Содержание

[Введение 3](#_Toc500748246)

[1 Анализ исходных данных 5](#_Toc500748247)

[2 Программное проектирование 6](#_Toc500748248)

[3 Программная реализация 6](#_Toc500748249)

[3.1 Детальная реализация функциональных частей ПО 6](#_Toc500748250)

[3.2 Сопроводительная документация 6](#_Toc500748251)

[3.3 Анализ ПО 6](#_Toc500748252)

[3.4 Тестирование ПО 6](#_Toc500748253)

[Заключение 7](#_Toc500748254)

[Литература 8](#_Toc500748255)

[Приложение А 9](#_Toc500748256)

# Введение

Написание программного обеспечения для разных ОС – сложная задача. Главное препятствие, стоящем перед программистом: у каждой ОС есть свой API, которое не похоже на API других ОС, что препятствует переносимости кода. Однако каждая ОС по-своему раскрывает возможности оборудования и по-своему развязывает руки программистам, что позволяет им писать по-настоящему эффективный код, реализующий необходимый функционал.

Современное программирование стало гораздо проще благодаря появлению новых, более высокоуровневых языков программирования, позволяющих создавать более высокоуровневые абстракции из более примитивных типов. Особую роль сыграло появление высокоуровневых языков программирования, выполняемых в виртуальных средах исполнения, таких как Java и C#. Эти языки программирования позволили программисту перестать думать о особенностях разных ОС, а, вместо этого, сосредоточить своё внимание на написании эффективного и чистого кода. Однако эти языки программирования имеют и свои недостатки – при выполнение программы их промежуточный код необходимо компилировать в исполняемый код, что немного замедляет работу приложения. Поэтому в настоящие время до сих пор актуальны такие языки программирования, как C и C++. Так же эти языки используются для написания кода, специфичного для конкретной ОС, в кроссплатформенных языках программирования.

Моей задачей является написание объектно-ориентированной обёртки на Win32 API для создания высокоуровневой абстракции над API windows, которая к тому же будет легко интегрироваться в код, написанный на C++ с вызовами Win32 API процедур.

Объектно-ориентированный подход к программированию популярен на сегодняшний день. Он облегчает написание кода за счёт введения абстракций, которых так не хватало процедурно-ориентированному подходу. Однако существует не так уж и много C++ библиотек, которые могли бы предоставить полное покрытие Win32 API, предоставить интерфейс для создания GUI и при этом не превратиться в фреймворк. Есть широко известный фреймворк, который был разработан самой компанией Microsoft - Microsoft Foundation Classes (далее MFC). Этот фреймворк развивается достаточно давно и успел обрасти как полезным функционалом, так и лишними запутанными связями, что препятствует продолжению активного развития фреймворка без опасения «что-то сломать». К тому же сам факт того, что MFC – фреймворк, заставляет хорошенько подумать перед тем как выбрать его. Это связано с тем, что фреймворки сковывают программиста и лишают его возможности «играть по своим правилам».

Так же существует ещё одно популярное решение – Windows Template Library (далее WTL). WTL – это свободно распространяемая библиотека шаблонов для написания GUI приложений для операционной системы Windows, однако данная библиотека решает вопрос построения GUI и никак не решает всего остального.

Популярный фреймворк QT является лучшим решением для написания прикладных программ с GUI, однако в нём нет возможности прямого использования API операционной системы из-за своей кроссплатформенности.

Одним из самых популярных решений является библиотека GIMP ToolKit plus (далее GTK+). GTK+ предоставляет простой интерфейс для разработки графических приложений, однако данная библиотека написана на языке программирования C, что заметно в структуре программ, использующих эту библиотеку.

wxWidgets – кроссплатформенная библиотека для разработки кроссплатформенных приложений. Библиотека не только отлично справляется со своей основной задачей – эффективное построение UI, но и ещё позволяет выполнять ряд других задач. wxWidgets позволяет сосредоточить всё внимание на написание логики приложения, а не на построение его базового интерфейса. Так же эта библиотека распространяется под лицензией LGPL, что позволяет внедрять решения, основанные на wxWidgets в закрытые проекты. Однако вновь появляются ограничения, связанные с кроссплатформенностью данной библиотеки.

