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 91 ПЗ

|  |  |  |
| --- | --- | --- |
| Студент |  | А.П. Струнец |
| Руководитель |  | Е.Г. Макейчик |

Минск 2024**РЕФЕРАТ**

ПРОГРАММНОЕ СРЕДСТВО : курсовой проект / А. П. Струнец. – Минск : БГУИР, 2024, – п.з. – 59 с., чертежей (плакатов) – 2 л. формата А4.

Курсовой проект посвящен разработке программного обеспечения для расчета стоимости, основанного на принципах объектно-ориентированного программирования (ООП) с использованием фреймворка Qt. Основная цель проекта заключается в создании приложения, которое позволит пользователям вычислять разные данные об их поездке, с учетом параметров их пути и данных автомобиля.

В процессе проектирования были применены ключевые методы, включая создание классов на языке C++, реализацию сигналов и слотов для взаимодействия элементов интерфейса и разработку графического интерфейса с использованием Qt Designer. В проекте разработаны два класса: Trip, отвечающий за вычисление стоимости одного километра пути на основе расхода топлива на 100 километров пути и стоимости топлива за 1 литр, и LongTrip, который наследует функционал класса Trip и добавляет возможность расчета конечной цены поездки с учетом пройденного пути.

Результатом работы стало интерактивное приложение с интуитивно понятным графическим интерфейсом и системой проверки корректности ввода данных, что значительно повышает надежность расчетов. Проект продемонстрировал успешное применение принципов ООП, возможностей Qt Widgets и механизмов фреймворка для создания удобного и эффективного инструмента для финансового анализа.

Курсовой проект был проверен на антиплагиат в системе <https://www.antiplagiat.ru>, процент оригинальности текста составил 97%.

СОДЕРЖАНИЕ

[Введение 6](#_Toc112855518)

[1 Среда разработки программного средства 8](#_Toc112855519)

[1.1 Фреймворк Qt 8](#_Toc112855520)

[1.2 Язык программирования C++ 11](#_Toc112855521)

[1.3 Установка фреймворка Qt 16](#_Toc112855522)

[2 Разработка графического интерфейса программного средства 24](#_Toc112855523)

[2.1 Графический пользовательский интерфейс (GUI) 24](#_Toc112855524)

[2.2 Qt Designer 25](#_Toc112855525)

[2.3 Создание графического пользовательского интерфейса 28](#_Toc112855526)

[3 Разработка программного средства 33](#_Toc112855527)

[3.1 Unified Modeling Language (UML) 33](#_Toc112855528)

[3.2 Классы разработанного программного средства 36](#_Toc112855529)

[3.3 Директивы 38](#_Toc112855530)

[3.4 Механизм сигналов и слотов в Qt 41](#_Toc112855531)

[Заключение 48](#_Toc112855532)

[Список используемых источников 49](#_Toc112855533)

[Приложение А (обязательное) UML-диаграмма классов программного средства 50](#_Toc112855534)

[Приложение Б (обязательное) Блок-схема алгоритма работы программного средства 52](#_Toc112855535)

[Приложение В (обязательное) Листинг кода программного средства 54](#_Toc112855536)

# ВВЕДЕНИЕ

ООП – одна из возможных методологий программирования. В отличие от процедурного программирования, которому и пришло на смену объектно-ориентированное, при написании кода уделяется больше внимания организации программы и структурированию кода, нежели его логике и вопросам написания. Главным понятием в ООП является объект. Объект содержит в себе данные и функции для их обработки. В свою очередь объект является экземпляром класса.

Класс может быть представлен как шаблон, который определяет общие характеристики и поведение объектов. Он вариативен и позволяет создавать множество объектов, обладающих схожими свойствами, но при этом каждое из этих сущностей может иметь свои уникальные значения. Важным аспектом ООП является инкапсуляция, которая позволяет скрывать внутренние состояния объектов и взаимодействовать с ними только через аккуратно определённые интерфейсы. Это способствует уменьшению зависимости между компонентами системы и улучшает ее модульность.

Другим ключевым понятием является наследование, которое позволяет создавать новые классы на основе уже существующих, заимствуя их свойства и методы. Это не только ускоряет процесс разработки, но и способствует повторному использованию кода. Полиморфизм же позволяет объектам разных классов обрабатываться через один и тот же интерфейс, что упрощает архитектуру программы и делает её более гибкой.

Так же абстракция в объектно-ориентированном программировании является принципом, позволяющим разработчикам концентрироваться на сущностях и их взаимодействиях, не отвлекаясь на детали реализации. Этот принцип позволяет выделить ключевые характеристики объектов и определить лишь те аспекты, которые имеют значение в контексте конкретной задачи. Использование абстракции способствует улучшению читаемости и поддерживаемости кода. Она позволяет разработчикам создавать интерфейсы, которые описывают поведение объектов, не привязывая их к конкретным реализациям.

Таким образом, объектно-ориентированное программирование представляет собой мощную методологию, способствующую созданию сложных и масштабируемых программных систем, которые легче поддерживать и развивать в долгосрочной перспективе.

Целью курсового проекта является разработка программного средства с использованием методологии ООП.

Задачи курсового проекта:

1 Изучение принципов работы ООП.

2 Установка Qt фреймворка и его освоение.

3 Изучение языка программирования С++ в контексте парадигмы ООП.

4 Приобретение навыков разработки GUI приложения.

5 Овладение опытом разработки UML-диаграммы классов, блок-схемы алгоритма.

6 Разработка программного средства.

# 1 СРЕДА РАЗРАБОТКИ ПРОГРАММНОГО СРЕДСТВА

## 1.1 Фреймворк Qt

Qt – это мощный фреймворк, который позволяет разработчикам создавать кроссплатформенные приложения на языке C++. Он обеспечивает возможность запуска созданных программ на большинстве современных операционных систем без необходимости изменения исходного кода; достаточно просто скомпилировать приложение для каждой платформы.

Разработка фреймворка Qt началась в 1991 году в компании Trolltech, и первая версия для публичного использования была выпущена 20 мая 1995 года. В настоящее время фреймворк развивается компанией Qt, и его последняя версия – Qt 6. Она была представлена в декабре 2020 года. С тех пор регулярно выходят обновления и подверсии, приносящие значительные улучшения и новые функции.

Qt предлагает двойную лицензию: открытую и коммерческую. Это позволяет разработчикам использовать бесплатную версию фреймворка, а при необходимости приобретать коммерческую лицензию для получения дополнительных возможностей и профессиональной поддержки. Благодаря тому, что Qt является проектом с открытым исходным кодом, любой желающий может получить доступ к исходному коду фреймворка и скомпилировать его самостоятельно.

Кроме того, активное сообщество разработчиков и пользователей вносит свой вклад в улучшение и развитие Qt, что способствует его постоянному обновлению и адаптации к современным требованиям. Фреймворк также поддерживает интеграцию с различными инструментами и библиотеками, что позволяет разработчикам создавать более сложные и мощные приложения. Qt продолжает оставаться одним из лидеров в области кроссплатформенной разработки, обеспечивая широкий спектр возможностей для реализации инновационных идей.

Фреймворк включает обширный набор классов, необходимых для разработки прикладного программного обеспечения, охватывающий все — от графических элементов интерфейса до классов для работы с сетевыми протоколами и базами данных. Qt полностью ориентирован на объектно-ориентированное программирование, что обеспечивает гибкость и удобство в разработке. Кроме того, он поддерживает компонентное программирование, позволяя создавать модульные и легко расширяемые приложения, адаптированные к конкретным задачам.

Популярность Qt обусловлена его мощью и простотой использования, что делает его одним из наиболее востребованных инструментов для разработки кроссплатформенного ПО. Фреймворк применяется в различных областях – от мобильных и настольных приложений до игр и промышленных систем. Используя Qt, разработчики могут создавать высококачественные и производительные приложения, экономя время на адаптацию кода для разных платформ.

Qt также обладает активным сообществом и обширной документацией, что упрощает процесс обучения и помогает находить решения возникающих проблем. Поддержка современных технологий, таких как облачные вычисления и интеграция с аппаратным обеспечением, расширяет возможности разработки и позволяет создавать инновационные решения.

