override — переопределенный метод, отвечающий за ввод в консоли. Спецификатор доступа: private.

Описание класса «SortingByOperationsDialog». Данный класс является дочерним классом «Dialog» и выводит все счета клиента, отсортированные по количеству операций, на экран. Он переопределяет следующий метод:

* void HandleInput(int& currentLine, const char& INPUT) override — переопределенный метод, отвечающий за ввод в консоли. Спецификатор доступа: private.

Описание класса «SortingByMoneyDialog». Данный класс является дочерним классом «Dialog» и выводит все счета клиента, отсортированные по количеству денег, на экран. Он переопределяет следующий метод:

* void HandleInput(int& currentLine, const char& INPUT) override — переопределенный метод, отвечающий за ввод в консоли. Спецификатор доступа: private.

Описание класса «AccountFactory». Данный класс отвечает за открытие новых счетов. Он обладает следующими методами:

* static Account\* MakeCreditAccount() — метод, отвечающий за открытие кредитного счёта. Спецификатор доступа: public.
* static Account\* MakeDepositAccount() — метод, отвечающий за открытие депозитного счёта. Спецификатор доступа: public.
* static Account\* MakeCheckingAccount() — метод, отвечающий за открытие расчетного счёта. Спецификатор доступа: public.
* template<typename T> static void Input(T& destination) — шаблонный метод, отвечающий за ввод данных. Спецификатор доступа: private.

Описание класса «Application». Данный класс является точкой входа в программу. Он обладает следующим методом:

* void Start() — метод, отвечающий за старт программы. Спецификатор доступа: public.

Описание перечисления «EColor». Данное перечисление хранит базовые консольные цвета. Оно обладает следующими элементами:

* Black = 0
* Blue = 1
* Green = 2
* Cyan = 3
* Red = 4
* Magenta = 5
* Brown = 6
* LightGray = 7
* DarkGray = 8
* LightBlue = 9
* LightGreen = 10
* LightCyan = 11
* LightRed = 12
* LightMagenta = 13
* Yellow = 14
* White = 15

Описание перечисления «EKeys». Данное перечисление хранит коды клавиш клавиатуры. Оно обладает следующими элементами:

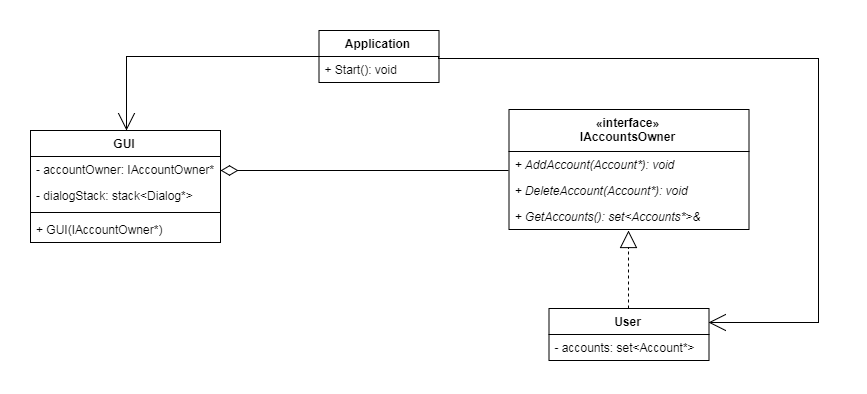
* Enter = 13
* Up = 72
* Down = 80

1. ПРИЛОЖЕНИЕ 1. ДИАГРАММА КЛАССОВ

# Диаграмма взаимодействия пользователя с GUI

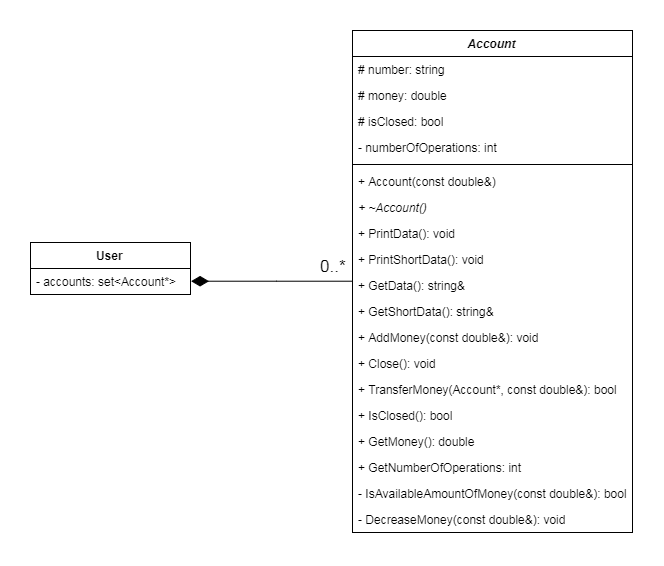
Данная диаграмма иллюстрирует взаимодействие пользователя с графическим пользовательским интерфейсом. Класс «User» представляет пользователя (клиента банка), которой реализует интерфейс «IAccountOwner», предоставляющий методы для взаимодействия с клиентом другими классами (в данном случае для класса «GUI»). Таким образом поддерживается гибкость системы.