Все рассмотренные библиотеки и фреймворки отлично подходят для решения огромного спектра задач, однако они не решают одну специфичную, но часто встречающуюся задачу: предоставление доступа к низкоуровневому API операционной системы и возможность интегрирования низкоуровневого процедурного кода Win32 API с высокоуровневыми абстракциями на основе этого API.

# 1 Анализ исходных данных

Первоначальной задачей моего проекта является написание объектно-ориентированной обёртки над частью функционала Win32 API для организации удобного объектно-ориентированного построения GUI и широким использованием идиомы RAII при написание кода, для обеспечения его надёжности. Перед началом проектирования необходимо выполнить анализ исходных данных. Далее по порядку:

Актуальная версия операционной системы Windows является Windows 10 Fall creators update (build 1709).

Графические технологии для построения:

GDI+ : объектно-ориентированная обёртка над GDI API;

Direct2D: объектно-ориентированная технология рендеринга двухмерных изображений на графическом процессоре. Является частью экосистемы DirectX;

DirectX 9-12: объектно-ориентирования технология, предоставляющая низкоуровневый доступ к ресурсам графического процессора для достижения максимально возможной эффективности видео рендеринга.

Для раскрытия всех возможностей API операционной системы Windows необходимо огромное количество времени, поэтому некоторые функциональные возможности придётся обойти стороной.

API DirectX – невероятно мощное решение, но на его изучение и эффективное внедрение уйдёт много времени. Что касается Direct2D – этот API по сложности не сильно далёк от старшего брата в лице DirectX.

GDI+ - простое решение, которое почти идеально подойдёт для базовой реализации, так как данная библиотека имеет достаточно простой интерфейс, что бы не тратить много времени на его изучение.

Язык программирования: C++.

С++ является очень мощным и гибким языком программирования, что позволит разработать более эффективные решения, по сравнению с другими решениями на других языках программирования. Современный стандарт ANSI C++17 обзавёлся множеством нововведений, которые призваны облегчить жизнь программистам и улучшить их производительность.

В рамках этого проекта необходимо разработать тестовое приложение, которое будет демонстрировать работу обёртки. Этим тестовым приложением будет программа «Windows explorer» - простой файловый обозреватель на базе данной обёртки для операционной системы Windows.

Необходимо, что бы эта программа могла выполнять все свои функциональные обязанности в полной мере на любой операционной системе Windows, начиная с версии Windows Vista (6.0).

# 2 Программное проектирование

Любое приложение в операционной системе Windows, общающееся с пользователем с помощью графического интерфейса должно иметь окно. Именно сущность окна и должна стать основополагающей для построения всей остальной логики. Первичное визуальное представление окна представлено на рисунке 2.1.