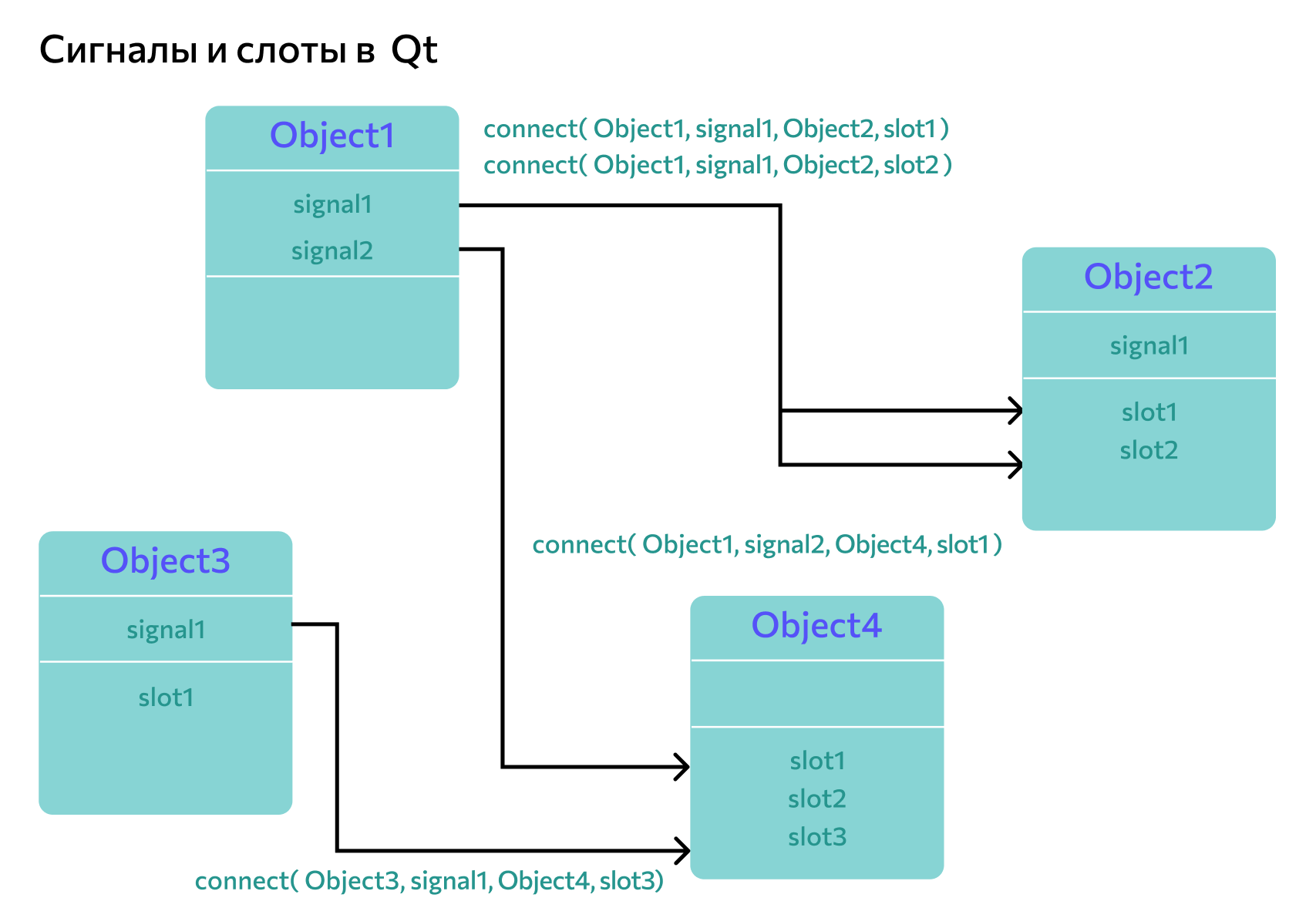
Еще одним важным аспектом является система сигналов и слотов, которая упрощает взаимодействие между объектами и делает реализацию событийного программирования более доступной. Это особенно полезно при создании интерактивных пользовательских интерфейсов (см. рисунок 1.1).

Рисунок 1.1 – Система сигналов и слотов в Qt

Кроме того, Qt предлагает мощные инструменты для разработки приложений с высоким уровнем графики, включая поддержку OpenGL и возможности 2D и 3D графики, что позволяет создавать визуально привлекательные приложения и игры.

Qt также поддерживает международные стандарты, что упрощает локализацию приложений для различных языков и культур, что особенно важно для разработчиков, ориентированных на глобальный рынок.

Инфраструктура Qt включает в себя полный набор интуитивно понятных модулей, классов и библиотек C++, использующих API для упрощения разработки приложений. С помощью этого фреймворка было создано множество популярных программ, таких как Kaspersky Internet Security, Skype, Opera, Telegram и многих других.

Архитектура Qt представляет собой мощную и гибкую платформу, предназначенную для разработки кроссплатформенных приложений с графическим пользовательским интерфейсом. Основой Qt является модульная структура, позволяющая разработчикам выбирать только необходимые компоненты и модули, что способствует оптимизации производительности и снижению потребления ресурсов.

Ключевые элементы архитектуры включают фреймворки для работы с графикой, сетевыми протоколами, базами данных и многопоточными процессами. Фокус на простоте интеграции различных модулей позволяет создавать сложные системы, не жертвуя при этом гибкостью и скоростью разработки.

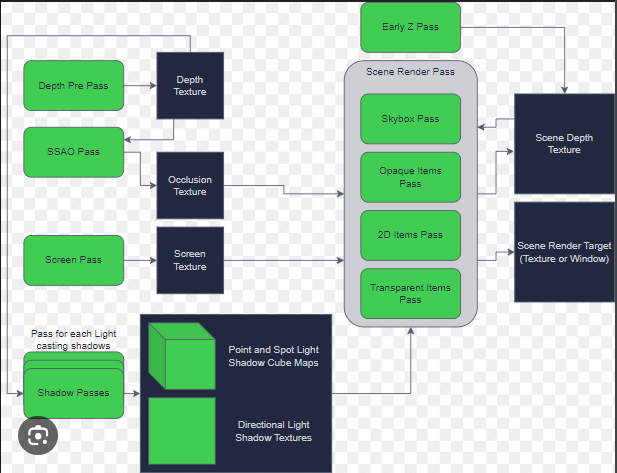
Кроме того, Qt применяет язык разметки QML, который предоставляет мощные инструменты для создания интуитивно понятного интерфейса. В сочетании с C++, QML позволяет быстро разрабатывать динамичные и отзывчивые приложения (см. рисунок 1.2).

Рисунок 1.2 – Архитектура Qt

Таким образом, Qt предоставляет разработчикам мощные инструменты для создания современных приложений с богатым интерфейсом и высокой производительностью. Этот фреймворк идеально подходит для реализации амбициозных проектов и достижения выдающихся результатов в разработке программного обеспечения, открывая новые возможности для инновационных идей и креативного подхода в создании ПО.

## 1.2 Язык программирования C++

C++ – один из наиболее востребованных языков программирования, активно применяемый для разработки разнообразных программ, драйверов, приложений, операционных систем и видеоигр. Это надстройка над языком C, обеспечивающая совместимость с уже существующим кодом и библиотеками.

История C++ началась в начале 1980-х годов, когда Бьёрн Строуструп, работая в Bell Labs, создал язык, который сочетал бы мощь C с концепциями объектно-ориентированного программирования. Первоначально названный "C with Classes", он стал расширением C с поддержкой классов и объектов. В 1985 году Строуструп выпустил первую книгу о C++, что значительно способствовало его популяризации.

Одной из главных особенностей C++ является объектно-ориентированное программирование (ООП). Этот подход позволяет разработчикам создавать более структурированные и модульные приложения, что упрощает процесс разработки и дальнейшего сопровождения кода. Принципы инкапсуляции, наследования и полиморфизма, внедрённые в язык, способствуют созданию гибких и масштабируемых решений. ООП помогает упростить управление сложностью больших проектов, разбивая их на более мелкие, управляемые части.

Рассмотрим С++ в ООП. Центральным понятием ООП является класс. Класс представляет собой описание, схему, модель реально существующего объекта. Класс является типом данных, определяемым пользователем. В результате объявления класса создается новый тип данных, в котором объединены элементы: поля и методы, обрабатывающие эти данные. Поле класса – это элемент класса, описывающий данные. Метод класса – это процедура или функция, включенная в описание класса. Метод класса вызывается конкретным экземпляром класса и привязан к описанию и структуре класса. При этом каждый экземпляр класса может иметь свои уникальные значения полей, но будет использовать общие методы, определенные на уровне класса. Это позволяет создавать эффективные и легко управляемые структуры данных, а также поддерживать модульность и повторное использование кода. (см. рисунок 1.3).

Рисунок 1.3 – Класс, поле и метод класса в C++

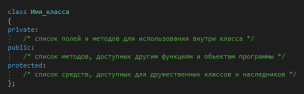
Поля и методы класса могут быть закрытыми или открытыми. К закрытым полям и методам класса можно обращаться только внутри класса. Открытые элементы класса доступны за его пределами. Как правило, открытая часть кода обеспечивает управляемое взаимодействие (интерфейс) с закрытыми элементами объекта. Управляют доступностью элементов класса модификаторы доступа private, public и ptotected, после которых следует знак двоеточия. В общем виде описание класса выглядит следующим образом (см. рисунок 1.4).

Рисунок 1.4 – Модификаторы доступа в C++

В С++ существуют два способа определить переменную. Можно сначала объявить переменную, а затем присвоить ей значение (см. рисунок 1.5).



Рисунок 1.5 – Способ объявления переменной с последующей инициализацией

Или, определив переменную, сразу же инициализировать ее (см. рисунок 1.6).

Рисунок 1.6 – Способ объявление переменной, сразу с инициализацией

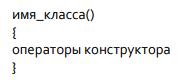
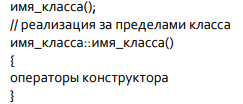
 Конструктор – это метод класса, который служит для инициализации объекта при его создании. Имя конструктора совпадает с именем самого класса. При необходимости конструктор может получать параметры, но не может возвращать значения. Он автоматически вызывается при создании объекта, гарантируя, что все поля объекта будут правильно инициализированы. Если конструктор не определен явно, то компилятор создаст конструктор по умолчанию, который инициализирует поля значениями по умолчанию. Важно, что конструктор может быть перегружен, то есть в одном классе могут быть несколько конструкторов с разными параметрами. (см. рисунок 1.7).

Рисунок 1.7 – Пример конструктора в классе

Если реализация конструктора определяется за пределами класса, то в классе указывается прототип конструктора в следующем формате (см. рисунок 1.8).

Рисунок 1.8 – Пример конструктора вне класса

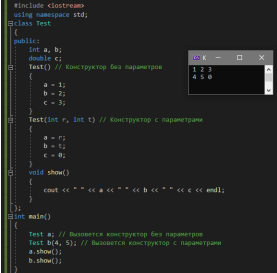
Так же существует параметрический конструктор (см. рисунок 1.9).

Рисунок 1.9 – Пример параметрического конструктора

Деструктор – это специальный метод класса в языке C++, который выполняет удаление объектов. Деструктор удаляет из памяти отработавшие объекты и освобождает выделенную для них память (см. рисунок 1.10).

Рисунок 1.10 – Пример деструктора

Существует множество библиотек и фреймворков, таких как Qt и Boost, которые значительно облегчают разработку, предлагая готовые решения для часто возникающих задач. Boost охватывает широкий спектр задач, от алгоритмов до сетевого взаимодействия. Эти инструменты позволяют разработчикам сосредоточиться на логике приложения, не отвлекаясь на низкоуровневые детали.

Несмотря на свою сложность и высокую кривую обучения, C++ остаётся актуальным благодаря своей производительности и контролю над ресурсами. Это делает его идеальным выбором для разработки программного обеспечения в таких сферах, как видеоигры, встраиваемые системы и научные вычисления, где критически важны скорость выполнения и эффективность использования ресурсов. C++ также широко используется для создания операционных систем, системного программного обеспечения и приложений для обработки больших данных.