Класс «Application» отвечает за старт программы. В методе Start() он создаёт объекты классов «User» и «GUI», в который передает созданный объект класса «User» в качестве аргумента. Далее управление программой передаётся в объект класса «GUI».



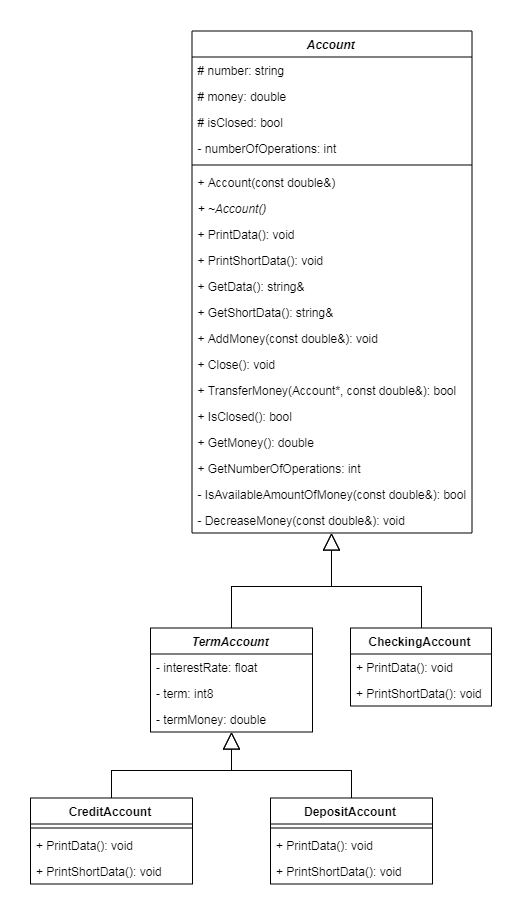
# Диаграмма связи пользователя с банковскими счетами

Данная диаграмма иллюстрирует связь пользователя с банковскими счетами. Класс «User» предоставляет хранилище для всех счетов. В то же время пользователь не знает конкретный тип счёта для поддержания гибкости системы. Также класс «User» не взаимодействует напрямую со своими счетами, т.к. жизненным циклом программы руководит класс «GUI».



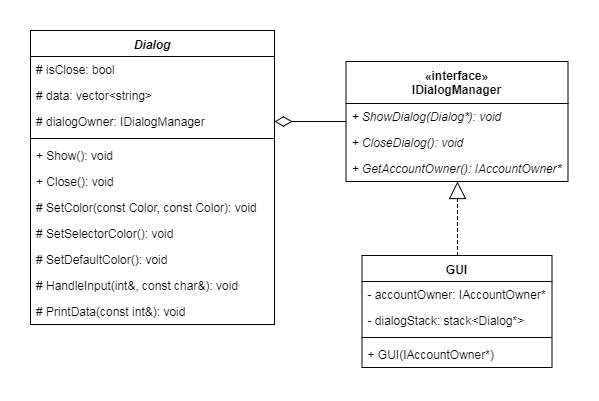
# Диаграмма наследования счетов

Данная диаграмма иллюстрирует наследование счетов. Абстрактный класс «Account» является базовым для всех счетов в программе и предоставляет базовый интерфейс для взаимодействия. Класс «CheckingAccount» представляет расчётный счёт, класс «TermAccount» так же является абстрактным и базовым для срочных счетов, такие как депозитный и кредитный. Класс «CreditAccount» представляет кредитный счёт, а «DepositAccount» — депозитный. Все они переопределяют базовые методы PrintData() и PrintShortData() базового класса «Account» для корректного вывода их данных в консоль.



# Диаграмма взаимодействия диалогов и GUI

Данная диаграмма иллюстрирует взаимодействие диалогов с графическим пользовательским интерфейсом. Класс «GUI» реализует интерфейс «IDialogManager», который предоставляет логику управления диалоговыми окнами. Здесь применен паттерн проектирования «State», а именно один из видов finite state machine: Pushdown Automata. Реализация довольно проста: при создании диалогового окна, вызывается метод ShowDialog, который помещает наш диалог в стек. Таким образом, текущее отображаемое диалоговое окно находится всегда на вершине стека. При открытии нового диалога, старое диалоговое остаётся в стеке, а новое помещается на вершину. При закрытии диалогового окна вызывается метод CloseDialog, который удаляет текущий диалог из стека (он всегда находится на его вершине) и показывает предыдущий, тем самым у нас так же сохраняется состояние всех предыдущих диалоговых окон. Также здесь применен паттерн проектирования «Mediator». Класс «GUI» является посредником между пользователем (класс «User») и диалогами.

