Министерство образования Республики Беларусь
Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет информационной безопасности
Кафедра инфокоммуникационных технологий
Дисциплина технологии программирования инфокоммуникационных систем

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
к курсовому проекту
на тему
ПРОГРАММНОЕ СРЕДСТВО
БГУИР КП 6-05-0611-06 ПЗ

Студент М.Д. Гаврильчик
Руководитель В. В. Чепикова

Минск 2024

РЕФЕРАТ

ПРОГРАММНОЕ СРЕДСТВО : курсовой проект / М.Д. Гаврильчик. – Минск : БГУИР, 2024, – п.з. – 30 с., чертежей (плакатов) – 2 л. формата А4.

В реферативной части кратко излагается содержание курсового проекта. Основными аспектами в содержании должны быть: предмет проектирования (исследования); цель проекта; данные, относящиеся к методам проектирования; результаты и выводы.

Объем реферата ограничен текстом, который можно разместить на одной странице пояснительной записки. Рекомендуемый объем реферата 850–1200 печатных знаков.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Объектно-ориентированное программирование (ООП) представляет собой одну из методологий программирования. В отличие от процедурного подхода, который оно заменило, ООП делает акцент на организацию и структурирование кода, а не только на логику и процесс написания. Ключевым понятием ООП является объект, который объединяет данные и функции для их обработки. Каждый объект является экземпляром класса.

Класс выступает в роли шаблона для создания объектов. В нем определяются свойства (переменные) и методы (функции), которые доступны объектам этого класса. Благодаря механизму наследования классы могут использовать свойства и методы других классов, что упрощает разработку и повторное использование кода. Важным аспектом ООП является инкапсуляция, позволяющая скрыть внутренние детали реализации объекта, что делает код более защищенным и удобным для модификации. Полиморфизм, еще одна важная особенность, обеспечивает возможность объектов одного класса изменять свое поведение в зависимости от контекста использования.

Цель данного курсового проекта — создание программного обеспечения с использованием объектно-ориентированной методологии программирования.

Задачи курсового проекта:

1 Разработать графический интерфейс программного средства.

2 Разработать UML-диаграмму классов

3 Создать класс с указанными двумя полями и тремя методами: конструктор для инициализации объекта; метод формирования строки с информацией об объекте; метод обработки значений полей.

4 Разработать блок-схему алгоритма.

5 Создать дочерний класс с дополнительным полем. Реализовать в дочернем классе методы: конструктор; метод обработки данных.

1 СРЕДА РАЗРАБОТКИ ПРОГРАММНОГО СРЕДСТВА

1.1 Фреймворк Qt

Qt — это мощный кроссплатформенный фреймворк, предназначенный для разработки как графических, так и неграфических приложений. Его основным языком является C++. Qt позволяет запускать написанное с его помощью программное обеспечение в большинстве современных операционных систем путём простой компиляции программы для каждой системы без изменения исходного кода. Включает в себя все основные классы, которые могут потребоваться при разработке прикладного программного обеспечения, начиная от элементов графического интерфейса и заканчивая классами для работы с сетью, базами данных. Является полностью объектно-ориентированным, расширяемым и поддерживающим технику компонентного программирования. Qt активно используется для создания программного обеспечения, работающего на разных операционных системах, включая Windows, macOS, Linux, Android, iOS, а также встраиваемые системы.

Одной из ключевых особенностей Qt является его модульная структура. Она включает такие модули, как Qt Core для работы с базовыми структурами данных, Qt GUI для создания графических интерфейсов, Qt Widgets для реализации классических элементов интерфейса, Qt QML для разработки динамических интерфейсов с использованием JavaScript, Qt Multimedia для работы с аудио и видео, Qt Network для сетевых приложений, а также Qt SQL для интеграции с базами данных и Qt WebEngine для встраивания веб-контента. Эта гибкость позволяет адаптировать фреймворк под различные задачи, от классических десктопных программ до сложных систем управления.

Qt предоставляет развитые инструменты разработки, такие как Qt Creator — интегрированная среда разработки с поддержкой визуального редактирования интерфейсов, автодополнением кода и инструментами отладки. Помимо этого, доступны Qt Designer для проектирования графического интерфейса, Qt Linguist для локализации приложений и Qt Assistant для удобной работы с документацией. Этот набор инструментов существенно упрощает процесс разработки и делает его более эффективным.

Особое внимание в Qt уделяется поддержке OpenGL и рендеринга, что позволяет создавать приложения с богатой графикой и анимацией. Также стоит отметить кроссплатформенность, благодаря которой разработчики могут создавать приложения, работающие на разных операционных системах без необходимости значительных изменений в коде. Это делает Qt

популярным выбором для сложных проектов, требующих гибкости и высокой производительности.

Qt доступен как по открытой лицензии (GPL и LGPL), так и по коммерческой. Такой подход позволяет выбрать подходящий вариант в зависимости от требований проекта и особенностей его реализации. Этот фреймворк широко применяется для создания графических интерфейсов, мультимедийных приложений, программного обеспечения для встраиваемых систем, а также для разработки инструментов анализа данных и утилит.

Qt славится своей обширной документацией, которая подробно описывает API, предлагает примеры и обучающие материалы. Это делает его удобным для изучения и использования, несмотря на высокую кривую обучения для новичков, особенно тех, кто не знаком с C++. Несмотря на некоторую сложность и большой размер библиотек, фреймворк остается одним из самых мощных инструментов для создания универсальных приложений.