Язык активно применяется в системах реального времени, высокопроизводительных вычислениях и встраиваемых системах, требующих максимально оптимизированного кода. Его высокая производительность делает C++ основным языком для разработки многих популярных игр и игровых движков, таких как Unreal Engine и Unity, подтверждая его значимость в индустрии игр. Также многие финансовые приложения и платформы для высокочастотной торговли используют C++, благодаря его способности обрабатывать большие объёмы данных с минимальными задержками.

Каждый год появляются новые стандарты C++, которые способствуют его эволюции и расширению возможностей. Последние версии, такие как C++11, C++14, C++17 и C++20, внедрили множество нововведений, включая лямбда-выражения, умные указатели, параллельное программирование и улучшенные контейнеры. Эти обновления делают язык более современным и удобным для разработчиков, позволяя создавать более безопасный и эффективный код.

С ростом популярности облачных технологий и распределённых систем C++ продолжает адаптироваться, предлагая новые средства для разработки приложений, способных работать в таких условиях. Таким образом, C++ остаётся актуальным и продолжает развиваться, занимая важное место в мире программирования.

От C++ также произошли несколько языков, унаследовавших его концепции и синтаксис, но предлагающих свои уникальные возможности. Например, C#, разработанный Microsoft, в значительной степени вдохновлён C++ и ориентирован на разработку приложений для платформы .NET. Он включает элементы объектно-ориентированного программирования, акцентируя внимание на простоте использования и безопасности.

Objective-C, использующий синтаксис C, поддерживает объектно-ориентированное программирование и добавляет возможности динамического разрешения сообщений. Этот язык был основным для разработки приложений на платформе Apple до появления Swift.

Язык D сочетает в себе эффективность и производительность с современными языковыми конструкциями, такими как сборка мусора и встроенные асинхронные возможности.

Rust, возникший из C и C++, предлагает строгие гарантии безопасности и управления памятью, что делает его привлекательным для системного программирования и высокопроизводительных приложений.

Таким образом, C++ продолжает занимать лидирующие позиции и вдохновлять на инновации в мире программирования, сочетая мощные инструменты для создания высокопроизводительных приложений и гибкость для решения разнообразных задач. Это делает его универсальным языком для разработчиков в самых разных областях, от системного программирования до разработки игр и научных приложений, позволяя ему оставаться важным игроком на современном рынке программного обеспечения.

## 1.3 Установка фреймворка Qt

Для загрузки фреймворка Qt необходимо перейти на сайт qt.io. Выбрать вкладку "Download. Try." (см. рисунок 1.11).

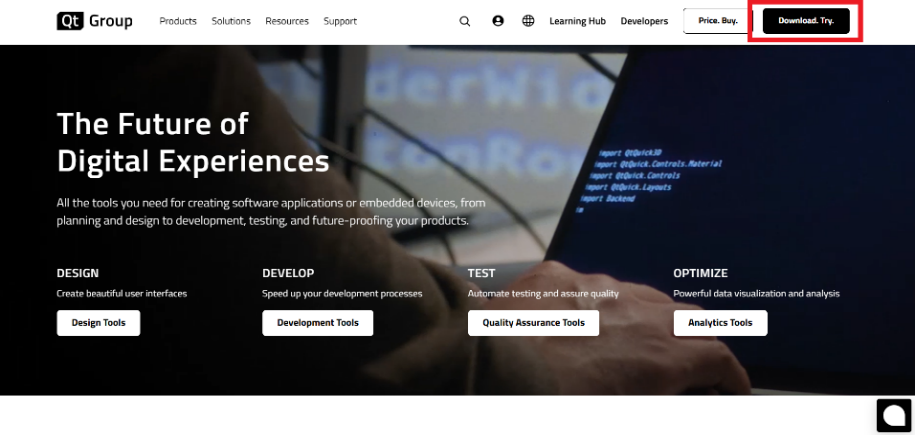
****

Рисунок 1.11 – Страница установки Qt

Перейти в раздел " Explore Qt Community Edition" (см. рисунок 1.12).

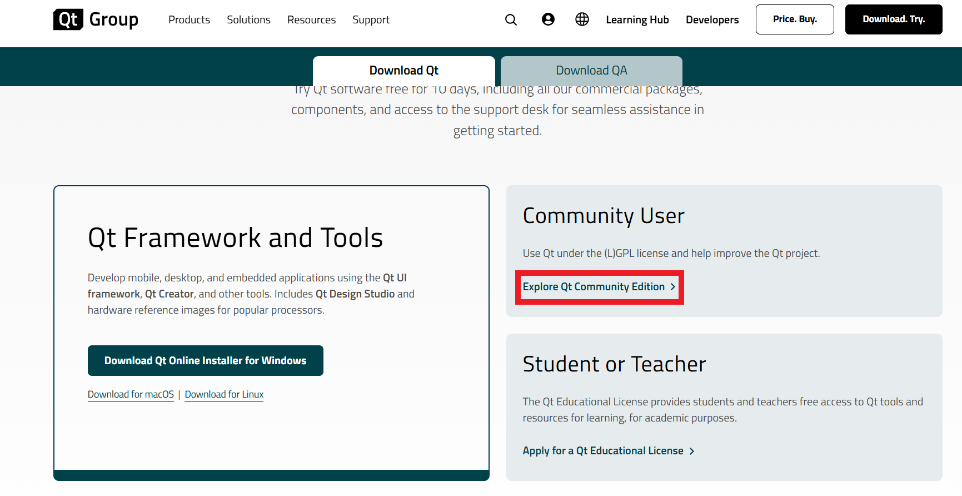


Рисунок 1.12 – Раздел " Explore Qt Community Edition "

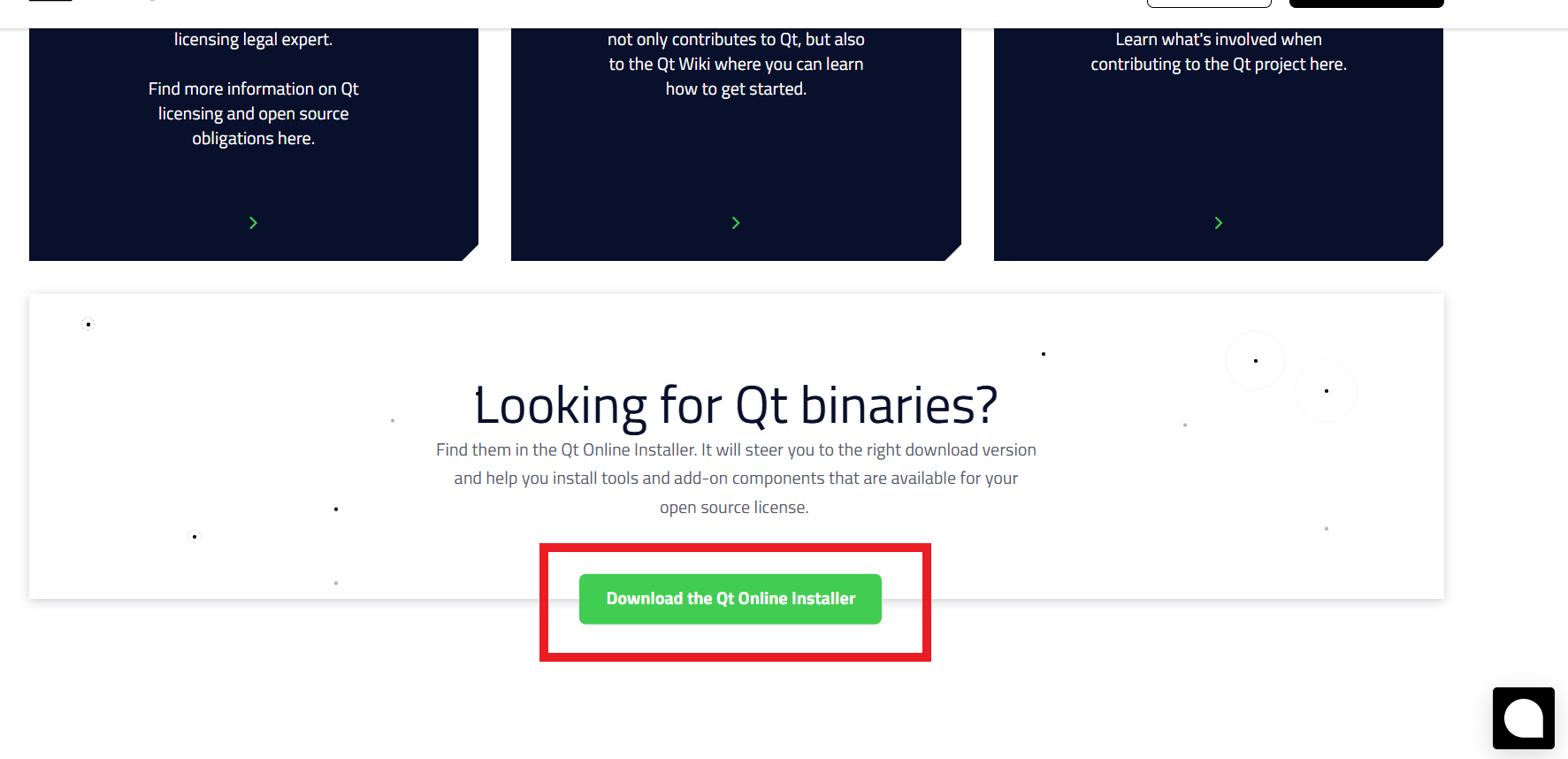
Далее нажать на кнопку "Download the Qt Online Installer" (см. рисунок 1.13).