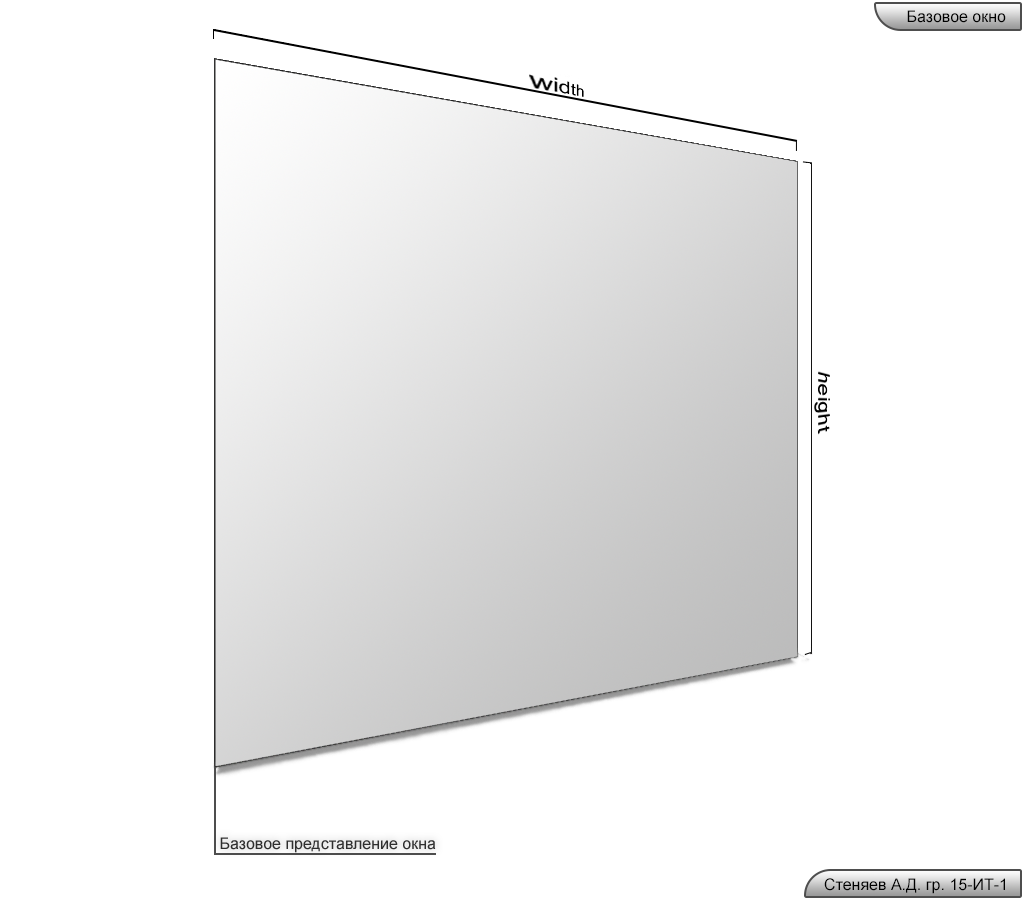


Рисунок 2.1 – базовое представление окна.

Для описания сущности окна введём класс Window, который будет описывать базовый функционал окна и инкапсулировать базовые обработчики от пользователя. Все последующие окна должны будут строится на основе базового класса Window. На рисунке 2.2 представлена диаграмма классов, расширяющих базовый класс Window.

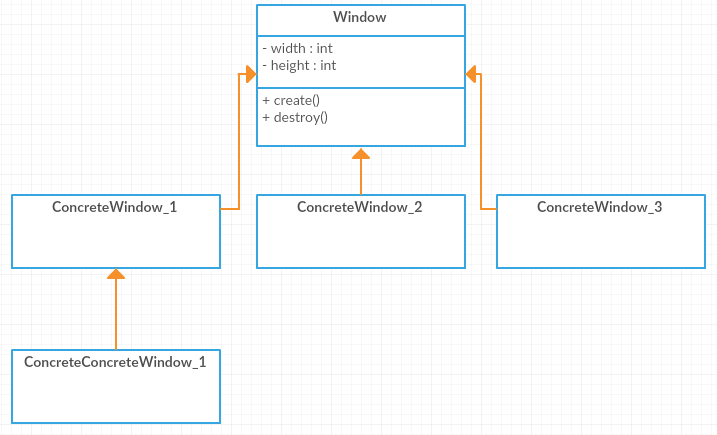


Рисунок 2.2 – диаграмма расширения базового класса.

Windows API предоставляет механизм создания дочерних окон, то есть окон, которые располагаются на поверхности другого окна, называемого родителем. На рисунке 2.3 показано, как располагаются окна «родитель» и «дочернее».