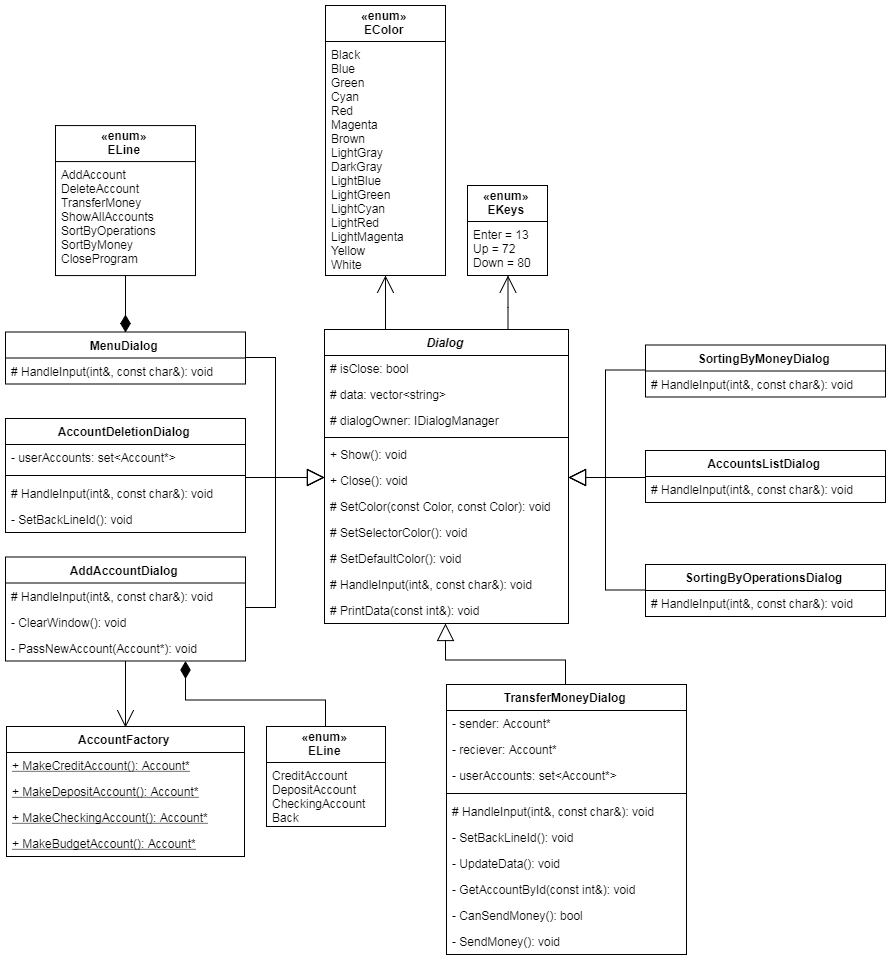


# Диаграмма наследования диалогов

Данная диаграмма иллюстрирует наследование диалогов в приложении. Абстрактный класс «Dialog» является базовым для всех диалогов в программе и предоставляет базовый интерфейс взаимодействия с ними. В файле класса «Dialog» определены перечисления:

* «EColor» — хранит базовые цвета для настройки цвета текста в консоли
* «EKeys» — хранит клавиши взаимодействия с пользовательским интерфейсом

Класс «MenuDialog» представляет главное меню приложения, которое отображается в начале программы. Класс «AccountDeletionDialog» выводит все счета счета, доступные для закрытия. Класс «AddAccountDialog» выводит список доступных счетов для открытия (в данной программе предусмотрено три типа счетов: расчётный, кредитный и депозитный). Класс «TransferMoneyDialog» представляет меню для перевода денег между своими счетами. Класс «AccountsListDialog» выводит все открытые банковские счета клиента. Класс «SortingByOperationDialog» выводит все открытые банковские счета клиента, отсортированные по количеству операций, на экран. Класс «SortingByMoneyDialog» выводит все открытые банковские счета клиента, отсортированные по количеству денег на счёте, на экран. В классах «MenuDialog» и «AddAccountDialog» определены перечисления «ELine», которые хранят номер строки в консоли, на которую назначено определенное действие. Здесь частично применен паттерн проектирования «State» hierarchical state machine. Каждый диалог переопределяет метод базового класса HandleInput таким образом, что если диалог не знает как обработать действие пользователя, он просто передаёт информацию о действии вверх по иерархии. Также на данной диаграмме можно увидеть класс «AccountFactory», использующийся классом «AddAccountDialog» для открытия новых счетов, который отчасти реализует паттерн «AbstractFactory».



1. Приложение 2. Листинги программы

# Код класса Application

Файл Application.h

#pragma once

class Application

{

public:

void Start();

};

Файл Application.cpp

#include "Application.h"

#include <windows.h>

#include "User.h"

#include "GUI.h"

int main()

{

Application\* application = new Application();

application->Start();

return 0;

}

void Application::Start()

{

User\* user = new User();

GUI\* gui = new GUI(user);

}

# Код интерфейса IAccountsOwner

#pragma once

#include "Account.h"

#include <set>

class IAccountsOwner

{

public:

virtual void AddAccount(Account\*) = 0;

virtual void DeleteAccount(Account\*) = 0;

virtual std::set<Account\*>& GetAccounts() = 0;

};

# Код класса User

Файл User.h

#pragma once

#include "IAccountsOwner.h"

#include "Account.h"

#include <set>

class User : public IAccountsOwner

{

public:

void AddAccount(Account\*) override;

void DeleteAccount(Account\*) override;

std::set<Account\*>& GetAccounts() override;

private:

std::set<Account\*> accounts;