1.2 Язык программирования C++

C++ — это язык программирования общего назначения, созданный Бьёрном Страуструпом в начале 1980-х годов как расширение языка C. Он предоставляет средства объектно-ориентированного программирования, что делает его мощным инструментом для создания сложных программных систем. C++ поддерживает несколько парадигм программирования, включая процедурное, объектно-ориентированное и обобщённое программирование.

Язык компилируемый и статически типизированный, что означает, что типы данных проверяются во время компиляции. C++ компилируется в машинный код, что делает его быстрым и эффективным в плане использования ресурсов. Однако это также накладывает дополнительные требования на разработчиков по управлению памятью и обработке ошибок.

Одной из ключевых особенностей C++ является поддержка объектно-ориентированного программирования. Концепции, такие как классы, объекты, наследование, инкапсуляция и полиморфизм, позволяют строить модульные, расширяемые и легко сопровождаемые программы. Кроме того, язык поддерживает шаблоны, что позволяет писать универсальные и повторно используемые компоненты.

Стандартная библиотека C++ (STL, Standard Template Library) включает множество контейнеров, таких как векторы, списки, множества и ассоциативные массивы, а также алгоритмы для работы с ними. STL является

мощным инструментом для работы с данными, предоставляя высокоуровневые абстракции.

C++ позволяет работать с низкоуровневыми операциями, такими как управление памятью через указатели и использование встроенных в процессор инструкций. Это делает язык популярным для разработки системного ПО, драйверов, игр и приложений реального времени, где производительность критически важна.

Модернизация языка продолжает активно развиваться. Последние стандарты C++ (например, C++11, C++14, C++17, C++20 и C++23) добавляют новые функции, такие как умные указатели, многопоточность, лямбда-функции и новые типы данных. Это делает язык более удобным и безопасным для разработчиков, улучшая читаемость и надежность кода.

C++ используется в различных сферах: от разработки операционных систем и серверного ПО до сложных научных расчетов, финансовых приложений и игровых движков. Его популярность объясняется сочетанием высокой производительности, гибкости и широкого сообщества разработчиков.

Преимущества C++ включают высокую производительность благодаря компиляции в машинный код и детальному управлению ресурсами. Гибкость языка позволяет использовать его как для низкоуровневого программирования, так и для создания сложных приложений. Широкая поддержка парадигм программирования делает его универсальным инструментом для разработки ПО. Наличие мощной стандартной библиотеки (STL) ускоряет процесс разработки, предоставляя готовые структуры данных и алгоритмы. C++ поддерживает переносимость: программы могут компилироваться под различные платформы с минимальными изменениями. Кроме того, язык имеет большую экосистему инструментов, библиотек и активное сообщество разработчиков.

Недостатки C++ связаны с его сложностью и требованиями к разработчику. Управление памятью вручную может приводить к утечкам памяти и ошибкам, таким как двойное освобождение. Высокий уровень сложности языка делает его сложным для изучения, особенно для новичков. Компиляция может быть медленной из-за необходимости обработки большого объёма кода, включая заголовочные файлы. Кроме того, из-за множества особенностей языка, таких как неявное преобразование типов или шаблоны, код может становиться сложным для сопровождения. Наконец, из-за низкоуровневых возможностей языка велика вероятность появления ошибок безопасности, таких как переполнение буфера или неправильное использование указателей.

1.3 Интеграция C++ и Qt

Интеграция C++ и Qt представляет собой мощный инструмент для создания современных приложений с графическим интерфейсом и широким функционалом.

Основой работы с Qt в C++ является модульная структура фреймворка. Qt предоставляет модули для работы с графическим интерфейсом (Qt Widgets, Qt Quick), работы с файлами и потоками (Qt Core), сетевого взаимодействия (Qt Network), баз данных (Qt SQL), мультимедиа (Qt Multimedia) и других задач. Использование C++ в сочетании с этими модулями обеспечивает производительность и гибкость при разработке приложений.

Одной из ключевых особенностей Qt является система сигналов и слотов, которая используется для передачи событий между объектами. Эта концепция заменяет традиционные механизмы обратных вызовов в C++ и упрощает обработку событий. Примером может служить связь между кнопкой и функцией: когда пользователь нажимает кнопку, генерируется сигнал, который соединяется с нужным слотом — функцией, выполняющей логику обработки события [2].

Qt поддерживает инструменты автоматической генерации кода. Специальный препроцессор MOC (Meta-Object Compiler) анализирует классы, которые используют макросы Qt (например, Q_OBJECT), и генерирует дополнительный код для поддержки сигналов, слотов, динамического приведения типов и других особенностей. Это позволяет интегрировать дополнительные возможности в стандартный C++.

Для разработки интерфейсов в Qt можно использовать как код (на основе Qt Widgets), так и декларативный язык QML, который сочетает JavaScript и JSON-подобный синтаксис. QML особенно полезен для создания анимаций и современных адаптивных интерфейсов. C++ может быть интегрирован с QML для реализации бизнес-логики, когда требуется высокая производительность или доступ к низкоуровневым функциям.

Qt Creator, IDE, поставляемая с Qt, упрощает процесс интеграции C++ и Qt. Она включает в себя визуальные инструменты для проектирования интерфейса, отладчик, профилировщик, и поддержку различных сборочных систем, включая qmake, CMake и QBS. IDE предоставляет автозаполнение, документацию и примеры для ускорения разработки [3].

Qt предоставляет мощную систему локализации, что делает приложения многоязычными. Она включает в себя инструменты для извлечения строк из кода, создания файлов перевода и их последующего применения в приложении. C++ позволяет управлять локализацией на уровне логики, используя ресурсы Qt.