Рисунок 1.13 – Кнопка "Download the Qt Online Installer"

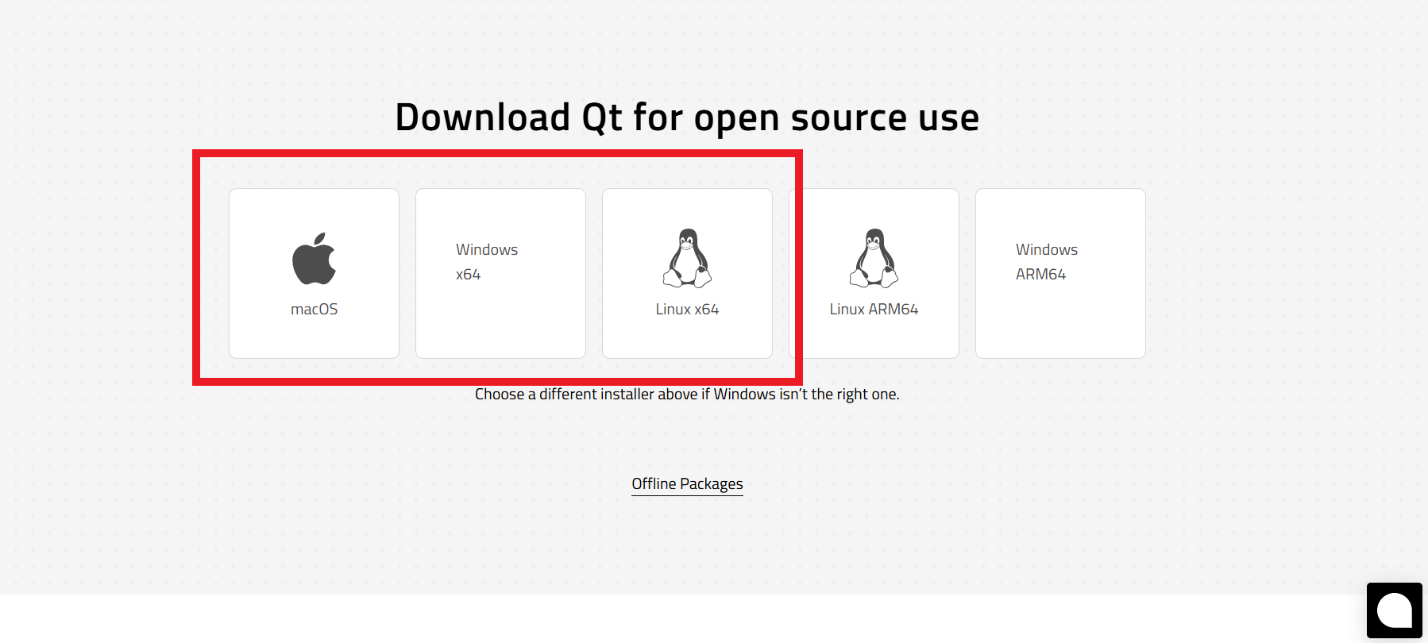
Скачать нужную вам версию под вашу операционную систему (см. рисунок 1.14).

Рисунок 1.14 – Выбор файла скачивания для разных операционных систем

После успешного скачивания загрузочного файла в данном окне необходимо произвести регистрацию в Qt Account или войти в него. Для этого нужно ввести почту и пароль. После ввода всех данных нужно нажать кнопку "Далее"(см. рисунок 1.15).

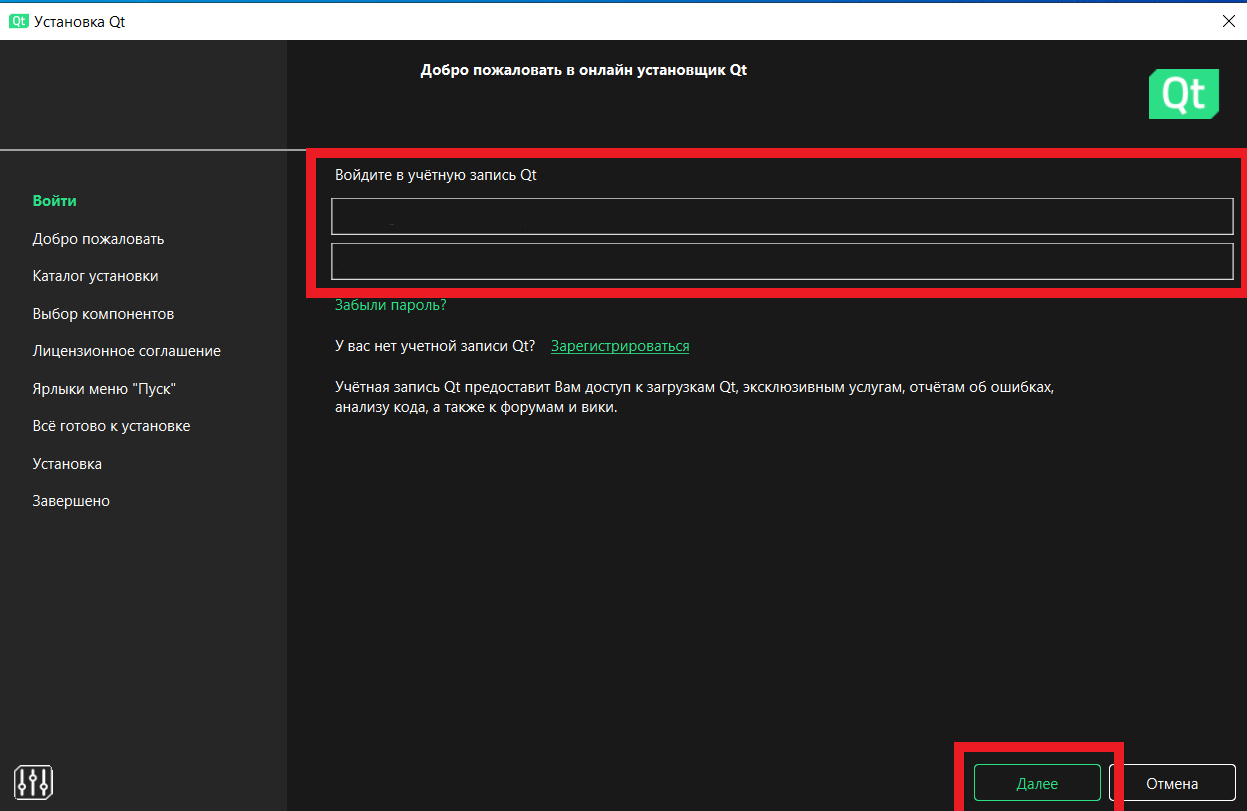


Рисунок 1.15 – Форма регистрации

В следующем окне необходимо согласиться с правилами и условиями использования Qt и нажать кнопку "Далее" (см. рисунок 1.16).

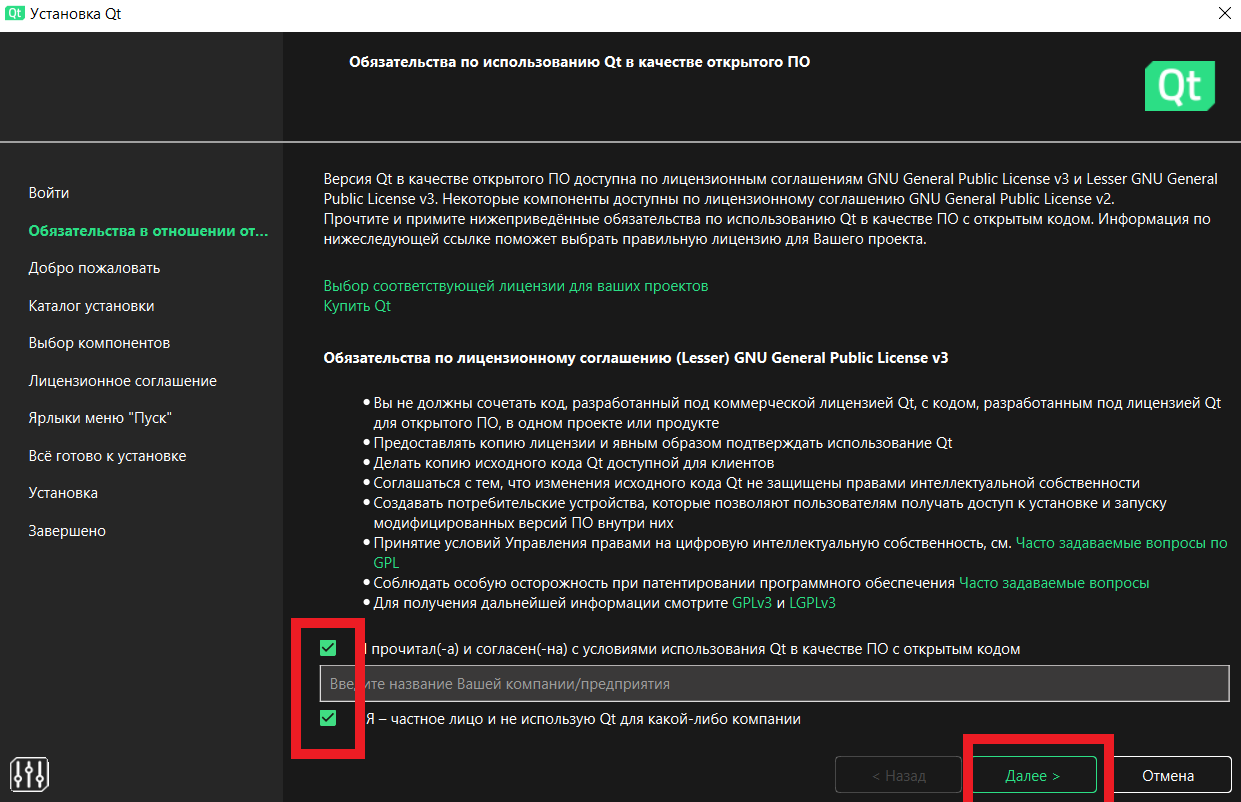


Рисунок 1.16 – Файл-установщик Qt Creator

В следующем окне "Добро пожаловать" нажать кнопку "Далее" (см. рисунок 1.17).