};

Файл User.cpp

#include "User.h"

void User::AddAccount(Account\* newAccount)

{

accounts.insert(newAccount);

}

void User::DeleteAccount(Account\* account)

{

accounts.erase(account);

delete account;

}

std::set<Account\*>& User::GetAccounts()

{

return accounts;

}

# Код класса Account

Файл Account.h

#pragma once

#include <string>

class Account

{

public:

Account(const double& MONEY);

virtual ~Account() = 0;

void PrintData() const;

void PrintShortData() const;

virtual std::string GetData() const;

virtual std::string GetShortData() const;

void AddMoney(const double& TRANSFERRED\_MONEY);

void Close();

bool TransferMoney(Account\* receiver, const double& AMOUNT\_OF\_MONEY);

bool IsClosed() const;

double GetMoney() const;

int GetNumberOfOperations() const;

protected:

std::string number;

double money;

bool isClosed;

private:

int numberOfOperations;

bool IsAvailableAmountOfMoney(const double& AMOUNT\_OF\_MONEY);

void DecreaseMoney(const double& MONEY\_DECREMENT);

};

Файл Account.cpp

#include "Account.h"

#include <iostream>

#include <sstream>

#include <cstdlib>

#include <ctime>

#include <iomanip>

#define SUCCESS true

#define FAILURE false

#define AVAILABLE true

#define NOT\_AVAILABLE false

Account::Account(const double& MONEY) : money(MONEY), isClosed(false)

{

srand(time(NULL));

for (int i = 0; i < 4; i++)

{

int randNumber = rand() % 10000 + 20000;

number += std::to\_string(randNumber);

}

numberOfOperations = 0;

}

Account::~Account() {}

void Account::PrintData() const

{

std::cout << GetData();

}

std::string Account::GetData() const

{

std::ostringstream outputStream;

outputStream << "Номер счёта: " << number << "\n Баланс: " << std::fixed << std::setprecision(2) << money << std::endl;

return outputStream.str();

}

void Account::PrintShortData() const

{

std::cout << GetShortData();

}

std::string Account::GetShortData() const

{

std::ostringstream outputStream;

outputStream << "№ " << number;

return outputStream.str();

}

bool Account::TransferMoney(Account\* receiver, const double& AMOUNT\_OF\_MONEY)

{

if (IsAvailableAmountOfMoney(AMOUNT\_OF\_MONEY) && receiver != nullptr && receiver != this)

{

receiver->AddMoney(AMOUNT\_OF\_MONEY);

DecreaseMoney(AMOUNT\_OF\_MONEY);

numberOfOperations++;

return SUCCESS;

}

else

{

return FAILURE;

}

}

bool Account::IsAvailableAmountOfMoney(const double& AMOUNT\_OF\_MONEY)

{

return AMOUNT\_OF\_MONEY < money;

}

void Account::AddMoney(const double& TRANSFERRED\_MONEY)

{

money += TRANSFERRED\_MONEY;

numberOfOperations++;

}

void Account::DecreaseMoney(const double& MONEY\_DECREMENT)

{

money -= MONEY\_DECREMENT;

}

void Account::Close()

{

isClosed = true;

}

bool Account::IsClosed() const

{

return isClosed;

}

double Account::GetMoney() const

{

return money;

}

int Account::GetNumberOfOperations() const

{

return numberOfOperations;

}

# Код класса TermAccount

Файл TermAccount.h

#pragma once

#include "Account.h"

#include "basetsd.h"

class TermAccount : public Account

{

public:

TermAccount(const double& MONEY, const float& INTEREST\_RATE, const short int& TERM, const double& TERM\_MONEY);

virtual ~TermAccount() = 0;

virtual std::string GetData() const;

virtual std::string GetShortData() const;

protected:

float interestRate;

short int term;

double termMoney;

};

Файл TermAccount.cpp

#include "TermAccount.h"

#include <sstream>

TermAccount::TermAccount(const double& MONEY, const float& INTEREST\_RATE, const short int& TERM, const double& TERM\_MONEY) :

Account(MONEY), interestRate(INTEREST\_RATE), term(TERM), termMoney(TERM\_MONEY) {}

TermAccount::~TermAccount() {}

std::string TermAccount::GetData() const

{

std::ostringstream outputStream;

outputStream << "Номер счёта: " << number <<

"\n Процентная ставка: " << interestRate <<

"\n Срок: " << term << std::endl;

return outputStream.str();

}

std::string TermAccount::GetShortData() const

{

std::ostringstream outputStream;

outputStream << Account::GetShortData() << " Cрок: " << term << " " << interestRate << "%";

return outputStream.str();

}

# Код класса CreditAccount

Файл CreditAccount.h

#pragma once

#include "TermAccount.h"

class CreditAccount : public TermAccount

{

public:

CreditAccount(const double& MONEY, const float& INTEREST\_RATE, const short int& TERM, const double& TERM\_MONEY);

std::string GetData() const;

std::string GetShortData() const;

};

Файл CreditAccount.cpp

#include "CreditAccount.h"