В процессе рисования сложных сцен (графики, составные картинки и прочее) и(или) их частой перерисовке начинает появляться неприятный эффект, называемый мерцанием. Это связанно с тем, что рисуемое изображение тут же отображается в видимой части окна, из-за чего пользователь видит весь процесс построения итогового изображения. Для предотвращения такого поведения используется достаточно простой метод – двойная буферизация. Суть метода заключается в том, чтобы рисовать сцену не в видимую часть окна, а в какую-то область памяти, из которой, при завершении рисования, перенести результат в видимую часть. Сам процесс копирования происходит намного быстрее, чем процесс рисования и отображение результата, поэтому метод является достаточно эффективным, однако он требует дополнительных ресурсов, что не является проблемой, так как в настоящие время современные компьютеры обладают достаточным запасом ресурсов, чтобы хранить в оперативной памяти несколько десятков таких буферов. В идеале размер буфера должен соответствовать размеру видимой области окна, что бы обновлялась вся область видимой части, однако буфер может иметь и больший размер, что полезно в случае, когда надо отображать большое количество данных, которые не влезают в рамки видимой области: можно создать буфер достаточно размера, а затем выводить в видимую область окна нужную его часть.

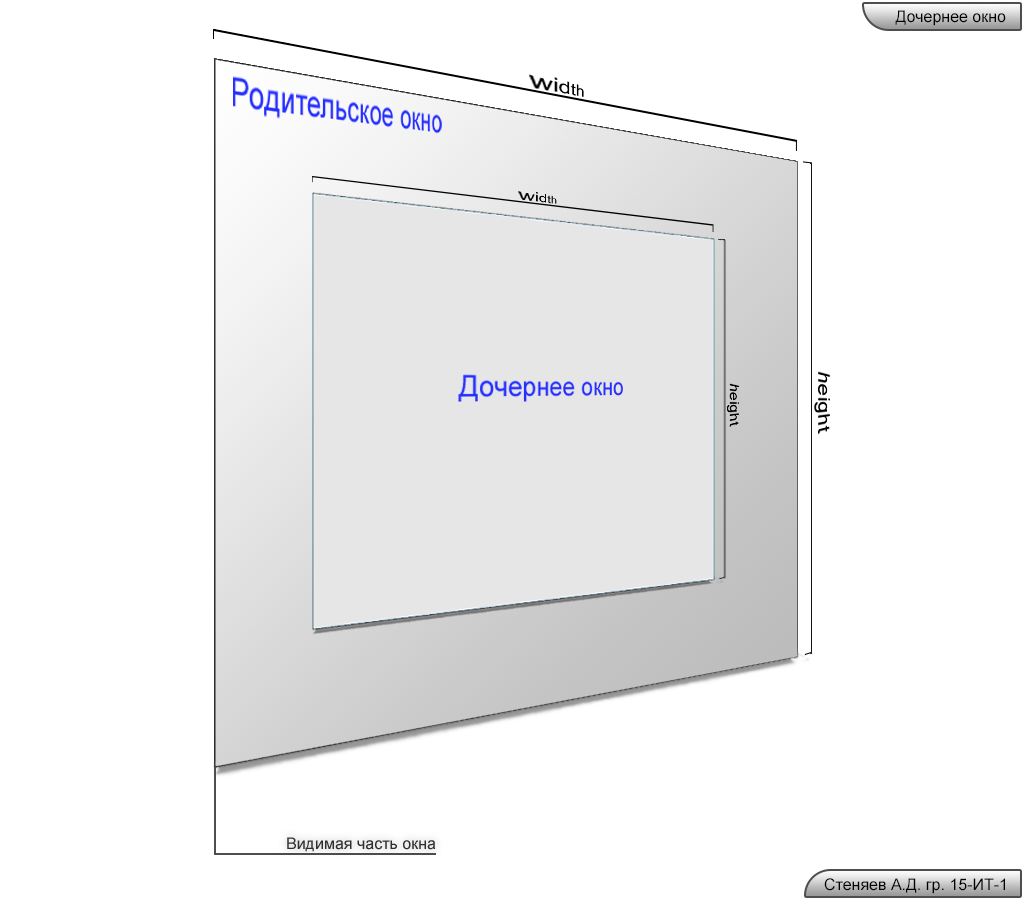


Рисунок 2.3 – расположение дочернего окна на поверхности родительского.