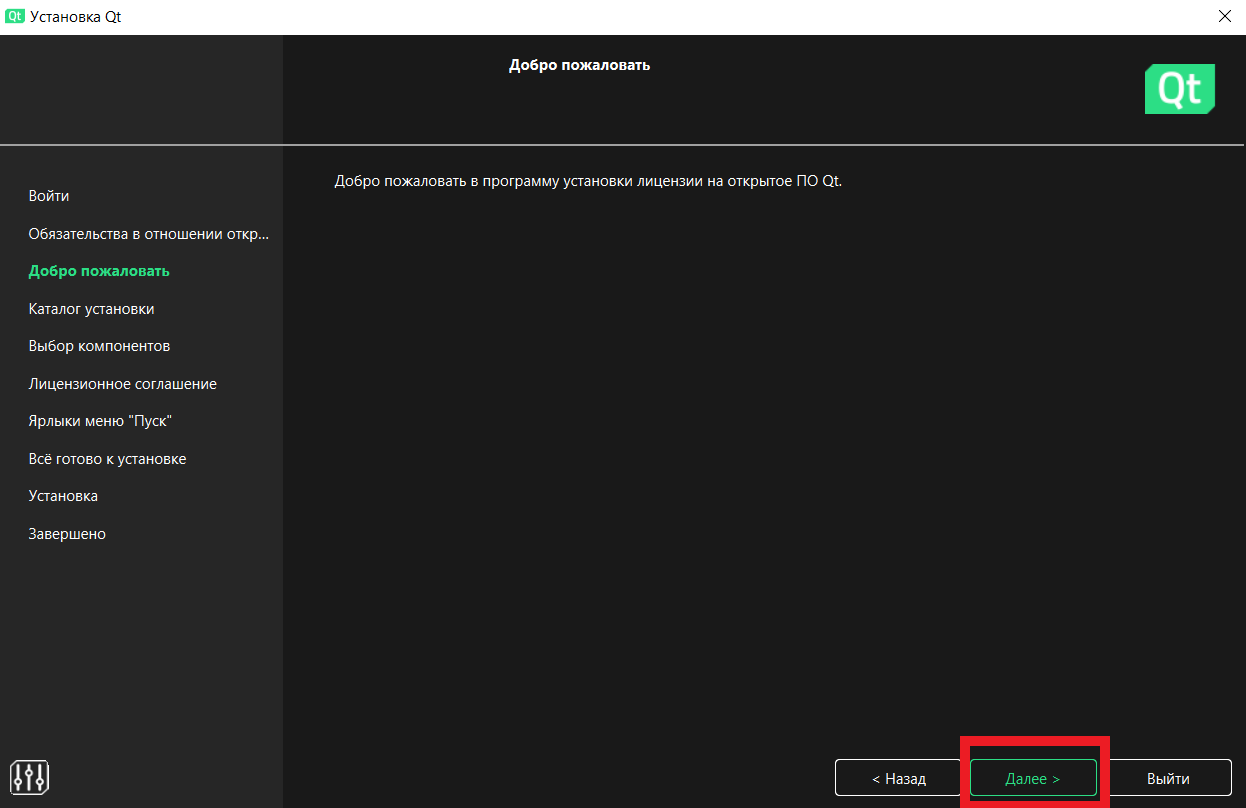


Рисунок 1.17 – Окно установщика, вкладка "Добро пожаловать"

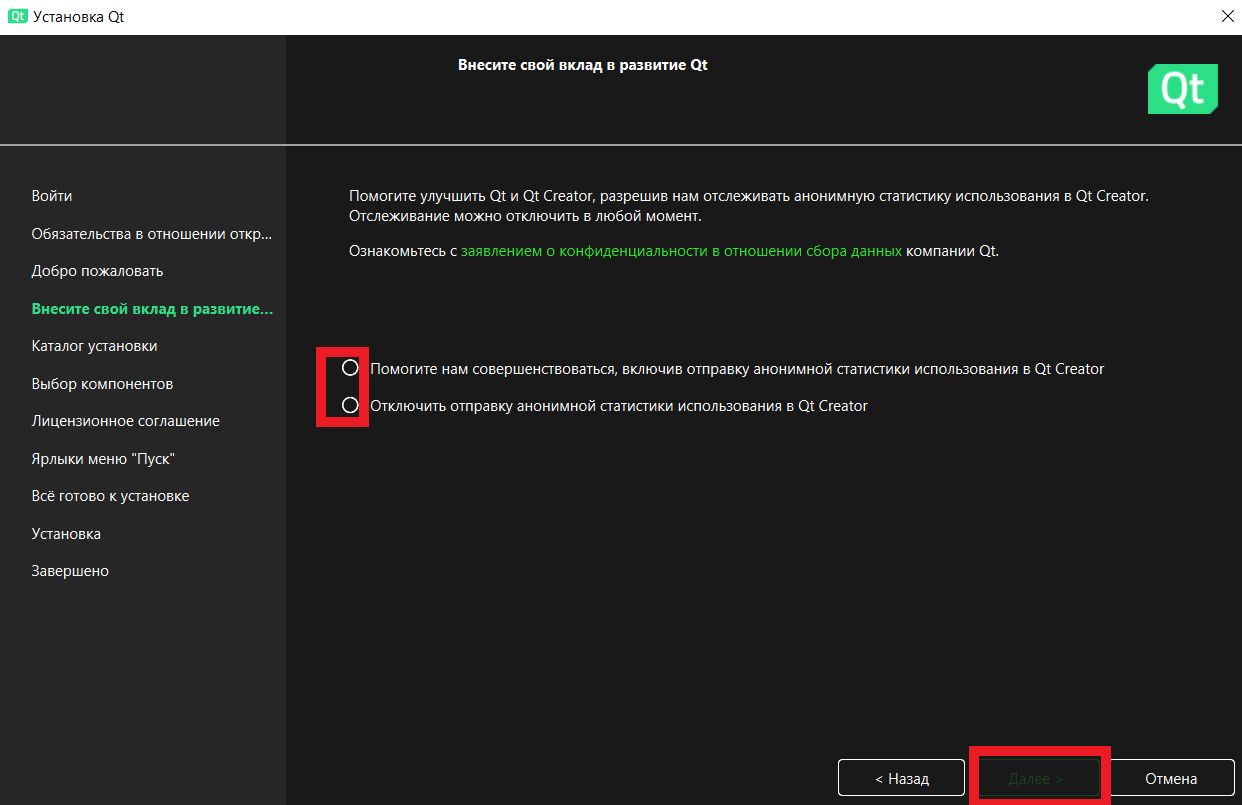
В следующем окне "Внесите свой вклад в развитие Qt" по желанию выбрать пункты и нажать на кнопку "Далее" (см. рисунок 1.18).

Рисунок 1.18 – Окно установщика, вкладка "Внести свой вклад в развитие Qt"

В данном окне представлен каталог установки. Надо указать, куда будет устанавливаться Qt и нажать кнопку "Далее" (см. рисунок 1.19).

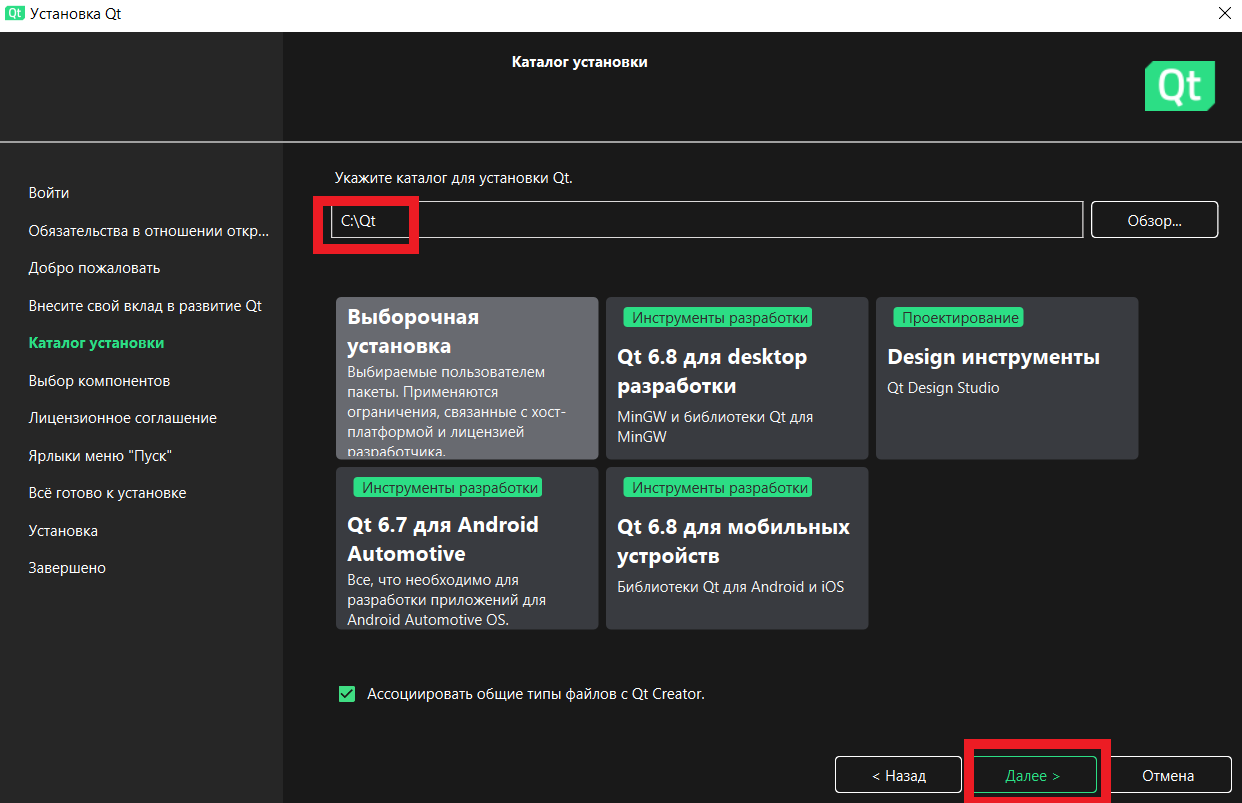


Рисунок 1.19 – Окно установщика, вкладка "Каталог установки"

В следующем окне необходимо выбрать пакеты с нужными компиляторами и нажать кнопку "Далее" (см. рисунок 1.20).