#include <sstream>

CreditAccount::CreditAccount(const double& MONEY, const float& INTEREST\_RATE, const short int& TERM, const double& TERM\_MONEY) :

TermAccount(MONEY, INTEREST\_RATE, TERM, TERM\_MONEY) {}

std::string CreditAccount::GetData() const

{

std::ostringstream outputStream;

outputStream << TermAccount::GetData() << " Тип: Кредитный" << "\n Долг: " << termMoney - money << std::endl;

return outputStream.str();

}

std::string CreditAccount::GetShortData() const

{

return TermAccount::GetShortData() + " Тип: Кредитный\n";

}

# Код класса DepositAccount

Файл DepositAccount.h

#pragma once

#include "TermAccount.h"

class DepositAccount : public TermAccount

{

public:

DepositAccount(const double& MONEY, const float& INTEREST\_RATE, const short int& TERM, const double& TERM\_MONEY);

std::string GetData() const;

std::string GetShortData() const;

};

Файл DepositAccount.cpp

#include "DepositAccount.h"

#include <sstream>

DepositAccount::DepositAccount(const double& MONEY, const float& INTEREST\_RATE, const short int& TERM, const double& TERM\_MONEY) :

TermAccount(MONEY, INTEREST\_RATE, TERM, TERM\_MONEY) {}

std::string DepositAccount::GetData() const

{

std::ostringstream outputStream;

outputStream << TermAccount::GetData() << " Тип: Депозитный" << "\n Начисленные проценты: " << termMoney << std::endl;

return outputStream.str();

}

std::string DepositAccount::GetShortData() const

{

return TermAccount::GetShortData() + " Тип: Депозитный\n";

}

# Код класса CheckingAccount

Файл CheckingAccount.h

#pragma once

#include "Account.h"

class CheckingAccount : public Account

{

public:

CheckingAccount(const double& MONEY);

std::string GetData() const override;

std::string GetShortData() const override;

};

Файл CheckingAccount.cpp

#include "CheckingAccount.h"

#include <sstream>

#include <iomanip>

CheckingAccount::CheckingAccount(const double& MONEY) : Account(MONEY) {}

std::string CheckingAccount::GetData() const

{

return Account::GetData() + " Тип: Расчётный\n";

}

std::string CheckingAccount::GetShortData() const

{

std::ostringstream outputStream;

outputStream << Account::GetShortData() << " " << std::fixed << std::setprecision(2) << money << " " << GetNumberOfOperations() << " Расчётный\n";

return outputStream.str();

}

# Код интерфейса IAccountsOwner

#pragma once

class IDialogManager

{

public:

virtual void ShowDialog(class Dialog\*) = 0;

virtual void CloseDialog() = 0;

virtual class IAccountsOwner\* GetAccountOwner() = 0;

};

# Код класса GUI

Файл GUI.h

#pragma once

#include "IAccountsOwner.h"

#include "Dialog.h"

#include "IDialogManager.h"

#include <stack>

class GUI : public IDialogManager

{

public:

GUI(IAccountsOwner\* owner);

void ShowDialog(Dialog\*) override;

void CloseDialog() override;

IAccountsOwner\* GetAccountOwner() override;

private:

IAccountsOwner\* accountsOwner;

std::stack<Dialog\*> dialogStack;

};

Файл GUI.cpp

#include "GUI.h"

#include "MenuDialog.h"

GUI::GUI(IAccountsOwner\* owner) : accountsOwner(owner)

{

ShowDialog(new MenuDialog(this));

}

void GUI::ShowDialog(Dialog\* dialog)

{

dialogStack.push(dialog);

dialogStack.top()->Show();

}

void GUI::CloseDialog()

{

if (dialogStack.size() > 1)

{

dialogStack.top()->Close();

dialogStack.pop();

dialogStack.top()->Show();

}

}

IAccountsOwner\* GUI::GetAccountOwner()

{

return accountsOwner;

}

# Код класса Dialog

Файл Dialog.h

#pragma once

#include "IDialogManager.h"

#include <string>

#include <vector>

enum Color

{

Black = 0,

Blue = 1,

Green = 2,

Cyan = 3,

Red = 4,

Magenta = 5,

Brown = 6,

LightGray = 7,

DarkGray = 8,

LightBlue = 9,

LightGreen = 10,

LightCyan = 11,

LightRed = 12,

LightMagenta = 13,

Yellow = 14,

White = 15

};

enum Keys

{

Enter = 13,

Up = 72,

Down = 80

};

class Dialog

{

public:

Dialog(IDialogManager\* dialogManager);

void Show();

void Close();

protected:

bool isClose;

int backLine;

std::vector<std::string> data;

IDialogManager\* dialogManager;

void SetColor(const Color text, const Color background) const;

void SetSelectorColor() const;

void SetDefaultColor() const;

virtual void HandleInput(int& currentLine, const char& INPUT);

virtual void PrintData(const int& CURRENT\_LINE) const;

};

Файл Dialog.cpp

#include "Dialog.h"

#include <iostream>

#include <Windows.h>

#include <conio.h>

Dialog::Dialog(IDialogManager\* dialogManager) : dialogManager(dialogManager), isClose(false) {}

void Dialog::Close()