Qt доступен в нескольких версиях: коммерческой и с открытым исходным кодом (Open Source). Коммерческая версия предоставляет дополнительные инструменты, техническую поддержку и лицензии для закрытого исходного кода. Open Source версия лицензируется под LGPL или GPL, что накладывает определённые ограничения на её использование в проприетарных проектах.

Интеграция C++ и Qt обеспечивает высокий уровень производительности, гибкости и удобства при разработке приложений, делая этот тандем популярным выбором среди разработчиков программного обеспечения.

2 РАЗРАБОТКА ГРАФИЧЕСКОГО ИНТЕРФЕЙСА ПРОГРАММНОГО СРЕДСТВА

2.1 Графический интерфейс

Графический пользовательский интерфейс (GUI) представляет собой одну из самых распространенных форм взаимодействия между пользователем и компьютером. Основное назначение GUI заключается в предоставлении пользователю удобного, интуитивно понятного способа работы с программами и данными. В этом интерфейсе элементы управления, такие как окна, кнопки, меню, текстовые поля, иконки и изображения, визуально отображаются на экране, что позволяет пользователю легко распознавать их функции. Взаимодействие с такими элементами осуществляется с помощью периферийных устройств, таких как мышь, клавиатура или сенсорные экраны, что делает процесс использования программного обеспечения более доступным и понятным.

Одной из ключевых характеристик графического пользовательского интерфейса является его визуальность. Графические элементы создают наглядное представление функциональности программы, позволяя пользователю интуитивно понимать, как выполнить ту или иную задачу. Например, кнопка с изображением мусорной корзины сразу дает понять, что она используется для удаления объектов. Такая визуальная ясность значительно снижает необходимость в подробных инструкциях и обучении, особенно для новых пользователей.

Еще одной важной особенностью GUI является интерактивность. Пользователь может взаимодействовать с элементами интерфейса в режиме реального времени, выполняя различные действия: нажатие кнопок, выбор пунктов меню, перемещение объектов с помощью перетаскивания или ввод данных в текстовые поля. Это делает интерфейс динамичным и удобным, позволяя быстро достигать желаемых результатов. Интерактивность также способствует реализации сложных функций, таких как работа с мультимедиа или настройка параметров, через простые графические компоненты.

GUI поддерживает многозадачность, что позволяет пользователю одновременно работать с несколькими окнами или приложениями. Например, в операционных системах с графическим интерфейсом пользователь может редактировать текстовый документ, просматривать веб-страницу и общаться в мессенджере, переключаясь между задачами без необходимости закрывать одно приложение для открытия другого. Это повышает производительность и комфорт работы, особенно при выполнении комплексных задач.

Использование графических элементов, таких как иконки и изображения, делает интерфейс не только функциональным, но и эстетически приятным. Иконки позволяют заменить длинные текстовые описания простыми символами, которые легко запоминаются и ассоциируются с определенными действиями или функциями. Например, значок с изображением папки ассоциируется с доступом к файлам, а иконка с изображением шестеренки — с настройками.

Меню и контекстные действия играют важную роль в организации функциональности приложения. Главное меню обеспечивает доступ к основным функциям, тогда как контекстные меню, вызываемые правой кнопкой мыши или долгим нажатием на сенсорных устройствах, предлагают действия, применимые к конкретному объекту. Это упрощает работу пользователя, исключая необходимость запоминать сложные команды.

Современные графические интерфейсы включают в себя элементы анимации и переходов, которые делают взаимодействие более плавным и приятным. Например, анимация появления окна или эффект наведения курсора на объект подсказывают пользователю, какие действия доступны в данный момент. Кроме того, использование анимации позволяет эффективно привлекать внимание к важным изменениям на экране, например, к уведомлениям или ошибкам.

Некоторые GUI предоставляют возможности для пользовательской настройки, позволяя изменять внешний вид и функциональность интерфейса под индивидуальные предпочтения. Пользователи могут выбирать цветовые схемы, изменять размеры и расположение элементов, а также настраивать шорткаты для ускорения выполнения определенных действий. Такая гибкость особенно полезна для пользователей с особыми потребностями или предпочтениями.

2.2 Qt Designer

Qt Designer — это мощный инструмент, предоставляемый фреймворком Qt, который значительно упрощает процесс разработки графических пользовательских интерфейсов (GUI) для приложений. Этот инструмент ориентирован на создание интерфейсов с использованием визуального редактора, который позволяет разработчикам проектировать и редактировать формы без необходимости писать код вручную. В результате, использование Qt Designer значительно ускоряет процесс разработки и улучшает качество интерфейсов, что особенно полезно при создании сложных приложений с множеством элементов управления.

Основной особенностью Qt Designer является возможность визуального проектирования. Разработчики могут с легкостью перетаскивать различные визуальные компоненты, такие как кнопки, текстовые поля, таблицы, метки и другие элементы управления, на форму, где они могут сразу увидеть, как интерфейс будет выглядеть в окончательном варианте. Это интуитивно понятный процесс, который позволяет сосредоточиться на внешнем виде и функциональности интерфейса, исключая необходимость вручную прописывать код для каждого компонента.

Qt Designer тесно интегрирован с фреймворком Qt, что позволяет разработчику seamlessly использовать созданные формы в проекте, не беспокоясь о проблемах совместимости. Это особенно важно, когда речь идет о написании кода на C++ или других языках программирования, поддерживаемых Qt. Все элементы, созданные в Qt Designer, можно легко связать с логикой приложения, обеспечивая гладкое взаимодействие между интерфейсом и функционалом программы.