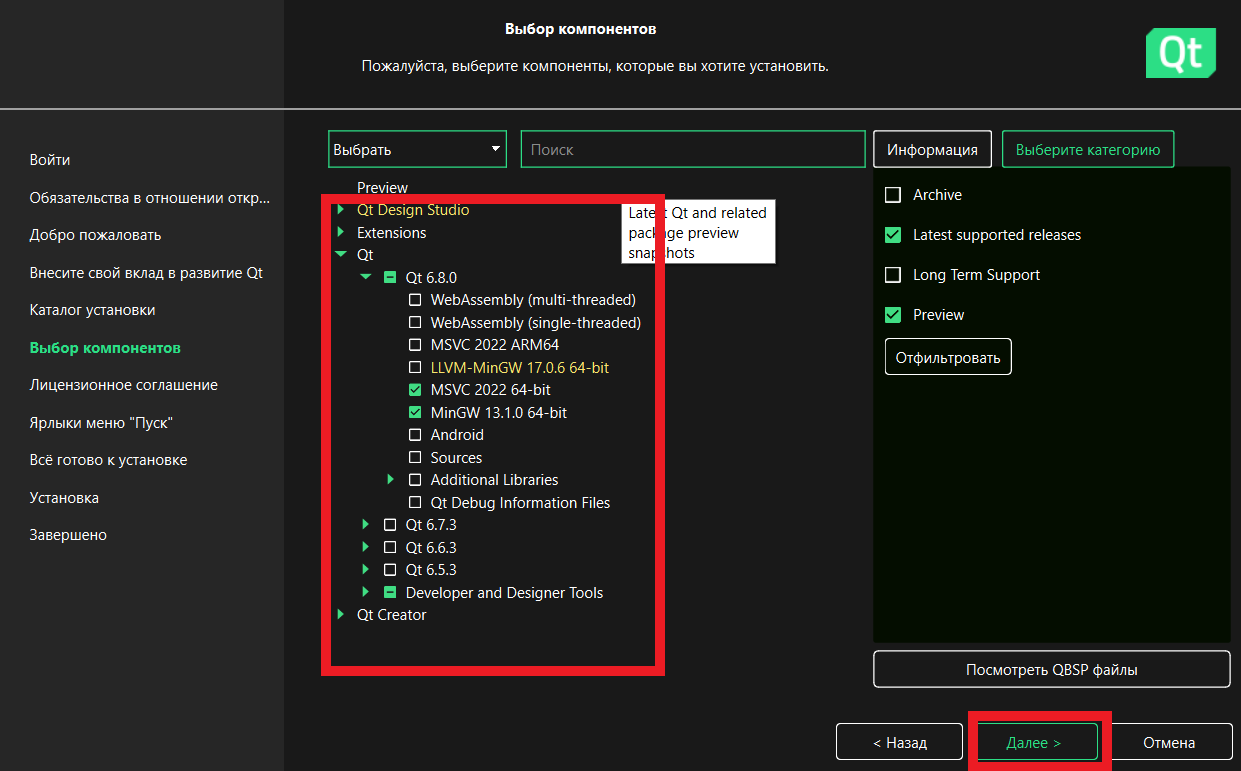


Рисунок 1.20 – Окно установщика, вкладка "Выбор компонентов"

MSVC – компилятор под Visual Studio. Пакеты данного компилятора нужно выбирать исходя из версии приложения Visual Studio и битности операционной системы.

MinGW – компилятор GNU/C++ для Windows, используется всеми сторонними средствами разработки.

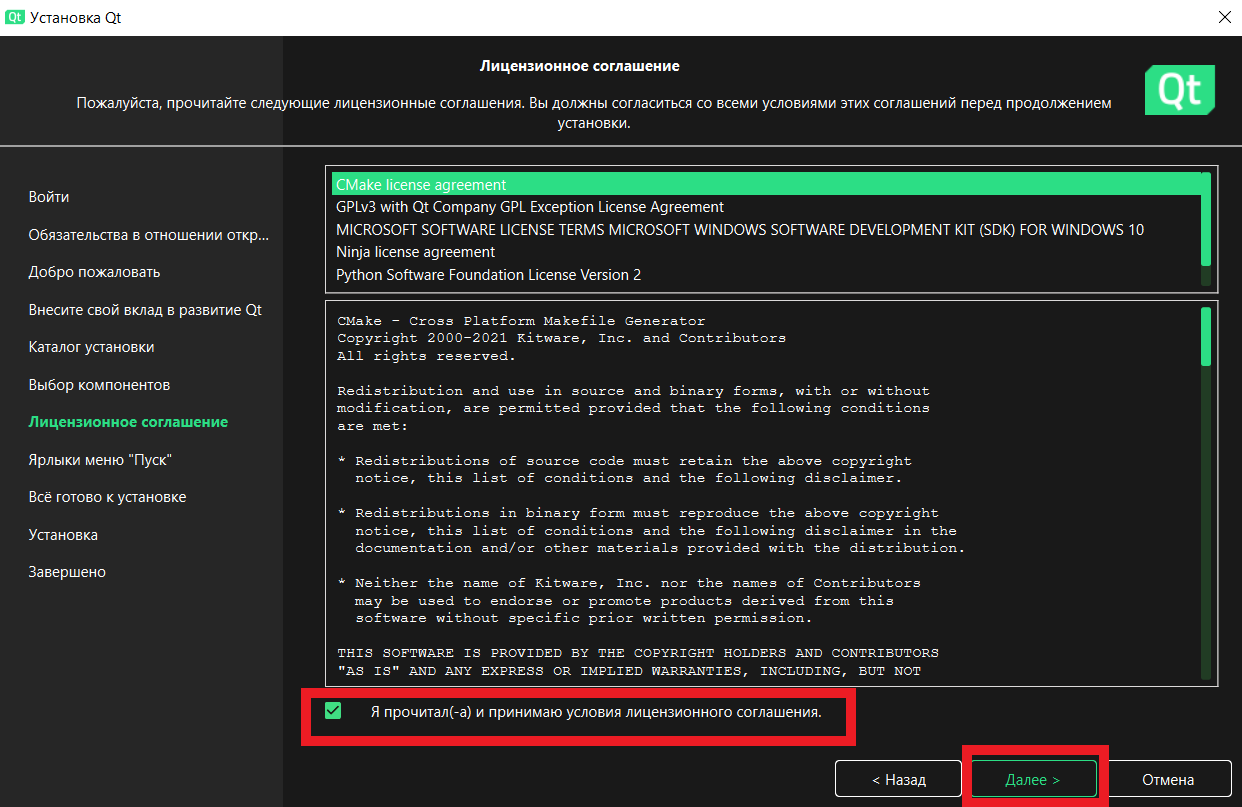
В следующем окне представлено лицензионное соглашение. Нужно подтвердить, что согласны с условиями данного соглашения. Необходимо поставить галочку и нажать кнопку "Далее" (см. рисунок 1.18).

Рисунок 1.20 – Окно установщика, вкладка "Лицензионное соглашение"

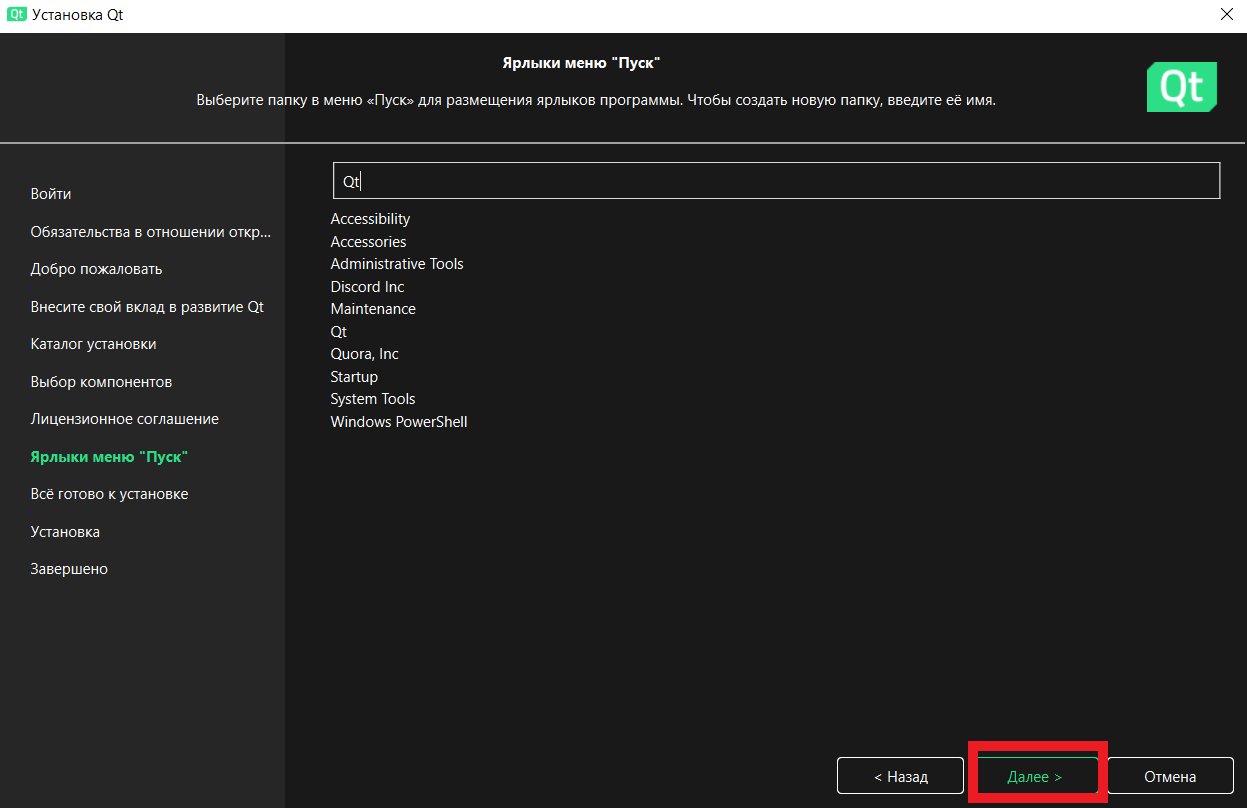
В данном окне необходимо выбрать папку, для размещения ярлыков программы и нажать кнопку "Далее" (см. рисунок 1.21).