{

isClose = true;

}

void Dialog::SetColor(const Color text, const Color background) const

{

HANDLE hStdOut = GetStdHandle(STD\_OUTPUT\_HANDLE);

SetConsoleTextAttribute(hStdOut, (WORD)((background << 4) | text));

}

void Dialog::SetSelectorColor() const

{

SetColor(Color::LightGreen, Color::Black);

}

void Dialog::SetDefaultColor() const

{

SetColor(Color::White, Color::Black);

}

void Dialog::Show()

{

system("cls");

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

int currentLine = 0;

while (!isClose)

{

PrintData(currentLine);

const char INPUT = \_getch();

HandleInput(currentLine, INPUT);

system("cls");

}

}

void Dialog::PrintData(const int& CURRENT\_LINE) const

{

for (unsigned int id = 0; id < data.size(); id++)

{

if (CURRENT\_LINE == id)

{

SetSelectorColor();

}

std::cout << data[id];

SetDefaultColor();

}

}

void Dialog::HandleInput(int& currentLine, const char& INPUT)

{

if (INPUT == Keys::Up && currentLine > 0)

{

currentLine--;

} else if (INPUT == Keys::Down && currentLine < (data.size() - 1))

{

currentLine++;

}

}

# Код класса MenuDialog

Файл MenuDialog.h

#pragma once

#include "Dialog.h"

class MenuDialog : public Dialog

{

public:

MenuDialog(IDialogManager\* dialogManager);

private:

enum ELine

{

AddAccount,

DeleteAccount,

TransferMoney,

ShowAllAccounts,

SortByOperations,

SortByMoney,

CloseProgram

};

void HandleInput(int& currentLine, const char& INPUT) override;

};

Файл MenuDialog.cpp

#include "MenuDialog.h"

#include <conio.h>

#include <Windows.h>

#include "AddAccountDialog.h"

#include "AccountDeletionDialog.h"

#include "AccountsListDialog.h"

#include "SortingByOperationsDialog.h"

#include "SortingByMoneyDialog.h"

#include "TransferMoneyDialog.h"

MenuDialog::MenuDialog(IDialogManager\* dialogManager) : Dialog(dialogManager)

{

data = {

"Открыть счёт\n",

"Закрыть счёт\n",

"Перевести деньги на другой счёт\n",

"Вывести все счета\n",

"Вывести счета, отсортированные по операциям\n",

"Вывести счета, отсортированные по доходности\n",

"Выйти из программы"

};

}

void MenuDialog::HandleInput(int& currentLine, const char& INPUT)

{

if (INPUT == Keys::Enter)

{

switch (currentLine)

{

case ELine::AddAccount:

{

dialogManager->ShowDialog(new AddAccountDialog(dialogManager));

break;

}

case ELine::DeleteAccount:

{

dialogManager->

ShowDialog(new AccountDeletionDialog(dialogManager));

break;

}

case ELine::TransferMoney:

{

dialogManager->

ShowDialog(new TransferMoneyDialog(dialogManager));

break;

}

case ELine::ShowAllAccounts:

{

dialogManager->

ShowDialog(new AccountsListDialog(dialogManager));

break;

}

case ELine::SortByOperations:

{

dialogManager->

ShowDialog(new SortingByOperationsDialog(dialogManager));

break;

}

case ELine::SortByMoney:

{

dialogManager->

ShowDialog(new SortingByMoneyDialog(dialogManager));

break;

}

case ELine::CloseProgram:

{

exit(0);

}

}

}

else

{

Dialog::HandleInput(currentLine, INPUT);

}

}

# Код класса AddAccountDialog

Файл AddAccountDialog.h

#pragma once

#include "Dialog.h"

#include "Account.h"

class AddAccountDialog : public Dialog

{

public:

AddAccountDialog(IDialogManager\* dialogManager);

private:

enum ELine

{

CreditAccount = 0,

DepositAccount = 1,

CheckingAccount = 2,

Back = 3

};

void HandleInput(int& currentLine, const char& INPUT) override;

void ClearWindow() const;

void PassNewAccount(Account\* newAccount) const;

};

Файл AddAccountDialog.cpp

#include "AddAccountDialog.h"

#include "AccountFactory.h"

#include "IAccountsOwner.h"

AddAccountDialog::AddAccountDialog(IDialogManager\* dialogManager) : Dialog(dialogManager)

{

data = { "Кредитный счёт\n", "Депозитный счёт\n", "Расчётный счёт\n", "Назад" };

}

void AddAccountDialog::HandleInput(int& currentLine, const char& INPUT)

{

if (INPUT == Keys::Enter)

{

switch (currentLine)

{

case ELine::Back:

{

dialogManager->CloseDialog();

break;

}

case ELine::CreditAccount:

{

ClearWindow();

Account\* newAccount = AccountFactory::MakeCreditAccount();

PassNewAccount(newAccount);

break;

}

case ELine::DepositAccount:

{

ClearWindow();

Account\* newAccount = AccountFactory::MakeDepositAccount();

PassNewAccount(newAccount);

break;

}

case ELine::CheckingAccount:

{

ClearWindow();

Account\* newAccount = AccountFactory::MakeCheckingAccount();

PassNewAccount(newAccount);

break;

}

}

}

else

{

Dialog::HandleInput(currentLine, INPUT);

}

}

void AddAccountDialog::ClearWindow() const

{

system("cls");

}

void AddAccountDialog::PassNewAccount(Account\* newAccount) const

{

dialogManager->GetAccountOwner()->AddAccount(newAccount);

system("pause");

}

# Код класса AccountDeletionDialog

Файл AccountDeletionDialog.h

#pragma once

#include "Dialog.h"

#include "Account.h"

#include <set>

class AccountDeletionDialog : public Dialog

{

public:

AccountDeletionDialog(IDialogManager\* dialogManager);

private:

std::set<Account\*> userAccounts;

void HandleInput(int& currentLine, const char& INPUT) override;

void SetBackLineId();

};

Файл AccountDeletionDialog.cpp

#include "AccountDeletionDialog.h"

#include "IAccountsOwner.h"

AccountDeletionDialog::AccountDeletionDialog(IDialogManager\* dialogManager) : Dialog(dialogManager)

{

userAccounts = dialogManager->GetAccountOwner()->GetAccounts();

for (Account\* currentAccount : userAccounts)

{

data.push\_back(currentAccount->GetShortData());

}

data.push\_back("Назад");

SetBackLineId();

}

void AccountDeletionDialog::SetBackLineId()

{

backLine = data.size() - 1;

}

void AccountDeletionDialog::HandleInput(int& currentLine, const char& INPUT)

{

if (INPUT == Keys::Enter)

{

if (currentLine == backLine)

{

dialogManager->CloseDialog();

}

else

{

std::set<Account\*>::iterator accountsIterator = userAccounts.begin();

std::advance(accountsIterator, currentLine);

Account\* account = \*accountsIterator;

dialogManager->GetAccountOwner()->DeleteAccount(account);

userAccounts = dialogManager->GetAccountOwner()->GetAccounts();

data.erase(data.begin() + currentLine);

SetBackLineId();

}

}

else

{

Dialog::HandleInput(currentLine, INPUT);

}

}

# Код класса TransferMoneyDialog

Файл TransferMoneyDialog.h

#pragma once

#include "Dialog.h"

#include "Account.h"

#include <set>

class TransferMoneyDialog :

public Dialog

{

public:

TransferMoneyDialog(IDialogManager\* dialogManager);

private:

std::set<Account\*> userAccounts;

Account\* sender;

Account\* receiver;

void UpdateData();

void HandleInput(int& currentLine, const char& INPUT) override;

Account\* GetAccountById(const int& id);

bool CanSendMoney();

void SendMoney();

};

Файл TransferMoneyDialog.cpp

#include "TransferMoneyDialog.h"

#include "IAccountsOwner.h"

#include <iostream>

TransferMoneyDialog::TransferMoneyDialog(IDialogManager\* dialogManager) : Dialog(dialogManager)

{

UpdateData();

}

void TransferMoneyDialog::UpdateData()

{

sender = nullptr;

receiver = nullptr;

data.clear();

userAccounts = dialogManager->GetAccountOwner()->GetAccounts();

for (Account\* currentAccount : userAccounts)

{

data.push\_back(currentAccount->GetData());

}

data.push\_back("Назад");

backLine = data.size() - 1;

}

void TransferMoneyDialog::HandleInput(int& currentLine, const char& INPUT)

{

if (INPUT == Keys::Enter)

{

if (currentLine == backLine)

{

dialogManager->CloseDialog();

}

else

{

if (sender == nullptr)

{

sender = GetAccountById(currentLine);

} else if (receiver == nullptr)

{

receiver = GetAccountById(currentLine);

}

if (CanSendMoney())

{

SendMoney();

}

}

} else

{

Dialog::HandleInput(currentLine, INPUT);

}

}

Account\* TransferMoneyDialog::GetAccountById(const int& id)

{

std::set<Account \*>::iterator accountsIterator = userAccounts.begin();

std::advance(accountsIterator, id);

return \*accountsIterator;

}

bool TransferMoneyDialog::CanSendMoney()

{

return sender != nullptr && receiver != nullptr;

}

void TransferMoneyDialog::SendMoney()

{

system("cls");

std::cout << "Сумма: ";

double money = 0.0;

std::cin >> money;

if (sender->TransferMoney(receiver, money))

{

std::cout << "Операция прошла успешно!";

} else

{

std::cout << "Ошибка!";

}

std::cout << std::endl;

UpdateData();

system("pause");

}

# Код класса AccountsListDialog

Файл AccountsListDialog.h

#pragma once

#include "Dialog.h"

class AccountsListDialog : public Dialog

{

public:

AccountsListDialog(IDialogManager\* dialogManager);

private:

void HandleInput(int& currentLine, const char& INPUT) override;

};

Файл AccountsListDialog.cpp

#include "AccountsListDialog.h"

#include "Account.h"

#include "IAccountsOwner.h"

#include <set>

AccountsListDialog::AccountsListDialog(IDialogManager\* dialogManager) : Dialog(dialogManager)