Кроме того, Qt Designer поддерживает создание и использование пользовательских виджетов и компонентов. Разработчики могут создавать собственные элементы управления, такие как кнопки или текстовые поля с индивидуальными характеристиками, а затем интегрировать их в редактор для дальнейшего использования. Это позволяет значительно расширить возможности интерфейса и адаптировать его под специфические потребности приложения, обеспечивая гибкость и многофункциональность.

Qt Designer также поддерживает создание многоязычных интерфейсов, что особенно важно для разработки программного обеспечения, ориентированного на глобальный рынок. Встроенные инструменты локализации позволяют легко перевести элементы интерфейса, такие как кнопки, меню и другие компоненты, на разные языки, что облегчает работу с многоязычными приложениями и делает их доступными для более широкой аудитории.

После того как интерфейс был спроектирован в Qt Designer, инструмент может автоматически генерировать код на C++, который можно использовать в проекте. Это значительно упрощает создание связей между графическим интерфейсом и бизнес-логикой приложения, поскольку разрабатывать код вручную для каждой отдельной компоненты уже не требуется. Это делает процесс интеграции интерфейса в проект быстрым и эффективным.

Qt Designer также предоставляет разработчикам возможность изменять стили и темы интерфейса, что позволяет адаптировать внешний вид приложения к различным платформам и требованиям дизайна. С помощью поддерживаемых стилей можно легко адаптировать интерфейс под

особенности конкретных операционных систем или даже под личные предпочтения пользователя, создавая уникальные и привлекательные приложения.

Наконец, Qt Designer включает функцию предварительного просмотра, позволяющую разработчикам тестировать и просматривать созданный интерфейс до его финальной интеграции в проект. Это позволяет выявить и исправить ошибки или внести изменения до того, как приложение будет окончательно собрано и представлено пользователю. Предварительный просмотр упрощает отладку интерфейса и позволяет быстро вносить необходимые улучшения.

В итоге, Qt Designer является незаменимым инструментом для разработки приложений с графическим интерфейсом. Он значительно упрощает процесс проектирования и создания интерфейсов, делая его доступным для разработчиков с разным уровнем опыта. Интеграция с другими инструментами и библиотеками Qt, а также его возможности по кастомизации и локализации интерфейсов делают его незаменимым при разработке приложений с использованием фреймворка Qt.

2.3 Разработка формы программного средства

Чтобы отобразить форму необходимо дважды кликнуть левой кнопкой мыши по названию файла .ui на вкладке "Проекты". Результат продемонстрирован на рисунке 2.1.

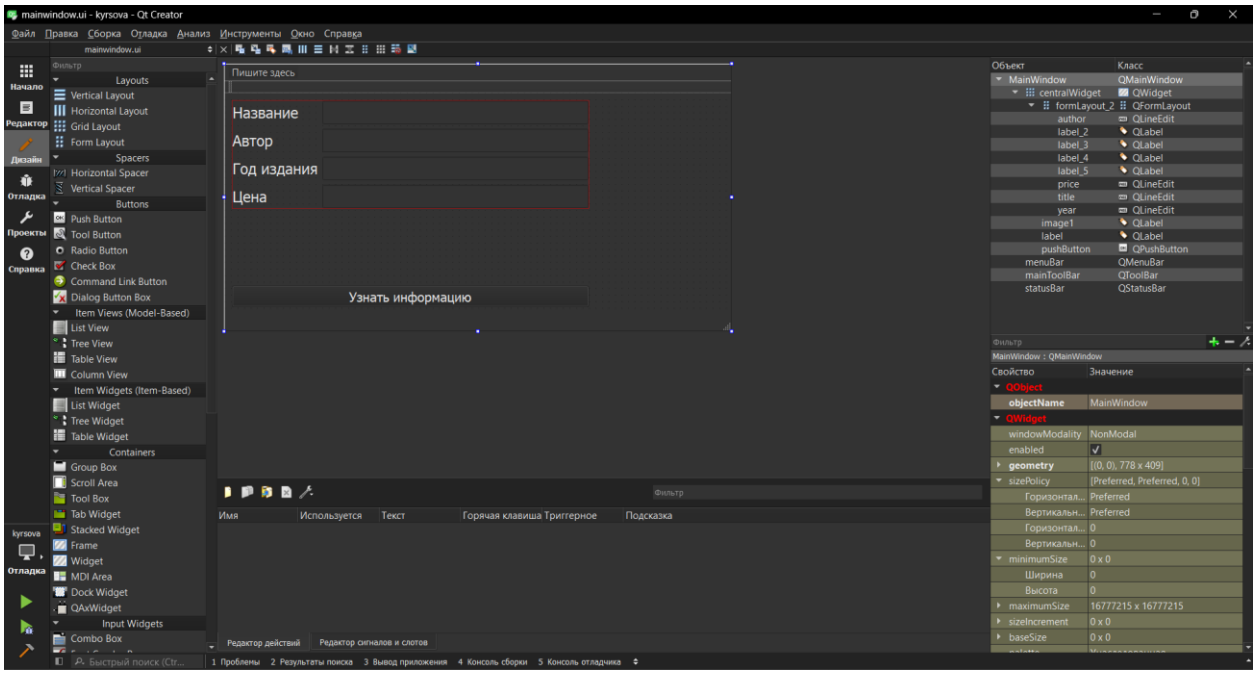

Рисунок 2.1 – Редактирование формы в режиме "Дизайн"

После того как вы открыли файл .ui и попали в режим дизайна, вы увидите окно, где можно разместить различные элементы управления.