Рисование должно производиться в три этапа:

* 1. Подготовка изображения к рисованию.
  2. Рисование изображения в временную память (буфер).
  3. Копирование содержимого буфера в видимую часть окна.

При подготовке изображения выполняются самые базовые действия:

* загрузка всех составных частей;
* заполнение буфера базовым цветом (очистка);
* расчёт позиций всех элементов на сцене;
* преобразования, выполняемые над растровыми изображениями (вращение, масштабирование, выделение рисуемой части и т.д.);
* установка регионов рисования.

На рисунке 2.4 показан пример составленной сцены, подготовленной к рисованию в буфер.



Рисунок 2.4 – подготовка к рисованию сцены в буфер.

После подготовки начинается сам процесс рисования изображения. Каждая часть сцены последовательно рисуется в буфер. Пока идёт этот процесс, пользователь не видит изменений в видимой части окна, что не особо важно, если сцена простая, так как она рисуется достаточно быстро.

Результат рисования сцены в буфер показан на рисунке 2.5.



Рисунок 2.5 – Результат рисования сцены в буфер.

При завершении рисования, итоговый результат стоит скопировать в видимую часть или в другой буфер, если это необходимо. Делается это для того, чтобы не потерять результаты, накопленные в буфере. После копирования данных буфер можно использовать повторно до тех пор, пока он не будет уничтожен. Результат копирования изображения из буфера в видимую часть показан на рисунке 2.6.



Рисунок 2.6 – Результат копирования данных из буфера в видимую часть окна.

Для внедрения этого метода в проект необходимо написать класс, описывающий буфер и предоставляющий удобный интерфейс для доступа к данным из этого буфера:

1. Пользователь буфера должен иметь возможность получить доступ к данным этого буфера. В Win32 API буфер можно создать на основе карты битов (BITMAP), а для доступа к этой карте битов в процессе рисования нужен контекст устройства рисования (HDC). Из этого следует, что класс должен предоставлять доступ к своему контексту устройства, что бы в буфер можно было рисовать, и к своему «хенделу» карты битов, что бы из него в любой момент можно было получить всю необходимую информацию или использовать для более корыстных целей (разделение памяти между двумя буферами).
2. Буфер должен хранить информацию о своём размере, что бы пользователь или другой буфер могли в любой момент узнать размер этого буфера.
3. Пользователю в любой момент может понадобится тот же буфер, но большего размера. Это значит, что буфер должен иметь возможность менять свой размер при необходимости, чтобы адаптироваться к новым требованиям пользоваться.
4. Буфер может существовать сколь угодно долго, а его содержимое может быть достаточно востребовано. Следовательно, необходимо реализовать возможность копирования этих данных в другие буферы (например, из буфера в видимую часть окна).

На рисунке 2.7 показана диаграмма использования класса «Буфер изображения».

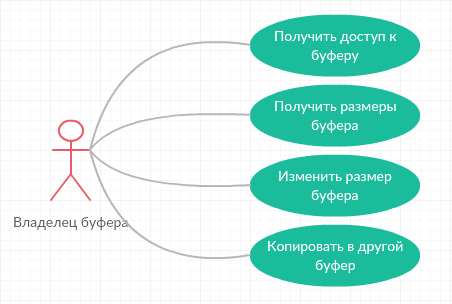


Рисунок 2.7 – диаграмма использования класса «Буфер изображения».

Для корректной работы программы окнам надо общаться. Для того что бы наладить общение между окнами не надо создавать много методов и хранить ссылки на другие окна. Для организации общения между окнами следует использовать паттерн проектирования «Слушатель». Использование данного паттерна позволит наладить связь между окнами следующим образом: окно, желающее получать уведомления о каких-либо действиях от другого окна, должно оформить подписку на соответствующие обновления. При возникновении какого-то события все подписавшиеся на рассылку окна будут получать уведомления о том, что это событие произошло. На основе этого можно наладить стабильное общение между окнами. Единственная проблема, которая может возникнуть в процессе разработки: установка этих связей. Для её решения необходимо грамотно подойти к проектированию как самой библиотеки, так и итогового приложения, уложившись при этом в короткий срок.