Рисунок 1.21 – Окно установщика, вкладка "Ярлыки меню Пуск"

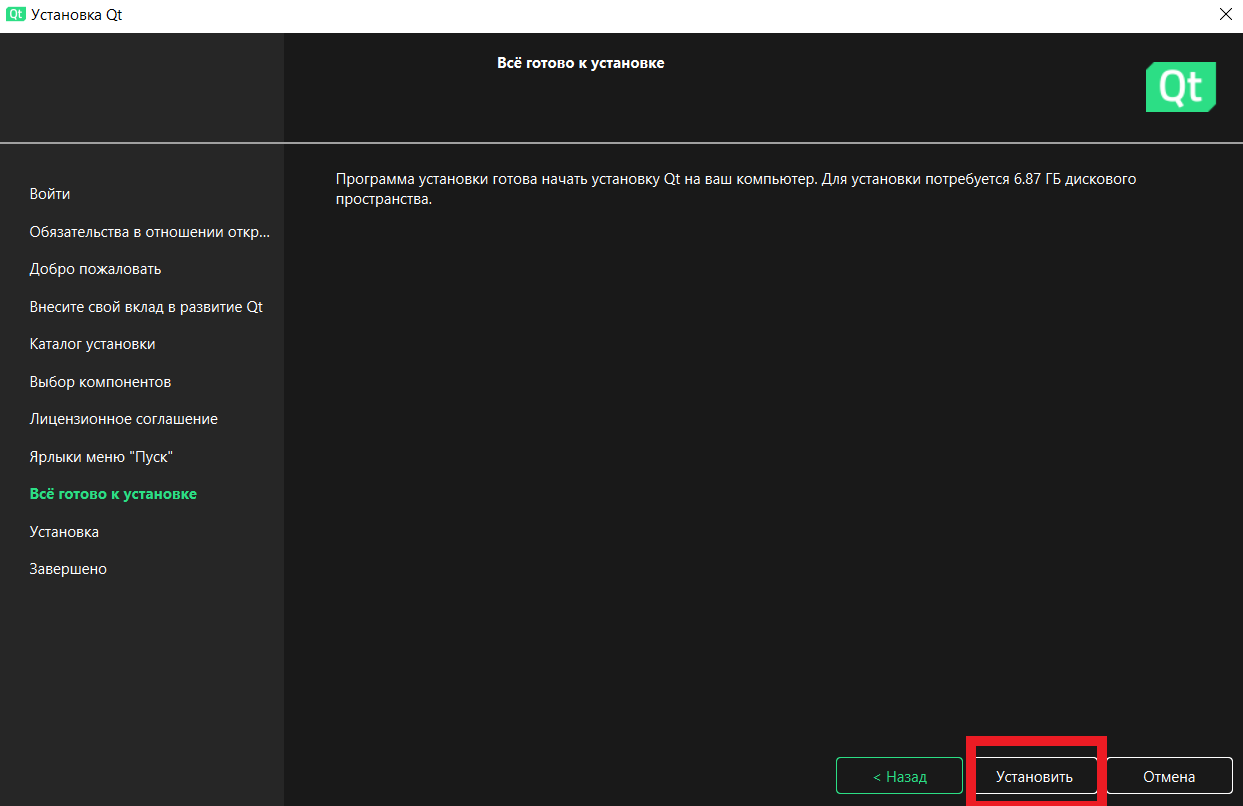
В следующем окне файл готов к установке. Нужно нажать кнопку "Установить" (см. рисунок 1.22).

Рисунок 1.22 – Окно установщика, вкладка "Всё готово к установке"

В следующем окне "Установка Qt" представлена сама установка файлов Qt. Есть возможность её остановить при возникновении такой потребности, для этого нужно нажать на кнопку "Отмена" (см. рисунок 1.23).

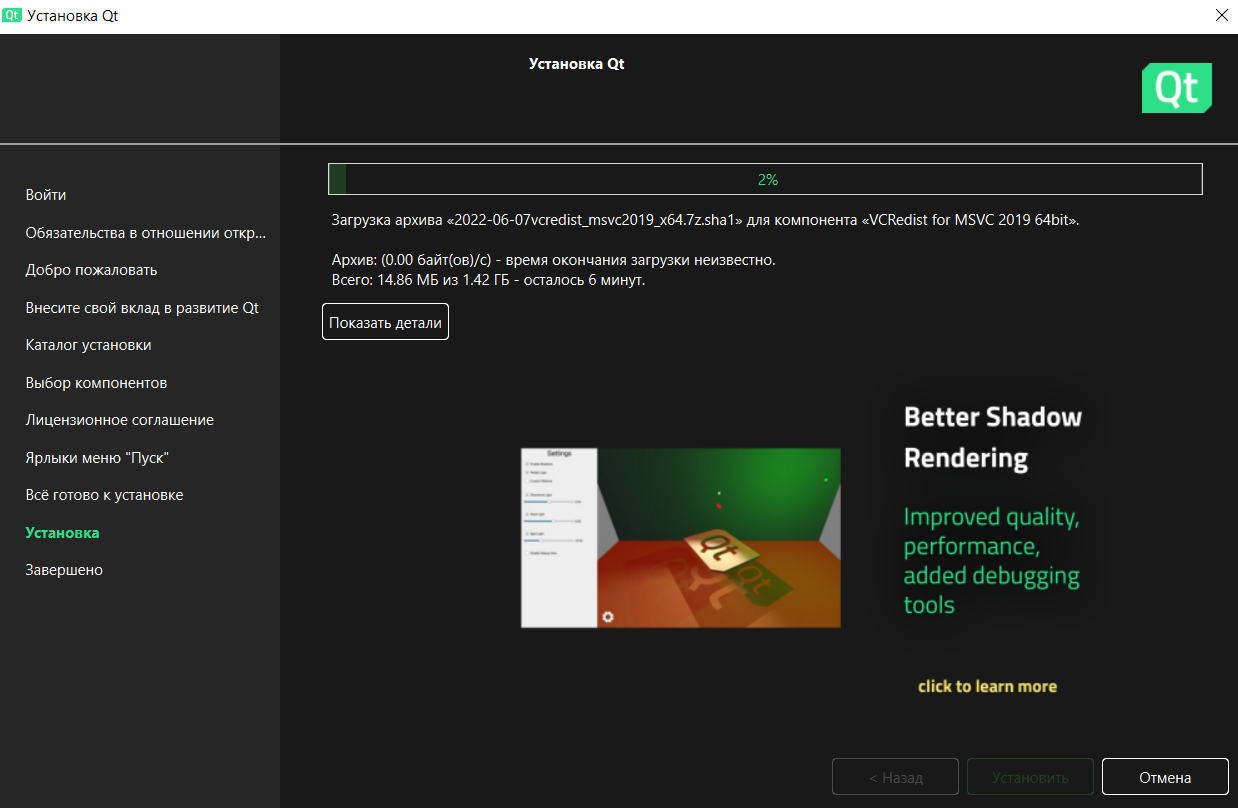


Рисунок 1.23 – Окно установщика, вкладка "Установка Qt"

Далее в окне "Установка Qt" будет представлена информация об успешной установке Qt. Нужно нажать кнопку "Завершить" (см. рисунок 1.24).

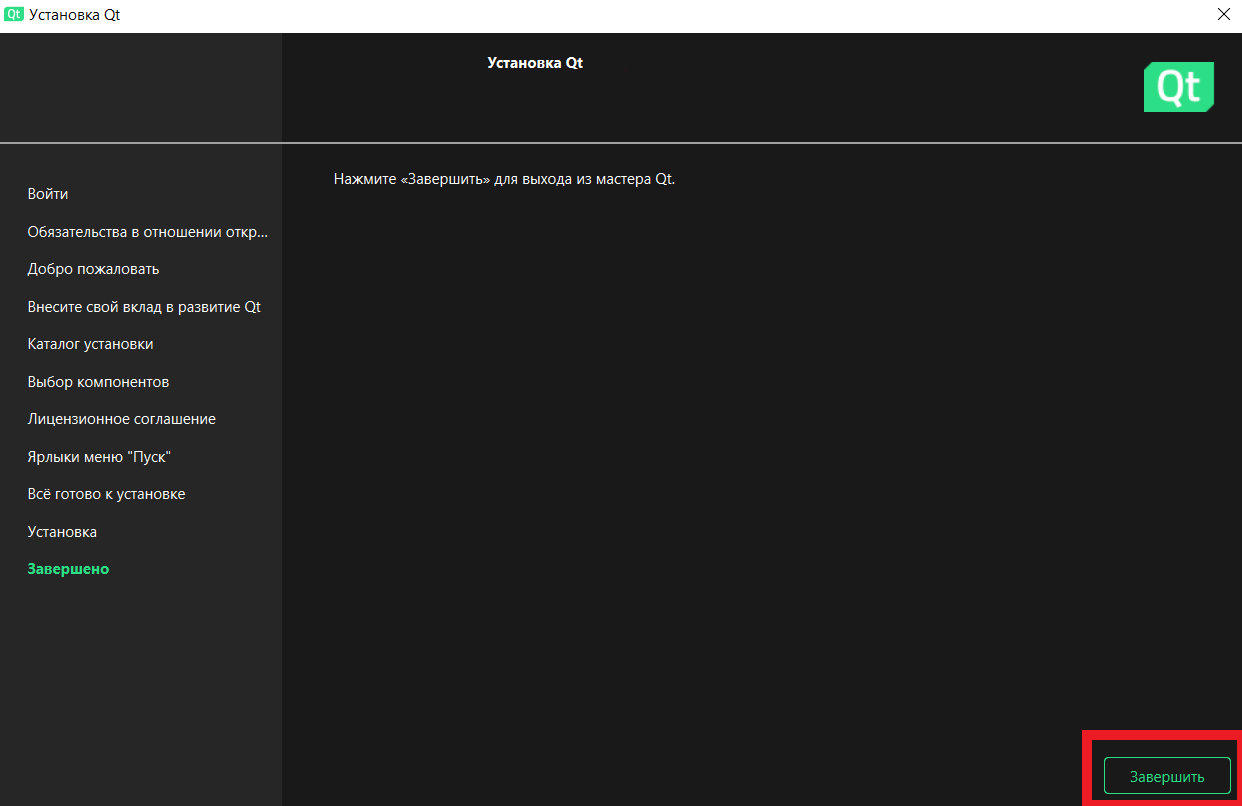


Рисунок 1.24 – Окно установщика, вкладка "Установка Qt"

Таким образом, вы произвели процесс скачивания и установки фреймворка Qt, который является ключевым шагом для начала работы с этим мощным инструментом. После успешной установки фреймворка Qt разработчик получает доступ ко всем функциональным возможностям, таким как редактор кода, отладчик, инструменты для работы с графическими интерфейсами (Qt Designer) и функции для кроссплатформенной разработки.