– Выбор виджетов: В левой части экрана находится "Toolbox" с различными виджетами. Вы можете выбирать нужные элементы, такие как кнопки, текстовые поля, таблицы и другие. Перетащите его на форму.

– Настройка свойств: В меню "Свойства" вы можете настроить различные параметры выбранного виджета, такие как текст, размер, цвет и т.д. Это позволяет вам индивидуализировать внешний вид каждого элемента.

– Размещение элементов: Используйте возможности выравнивания и расположения для определения точного местоположения каждого элемента на форме. Вы можете выстраивать элементы в сетку для более аккуратного расположения (см. рисунок 2.2).

Рисунок 2.2 – Размещение элементов в режиме "Дизайн"

– Добавление стилей с помощью StyleSheet: Qt предоставляет механизм стилизации элементов интерфейса с использованием CSS-подобных StyleSheet. Вы можете применять стили к различным виджетам, изменяя их внешний вид. Для этого вы можете использовать встроенные инструменты Qt Designer или добавить стили вручную в коде. Выберите элемент, к которому хотите применить стиль, и в свойствах найдите раздел "StyleSheet".

StyleSheet предоставляет широкие возможности для кастомизации интерфейса, такие как изменение цветовой схемы, шрифтов, границ и т.д. Это полезный инструмент для придания уникального и профессионального внешнего вида вашему приложению без необходимости в глубоких изменениях в коде.

– Сохранение и компиляция: После завершения работы над формой, сохраните изменения и скомпилируйте проект. Затем вы сможете запустить ваше программное средство для тестирования.

В этой части разработки программного средства с использованием Qt мы рассмотрели процесс создания формы в режиме дизайна. При помощи инструментов Qt Designer вы можете интуитивно размещать и настраивать различные элементы управления, создавая пользовательский интерфейс для вашего приложения. Дальнейший этап включает в себя добавление функциональности через обработку событий и написание соответствующего кода.

3 РАЗРАБОТКА ПРОГРАММНОГО СРЕДСТВА

3.1 Язык графического описания UML

Unified Modeling Language (UML) представляет собой стандартизированный язык графического описания, предназначенный для визуализации, проектирования и документирования различных аспектов разработки программных систем. UML активно используется для объектного моделирования, что позволяет создавать абстракции, соответствующие реальным объектам, и описывать их взаимодействие. Важно отметить, что UML применяется не только в программировании, но и в моделировании бизнес-процессов, системном проектировании и отображении организационных структур. Он стал основным инструментом для создания архитектуры программных продуктов и взаимодействия между их компонентами, значительно улучшая понимание и поддержку сложных проектов.

UML тесно связано с объектно-ориентированным программированием (ООП), которое базируется на концепции объектов и классов. В ООП программа состоит из взаимодействующих объектов, каждый из которых является экземпляром класса. Эти объекты обладают атрибутами и могут выполнять различные методы. UML предоставляет необходимые средства для визуализации этих объектов, их атрибутов, методов и связей между ними. Диаграммы UML помогают разработчикам понять, как различные компоненты программы взаимодействуют друг с другом, какие данные обрабатываются, какие методы выполняются и как классы организуют систему. Это значительно упрощает проектирование, создание и тестирование программных систем.

В основе объектно-ориентированного подхода лежат несколько ключевых принципов: инкапсуляция, наследование и полиморфизм. UML позволяет наглядно представлять все эти аспекты, что делает его неоценимым инструментом для анализа и проектирования системы. На диаграммах классов, например, отображаются классы и их атрибуты, а также методы, которые описывают поведение объектов. Эти диаграммы также показывают, как классы могут быть связаны друг с другом через ассоциации, агрегации, композиции и наследование. Это дает разработчикам ясное представление о структуре системы и ее компонентах, а также позволяет выявить потенциальные проблемы на ранних этапах разработки.

Основной элемент UML — это диаграммы, представляющие различные аспекты системы. Диаграммы классов являются основой для визуализации структуры системы и ее компонентов. Они представляют собой шаблоны для

создания объектов, которые инкапсулируют данные и методы для манипуляции этими данными. В диаграммах классов классы отображаются в виде прямоугольников, которые содержат три секции: имя класса, атрибуты и методы. Атрибуты описывают характеристики объектов, а методы определяют действия, которые можно выполнить с этими объектами. Важно, что на диаграммах классов также отображаются отношения между классами, такие как ассоциации, агрегации, композиции и наследование, которые показывают, как классы связаны друг с другом и как взаимодействуют.

Диаграммы классов UML используют несколько типов отношений, каждый из которых имеет свои особенности. Ассоциация обозначает связь между объектами различных классов и может иметь различную множественность (например, один к одному или один ко многим). Агрегация и композиция — это более сложные типы ассоциации, где объекты одного класса являются частями другого класса. Агрегация предполагает, что объекты могут существовать независимо друг от друга, тогда как композиция подразумевает, что объект одного класса не может существовать без другого. Наследование в UML отображает отношения между суперклассами и подклассами, что позволяет строить иерархию классов и повторно использовать код. Наследование обычно отображается стрелкой, которая указывает от дочернего класса к родительскому, и может быть использована для описания общих и специализированных классов. Интерфейсы, в свою очередь, представляют собой контракты, которые определяют набор методов, которые классы должны реализовать.