Пример установки связи между окнами показан на рисунке 2.8.

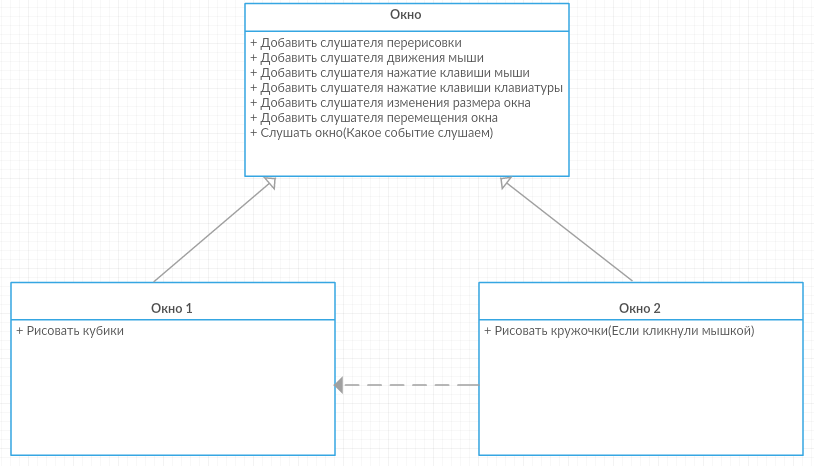


Рисунок 2.8 – Связь между окнами.

Некоторые данные и процедуры, которые являются общими для всех и могут потребоваться пользователю можно вынести в отдельный класс «System». К общим данным относятся данные о размерах экрана, данные о уровне яркости и звука, данные о аппаратном обеспечении и т.д. Класс System должен представлять единый интерфейс для получения всех необходимых системных данных. Тут всё просто: класс со статическими методами, которые возвращают какие-то системные данные в удобном для обработки виде.

# 3 Программная реализация

# 3.1 Детальная реализация функциональных частей ПО

# 3.2 Сопроводительная документация

# 3.3 Анализ ПО

# 3.4 Тестирование ПО

# Заключение

В ходе курсовой работы была сконструирована структурированная кабельная система для трех зданий. Выбрана схема расположения рабочих мест. Подсчитано необходимое количество материалов и оборудования, спроектированы вертикальная, горизонтальная и магистральная подсистема сети, выполнен расчет стоимости необходимого оборудования и материалов.

Также была выполнена настройка серверного сетевого программного обеспечения: сетевых параметров, DHCP-сервера, DNS-сервера, прокси-сервера, VLAN для малых подсетей, скрытой подсети; выполнена минимально-необходимая конфигурация клиентского сетевого оборудования: настроен доступ к прокси-серверу.

Таким образом, в ходе проделанной работы был создан проект, реализующий поставленную задачу. Были получены знания и опыт в конструировании структурированно кабельных систем.

# Литература

1) Википедия [Электронный ресурс] / *Wikimedia Foundation, Inc*. — Режим доступа: http://ru.wikipedia.org/. — Дата доступа: 14.05.2015.

2) IT it’s Easy [Электронный ресурс] – Режим доступа: <http://www.go-to-easyit.com>. – Дата доступа: 14.15.2015.

3) chelaxe@dfh [Электронный ресурс] – Режим доступа: <http://chelaxe.ru/> - Дата доступа: 14.15.2015.

4) IT Managers [Электронный ресурс] – Режим доступа: http://www.it-managers.ru/ - Дата доступа: 14.15.2015.

5) Хабрахабр [Электронный ресурс] – Режим доступа: <http://habrahabr.ru/> - Дата доступа: 14.15.2015.

# Приложение А

**План здания с указанием отделов, установленной компьютерной техники и коммуникаций**