{

std::set<Account\*> userAccounts = dialogManager->GetAccountOwner()->GetAccounts();

for (Account\* currentAccount : userAccounts)

{

data.push\_back(currentAccount->GetData());

}

data.push\_back("Назад");

backLine = data.size() - 1;

}

void AccountsListDialog::HandleInput(int& currentLine, const char& INPUT)

{

if (INPUT == Keys::Enter && currentLine == backLine)

{

dialogManager->CloseDialog();

} else

{

Dialog::HandleInput(currentLine, INPUT);

}

}

# Код класса SortingByOperationsDialog

Файл SortingByOperationsDialog.h

#pragma once

#include "Dialog.h"

class SortingByOperationsDialog : public Dialog

{

public:

SortingByOperationsDialog(IDialogManager\* dialogManager);

private:

void HandleInput(int& currentLine, const char& INPUT) override;

};

Файл SortingByOperationsDialog.cpp

#include "SortingByOperationsDialog.h"

#include <set>

#include <vector>

#include <algorithm>

#include "IAccountsOwner.h"

SortingByOperationsDialog::SortingByOperationsDialog(IDialogManager\* dialogManager) : Dialog(dialogManager)

{

std::set<Account\*> userAccountsSet = dialogManager->GetAccountOwner()->GetAccounts();

std::vector<Account\*> userAccountsVector(userAccountsSet.begin(), userAccountsSet.end());

std::sort(userAccountsVector.begin(), userAccountsVector.end(),

[] (Account\* a, Account\* b) { return a->GetNumberOfOperations() > b->GetNumberOfOperations(); });

for (Account\* account : userAccountsVector)

{

data.push\_back(account->GetShortData());

}

data.push\_back("Назад");

backLine = data.size() - 1;

}

void SortingByOperationsDialog::HandleInput(int& currentLine, const char& INPUT)

{

if (INPUT == Keys::Enter && currentLine == backLine)

{

dialogManager->CloseDialog();

}

else

{

Dialog::HandleInput(currentLine, INPUT);

}

}

# Код класса SortingByMoneyDialog

Файл SortingByMoneyDialog.h

#pragma once

#include "Dialog.h"

class SortingByMoneyDialog : public Dialog

{

public:

SortingByMoneyDialog(IDialogManager\* dialogManager);

private:

void HandleInput(int& currentLine, const char& INPUT);

};

Файл SortingByMoneyDialog.cpp

#include "SortingByMoneyDialog.h"

#include "IAccountsOwner.h"

#include <set>

#include <vector>

#include <algorithm>

SortingByMoneyDialog::SortingByMoneyDialog(IDialogManager\* dialogManager) : Dialog(dialogManager)

{

std::set<Account\*> userAccountsSet = dialogManager->GetAccountOwner()->GetAccounts();

std::vector<Account\*> userAccountsVector(userAccountsSet.begin(), userAccountsSet.end());

std::sort(userAccountsVector.begin(), userAccountsVector.end(), [](Account\* a, Account\* b) { return a->GetMoney() > b->GetMoney(); });

for (Account\* account : userAccountsVector)

{

data.push\_back(account->GetShortData());

}

data.push\_back("Назад");

backLine = data.size() - 1;

}

void SortingByMoneyDialog::HandleInput(int& currentLine, const char& INPUT)

{

if (INPUT == Keys::Enter && currentLine == backLine)

{

dialogManager->CloseDialog();

} else

{

Dialog::HandleInput(currentLine, INPUT);

}

}

# Код класса AccountFactory

Файл AccountFactory.h

#pragma once

#include "Account.h"

class AccountFactory

{

public:

static Account\* MakeCreditAccount();

static Account\* MakeDepositAccount();

static Account\* MakeCheckingAccount();

private:

template<typename T>

static void Input(T& destination);

};

Файл AccountFactory.cpp

#include "AccountFactory.h"

#include "CreditAccount.h"

#include "DepositAccount.h"

#include "CheckingAccount.h"

#include <iostream>

Account\* AccountFactory::MakeCreditAccount()

{

double money = 0.0;

double termMoney = 0.0;

float interestRate = 0.f;

short int term = 0;

std::cout << "Сумма кредитования: ";

Input(termMoney);

std::cout << "Срок кредитования (в годах): ";

Input(term);

std::cout << "Ставка по кредиту (в процентах): ";

Input(interestRate);

return new CreditAccount(money, interestRate, term, termMoney);

}

template<typename T>

void AccountFactory::Input(T& destination)

{

std::cin >> destination;

}

Account\* AccountFactory::MakeDepositAccount()

{

double money = 0.0;

double termMoney = 0.0;

float interestRate = 0.f;

short int term = 0;

std::cout << "Срок депозита (в годах): ";

Input(term);

std::cout << "Процент по вкладу: ";

Input(interestRate);

std::cout << "Взнос: ";

Input(money);

return new DepositAccount(money, interestRate, term, termMoney);

}

Account\* AccountFactory::MakeCheckingAccount()

{

double money = 0.0;

std::cout << "Сумма на счёте: ";

Input(money);

return new CheckingAccount(money);

}