В UML существует несколько других типов диаграмм, которые служат для описания других аспектов системы. Например, диаграмма последовательности отображает, как объекты взаимодействуют друг с другом в процессе выполнения системы, показывая последовательность сообщений между ними. Диаграмма состояний показывает возможные состояния объектов и переходы между ними в зависимости от событий, что полезно для моделирования динамических процессов. Диаграммы активностей и компонентов используются для моделирования более высокоуровневых аспектов системы, таких как процесс выполнения операций и взаимодействие между компонентами системы.

Одним из важных аспектов UML является возможность описания поведения системы с помощью различных диаграмм. Например, диаграммы последовательности и диаграммы состояний помогают анализировать, как объекты взаимодействуют во времени и как система изменяет свои состояния. Эти диаграммы играют важную роль в динамическом проектировании и тестировании программных систем. Это особенно важно при проектировании

сложных приложений, где взаимодействие между компонентами или объектами происходит в реальном времени, и важно отслеживать эти взаимодействия для оптимизации производительности и выявления ошибок.

Кроме того, диаграммы классов могут содержать дополнительные элементы, такие как комментарии и ограничения, которые уточняют поведение системы или раскрывают детали структуры. Это позволяет добавлять дополнительные пояснения и гарантировать, что все элементы системы имеют четкие и понятные описания, что облегчает работу всей команды разработчиков. Пояснения и ограничения помогают снизить риск неправильной интерпретации диаграмм и способствуют улучшению совместной работы.

Основные термины, используемые в UML-диаграммах классов, включают классы, объекты, атрибуты, методы, ассоциации, агрегации, композиции, наследование, интерфейсы, множественность и абстрактные классы. Каждый из этих элементов имеет свое значение в контексте моделирования и проектирования системы. Например, класс является шаблоном для создания объектов, объект — это экземпляр класса, атрибуты — это данные, принадлежащие классу, а методы — действия, которые могут быть выполнены с объектами. Наследование позволяет создавать иерархии классов, а агрегация и композиция помогают моделировать сложные отношения между компонентами системы.

UML является не только мощным инструментом для визуализации структуры системы, но и важным средством анализа и проектирования. С его помощью можно не только построить архитектуру системы, но и наглядно представить взаимосвязи между объектами и компонентами. Использование UML облегчает процесс разработки, делает код более понятным и сопровождаемым, что является критически важным для успешного завершения крупных проектов. Диаграммы классов UML особенно полезны при проектировании объектно-ориентированных приложений, поскольку они позволяют четко представлять структуру системы, ее компоненты и связи между ними, обеспечивая основу для эффективной разработки и дальнейшей поддержки программного обеспечения.

3.2 UML-диаграмма классов

В классе Book определены только те члены, которые являются общими для всех объектов этого класса. Все поля класса определены с атрибутом доступа protected, методы – public (см. рисунок 3.1). Описание класса Book, его полей и методов находится в файле Book.h.

```
class Book {
protected:
    std::string title;
    std::string author;
    int year;

public:
    // Конструктор
    Book(const std::string& title, const std::string& author, int year);

    // Методы
    int calculateBookAge() const;
    long long calculateDaysSincePublication() const;
    std::string displayBookInfo() const;
};
```

Рисунок 3.1 – Описание класса Book

Класс BookStore будет наследовать свойства класса Book. Все поля класса определены с атрибутом доступа private, методы – public (см. рисунок 3.2).

```
class BookStore : public Book {
private:
    double price;

public:
    // Конструктор
    BookStore(const std::string& title, const std::string& author, int year, double price);

    // Методы
    void discountPrice();
    std::string displayBookStoreInfo() const;
};
```

Рисунок 3.2 – Описание класса ChildProduct

UML-диаграмма классов программного средства представлена в приложении А.

3.3 Директивы препроцессора

Перед выполнением программы всегда выполняются директивы препроцессора, в основном они используются для подключения к программе заголовочных и библиотечных файлов. Перед выполнением программы всегда происходит обработка директив препроцессора, что позволяет осуществить различные манипуляции с исходным кодом программы до её компиляции. В данном проекте директивы препроцессора активно применяются для подключения заголовочных и библиотечных файлов, а также для условной компиляции и определения макросов [6].

Директивы #include используются для включения содержимого заголовочных файлов в исходный код программы. В данном проекте важными заголовочными файлами являются:

#include <QMessageBox>

#include <iostream>

#include <QIcon>

#include <QPixmap>

#include <QMessageBox> — подключает класс QMessageBox, который используется для создания и отображения диалоговых окон с различными типами сообщений. Эти окна могут быть информационными, предупреждающими или вопросительными, предоставляя пользователю возможность выбрать одну из предложенных кнопок для взаимодействия.

#include <iostream> — подключает стандартную библиотеку ввода/вывода для работы с консолью. Она используется для вывода данных в консоль с помощью объектов потока, таких как std::cout, а также для чтения ввода пользователя через std::cin.

#include <QIcon> — подключает класс QIcon, который позволяет работать с иконками. Он используется для отображения изображений в интерфейсе приложения, например, для иконок в кнопках, на вкладках или в заголовках окон.

#include <QPixmap> — подключает класс QPixmap, который предоставляет функциональность для работы с растровыми изображениями. Он используется для загрузки, обработки и отображения графических файлов, таких как PNG или JPEG, в приложении.

В коде присутствует использование директивы #ifdef, которая позволяет компилировать или исключать части кода в зависимости от наличия определенного макроса. Это может быть использовано, например, для включения или отключения отладочной информации в процессе отладки.

Блок-схема алгоритма работы метода родительского класса представлена в приложение Б.

3.4 Механизм сигналов и слотов

В Qt используется механизм сигналов и слотов. Сигнал вырабатывается, когда происходит определенное событие. Слот – это функция, которая вызывается в ответ на определенный сигнал.

В Qt механизм сигналов и слотов является основой для создания гибкой и отзывчивой архитектуры приложения. Сигналы и слоты позволяют объектам обмениваться информацией без необходимости жесткой связи между ними.

Для связывания сигналов и слотов используется функция connect, а для отсоединения — функция disconnect.

Этот механизм предоставляет возможность передавать параметры между объектами, что делает взаимодействие между ними более эффективным. Важно отметить, что Qt Designer значительно облегчает работу с этим механизмом, предлагая визуальные инструменты для создания и управления сигналами и слотами. Он также поддерживает дополнительные возможности, такие как сигналы с параметрами по умолчанию и использование лямбда-функций в качестве слотов [7].

Корректное использование сигналов и слотов способствует созданию модульного, легко поддерживаемого и тестируемого кода. Механизм сигналов и слотов обеспечивает низкосвязанную архитектуру, что способствует улучшению переиспользуемости кода и облегчает тестирование.

Кроме того, этот механизм идеально подходит для разработки графических интерфейсов. Он позволяет объектам, таким как кнопки и поля ввода, взаимодействовать друг с другом без необходимости вручную обрабатывать события, что существенно сокращает объем кода и позволяет сосредоточиться на более важной логике приложения.

Сигналы и слоты в Qt реализуют паттерн "Издатель-Подписчик", где объект (издатель) уведомляет другие объекты (подписчиков) о событии. Это позволяет создавать модульный и легко расширяемый код, что особенно важно при работе над большими и сложными проектами.

Листинг кода программного средства представлен в приложении В.

ЗАКЛЮЧЕНИЕ

В рамках курсового проекта было разработано программное средство, основанное на принципах объектно-ориентированного программирования (ООП) с использованием фреймворка Qt версии 5.14.2. Программное средство предназначено для решения задачи расчета стоимости, определения возраста и количества дней с момента публикации книги на основе введенных пользователем данных.

Для достижения поставленной цели были созданы следующие классы:

1 Book (Базовый класс для товара):Содержит поля для хранения названия (title), автора(author) и года выпуска (year). Реализует метод calculateBookAge(), который вычисляет возраст книги, и calculateDaysSincePublication(), который вычисляет количество дней с момента публикации книги.

2 BookStore (Дочерний класс, наследующий от Book): Добавляет поле для хранения цены книги (price). Реализует метод discountPrice(), который определяет стоимость книги с учётом её возраста.

Интерфейс программы разработан интуитивно понятным и легко воспринимаемым для пользователя. Добавленное ограничение на ввод, такое как валидация числовых значений, обеспечивает корректную работу программы.

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1 The Complete Guide (Addison-Wesley, 2003; русский перевод: Вандервурд Д. Джосаттис Н. Шаблоны C++: справочник разработчика. - М. : Издательский дом "Вильямс", 2003) и SOA in Practice: The Art of Distributed System Design (O'Reilly Media, 2007).

2 Qt Group. The Framework [Электронный ресурс]. – Режим доступа : https://www.qt.io/product/framework.

3 linux.org.ru. Какие плюсы и минусы существуют у Qt [Электронный ресурс]. – Режим доступа : https://www.linux.org.ru/forum/development/11471978.

4 Первая программа в Qt Creator [Электронный ресурс]. – Режим доступа : https://metanit.com/cpp/tutorial/1.7.php.

5 Диаграммы UML для моделирования процессов и архитектуры проекта [Электронный ресурс]. – Режим доступа : https://evergreens.com.ua/ru/articles/uml-diagrams.html.

6 Блог программиста. Блок-схемы алгоритмов [Электронный ресурс]. – Режим доступа : https://pro--prof-com.turbopages.org/turbo/pro-prof.com/s/archives/1462#part_1.

7 Хабр. Сигналы и слоты в Qt [Электронный ресурс]. – Режим доступа : https://habr.com/ru/post/50812/.

ПРИЛОЖЕНИЕ А
(обязательное)
UML-диаграмма классов программного средства

ПРИЛОЖЕНИЕ Б
(обязательное)
Блок-схема алгоритма работы программного средства

ПРИЛОЖЕНИЕ В
(обязательное)
Листинг кода программного средства

```
#ifndef BOOK_H
#define BOOK_H

#include <string>

class Book {
protected:
    std::string title;
    std::string author;
    int year;

public:
    // Конструктор
    Book(const std::string& title, const std::string& author, int year);

    // Методы
    int calculateBookAge() const;
    long long calculateDaysSincePublication() const;
    std::string displayBookInfo() const;
};

#endif // BOOK_H

#include "Book.h"
#include <iostream>
#include <ctime>
#include <sstream>


using namespace std;

// Конструктор
Book::Book(const string& title, const string& author, int year)
    : title(title), author(author), year(year) {
}

// Метод для вычисления возраста книги
int Book::calculateBookAge() const {
    time_t now = time(0);
    struct tm ltm = {};
    localtime_s(&ltm, &now); // Безопасная версия localtime
    int currentYear = 1900 + ltm.tm_year;
    return currentYear - year;
}

long long Book::calculateDaysSincePublication() const {
```

```
    time_t now = time(0);
    struct tm ltm = {};
    localtime_s(&ltm, &now); // Безопасная версия localtime

    // Количество дней, прошедших с начала года
    int dayOfYear = ltm.tm_yday + 1; // tm_yday дает количество дней с 1 января, начиная с 0

    // Возраст книги в годах
    int age = calculateBookAge();

    // Количество дней, прошедших за полные годы
    long long days = 365 * age + (age / 4); // Добавляем високосные года

    // Добавляем количество дней в текущем году
    days += dayOfYear;

    return days;
}



// Метод для формирования строки с информацией о книге
std::string Book::displayBookInfo() const {
    // Используем stringstream для формирования строки
    std::stringstream info;
    info << "Возраст книги: " << calculateBookAge() << " лет"
         << "\nДней с момента публикации: " << calculateDaysSincePublication() << " дней";
    return info.str(); // Возвращаем строку
}

#ifndef BOOKSTORE_H
#define BOOKSTORE_H

#include "Book.h"

class BookStore : public Book {
private:
    double price;

public:
    // Конструктор
    BookStore(const std::string& title, const std::string& author, int year, double price);

    // Методы
    void discountPrice();
    std::string displayBookStoreInfo() const;
};

#endif // BOOKSTORE_H
#include "Bookstore.h"
#include <iostream>
```

```
#include <sstream>

using namespace std;

// Конструктор
BookStore::BookStore(const string& title, const string& author, int year, double price)
    : Book(title, author, year), price(price) {
    discountPrice();
}

// Метод для уменьшения стоимости книги
void BookStore::discountPrice() {
    if (calculateBookAge() > 5) {
        price *= 0.8; // Скидка 20%
    }
}

// Метод для формирования строки с информацией о книге в магазине
string BookStore::displayBookStoreInfo() const {
    // Используем stringstream для формирования строки
    stringstream info;
    // Добавляем информацию из базового класса Book
    info << displayBookInfo();
    // Добавляем информацию о цене
    info << "\nЦена со скидкой: " << price << " BYN";
    return info.str(); // Возвращаем строку
}

#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>

namespace Ui {
class MainWindow;
}


class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    explicit MainWindow(QWidget *parent = 0);
    ~MainWindow();

private slots:
    void on_pushButton_clicked();

private:
    Ui::MainWindow *ui;
```

```
};


#endif // MAINWINDOW_H

#include "mainwindow.h"
#include "ui_mainwindow.h"
#include <QMessageBox>
#include <iostream>
#include <QIcon>
#include <QPixmap>
#include "Book.h"
#include "BookStore.h"


// Конструктор окна
MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow)
{

    ui->setupUi(this); // Настройка интерфейса
    setWindowTitle("Информация о книгах");
    QPixmap pix(":/book/image/book1.png");
    QPixmap pix2(":/book2/image/book2.jpg");
    setWindowIcon(QIcon(":/icon/image/icon.jpg"));

    ui->image1->setPixmap(pix.scaled(200, 200, Qt::KeepAspectRatio));



}

// Деструктор окна
MainWindow::~MainWindow()
{
    delete ui; // Удаляем интерфейс
}

// Слот для обработки нажатия на кнопку
void MainWindow::on_pushButton_clicked()
{
    // Получение данных из текстовых полей интерфейса
    QString title = ui->title->text();    // Название книги
    QString author = ui->author->text();  // Автор книги
    QString year = ui->year->text();      // Год издания
    QString price = ui->price->text();    // Цена книги

    // Проверка на пустые значения
    if (title.isEmpty() || author.isEmpty() || year.isEmpty() || price.isEmpty()) {
```

```
        QMessageBox::warning(this, "Ошибка", "Пожалуйста, заполните все поля."); // Сообщение об ошибке
        ui->statusBar->showMessage("ошибка в вводе значений");
        return; // Прерываем выполнение, если есть пустые поля
    }



    // Проверка: является ли год корректным целым числом
    bool isYearInt; // Переменная для хранения результата проверки
    int bookYear = year.toInt(&isYearInt); // Преобразуем строку в int
    if (!isYearInt || bookYear <= 0) { // Если год не число или меньше/равен нулю
        QMessageBox::warning(this, "Ошибка", "Введите корректный год (целое число больше нуля)."); // Сообщение об ошибке
        ui->statusBar->showMessage("ошибка в вводе значений");
        return; // Прерываем выполнение
    }


    // Проверка: является ли цена корректным числом с плавающей точкой
    bool isPriceDouble; // Переменная для хранения результата проверки
    double bookPrice = price.toDouble(&isPriceDouble); // Преобразуем строку в double
    if (!isPriceDouble || bookPrice <= 0.0) { // Если цена не число или меньше/равна нулю
        QMessageBox::warning(this, "Ошибка", "Введите корректную цену (положительное число)."); // Сообщение об ошибке
        ui->statusBar->showMessage("ошибка в вводе значений");
        return; // Прерываем выполнение
    }


    // Преобразование строк из Qt в стандартные строки C++ для передачи в BookStore
    std::string bookTitle = title.toStdString(); // Преобразуем QString в std::string
    std::string bookAuthor = author.toStdString(); // Преобразуем QString в std::string

    // Создание объекта BookStore с введёнными данными
    BookStore bookStore(bookTitle, bookAuthor, bookYear, bookPrice);

    // Преобразуем информацию из BookStore в QString и устанавливаем её на label
    ui->label->setText(QString::fromStdString(bookStore.displayBookStoreInfo()));

    ui->statusBar->showMessage("созван объект класс BookStore");
}

#include "mainwindow.h"
#include <QApplication>

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    MainWindow w;
    w.show();
```

```
    return a.exec();
}
```