# 2 РАЗРАБОТКА ГРАФИЧЕСКОГО ИНТЕРФЕЙСА  ПРОГРАММНОГО СРЕДСТВА

## 2.1 Графический пользовательский интерфейс (GUI)

Графический пользовательский интерфейс (GUI) представляет собой вид интерфейса, в котором элементы, такие как меню, кнопки, значки и списки, отображаются в визуальной форме. Основное преимущество GUI заключается в том, что он обеспечивает пользователям интуитивное и визуальное взаимодействие с программами и системами, значительно обогащая их опыт (см. рисунок 2.1).

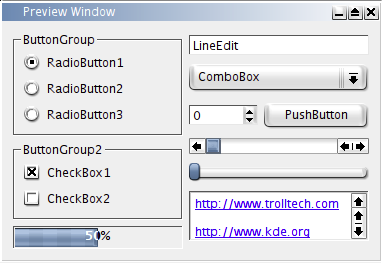


Рисунок 2.1 – Пример базового графического пользовательского интерфейса

В современных графических интерфейсах часто используются метафоры из реального мира, что упрощает понимание и освоение системы. Например, значок папки символизирует хранение данных, а кнопка "Домой" возвращает пользователя к начальной точке навигации. Более того, GUI предлагает возможности для персонализации, позволяя каждому пользователю адаптировать интерфейс под свои нужды и предпочтения.

Разработка графического интерфейса требует внимательного сочетания эстетики и функциональности. Дизайнеры стремятся создать не только визуально привлекательный, но и удобный интерфейс, учитывая такие аспекты, как цветовая гамма, типографика, компоновка и доступность. Важно соблюдать принципы юзабилити, которые обеспечивают комфортное взаимодействие пользователей с интерфейсом. Эффективный GUI создает положительное впечатление и улучшает общее восприятие программы, что, в конечном счете, ведет к повышению удовлетворенности пользователей. Ключевыми аспектами здесь являются интуитивность, обратная связь, доступность и эстетическая привлекательность.

Не менее важную роль в создании графических интерфейсов играют технологии и инструменты, позволяющие создавать интерфейсы с высокой интерактивностью и визуальным стилем. Существуют различные фреймворки и библиотеки, такие как Qt, GTK и JavaFX, которые предоставляют разработчикам готовые компоненты для быстрого создания интерфейсов, существенно ускоряя процесс разработки.

С учетом стремительного развития технологий и меняющихся ожиданий пользователей графические интерфейсы становятся все более адаптивными и контекстуальными. Это означает, что интерфейсы могут изменять свое поведение в зависимости от условий использования, таких как размер экрана или доступные устройства. Адаптивные интерфейсы обеспечивают пользователям более персонализированный и эффективный опыт взаимодействия.

В эпоху мобильных технологий также следует отметить значимость проектирования интерфейсов для мобильных устройств, где размер экрана и сенсорное управление требуют особого подхода. Дизайнеры должны учитывать, как пользователи взаимодействуют с интерфейсом на небольших экранах, оптимизируя элементы для удобства.

Таким образом, графический пользовательский интерфейс продолжает оставаться важнейшим компонентом разработки программного обеспечения, открывая широкий спектр возможностей для взаимодействия пользователей с технологиями и создавая более эффективный и комфортный опыт. Учитывая как эстетические, так и функциональные аспекты, разработчики могут создавать интерфейсы, которые будут не только привлекательными, но и интуитивно понятными и доступными для всех пользователей.

## 2.2 Qt Designer

Qt Designer является важнейшим инструментом в арсенале разработчиков, работающих с фреймворком Qt. Это не просто удобная утилита, но и мощный инструмент, который значительно упрощает процесс создания графических пользовательских интерфейсов (GUI). Он позволяет ускорить разработку и сэкономить время, обеспечивая возможность визуально проектировать интерфейсы, избегая необходимости вручную прописывать код для каждого элемента управления. В отличие от традиционных методов разработки, когда приходится детально настраивать каждый элемент интерфейса, Qt Designer позволяет работать с виджетами, перетаскивая их на форму, и оперативно настраивать параметры с помощью панели свойств.

Этот инструмент предоставляет разработчику доступ к широкому набору стандартных виджетов, включая кнопки, текстовые поля, метки и таблицы, а также более сложные компоненты, такие как графики, диалоговые окна и панели инструментов. Возможности Qt Designer подходят для самых разных типов приложений – от простых утилит до многокомпонентных программ. Он позволяет разрабатывать как мобильные приложения, так и настольные, при этом интерфейсы будут корректно масштабироваться в зависимости от устройства и разрешения экрана. Это достигается благодаря встроенной поддержке макетов, которые автоматически подстраивают расположение и размер виджетов при изменении размера окна. Макеты в Qt Designer помогают создавать гибкие и адаптивные интерфейсы, которые могут корректно отображаться на устройствах с разными размерами экранов, включая смартфоны, планшеты и десктопы.

Макеты являются одной из сильных сторон Qt Designer, предоставляя разработчикам возможность динамически управлять положением и размером элементов интерфейса. В зависимости от потребностей проекта, можно использовать различные типы макетов – горизонтальные, вертикальные, сеточные или формы – которые позволяют интерфейсу быть адаптивным и удобным. Например, горизонтальные и вертикальные макеты идеально подходят для создания элементов управления, которые должны быть расположены в линейном порядке, а сетки и формы отлично подходят для более сложных, структурированных интерфейсов. Благодаря этим возможностям интерфейсы, созданные в Qt Designer, могут легко адаптироваться под разные размеры экранов, автоматически изменяя расположение элементов и сохраняя удобство использования.

Особое внимание стоит уделить концепции сигналов и слотов, которая является основой взаимодействия элементов интерфейса в Qt. Это концептуальное решение значительно упрощает обработку событий. Например, не нужно прописывать вручную обработчики для каждого действия пользователя, таких как нажатие кнопок или изменение текста в поле ввода. В Qt Designer достаточно просто визуально связать сигнал от одного виджета (например, нажатие кнопки) с соответствующим слотом (например, вызов функции). Этот механизм позволяет ускорить разработку интерактивных интерфейсов, минимизируя количество кода и делая взаимодействие между элементами удобным и наглядным.

Qt Designer также активно поддерживает возможность локализации интерфейсов, что делает его особенно полезным для разработки многоязычных приложений. Он предлагает инструменты для интеграции различных языков, а также позволяет быстро переключаться между языковыми версиями интерфейса, что особенно важно для приложений, ориентированных на международную аудиторию. С помощью встроенных средств локализации можно легко адаптировать приложение под различные языки и культурные особенности, что позволяет создавать приложения, подходящие для пользователей по всему миру.

Платформенная независимость Qt является еще одним ключевым преимуществом. Интерфейсы, созданные с помощью Qt Designer, работают на различных операционных системах – Windows, macOS, Linux и многих других – без необходимости значительных изменений в коде. Qt автоматически адаптирует интерфейс под особенности каждой платформы, что упрощает создание кросс–платформенных решений. Это позволяет разработчикам создавать приложения, которые могут работать на множестве устройств и операционных систем, не беспокоясь о дополнительных корректировках интерфейса для каждой из них.

Кроме того, Qt Designer предоставляет гибкость в плане оформления интерфейса с помощью стилей и тем. Это позволяет создавать визуально уникальные приложения, адаптированные под предпочтения пользователя или фирменный стиль компании. Qt Designer поддерживает как стандартные стили для различных операционных систем (Windows, macOS и т. д.), так и возможность применения кастомных стилей с помощью CSS-подобного синтаксиса. Это открывает широкие возможности для создания визуально привлекательных и функциональных интерфейсов, которые соответствуют современным требованиям дизайна.

Важно отметить, что Qt Designer хорошо интегрируется с системой контроля версий, что делает его особенно полезным при работе в команде. Возможность отслеживания изменений и синхронизации работы нескольких разработчиков позволяет избежать конфликтов и ошибок при разработке интерфейсов. Это также дает возможность эффективно управлять проектами, обеспечивая постоянное обновление и улучшение интерфейсов в процессе разработки.

Когда интерфейс готов, его можно легко интегрировать с основным проектом. Интерфейсы, созданные в Qt Designer, сохраняются в виде файлов с расширением .ui, которые можно скомпилировать в C++ код. Это позволяет разработчикам сосредоточиться на логике приложения, не тратя время на ручную настройку каждого виджета. Таким образом, проектирование интерфейсов и написание функционального кода могут идти параллельно, что ускоряет процесс разработки и делает его более эффективным.

Qt Designer – это не просто визуальный редактор, но и мощный инструмент для создания высококачественных, адаптивных и кросс–платформенных интерфейсов. Его возможности позволяют разработчикам быстро создавать функциональные и визуально привлекательные интерфейсы для настольных и мобильных приложений, поддерживающие многоязычность и адаптирующиеся под различные устройства. С помощью Qt Designer можно значительно повысить производительность разработки и создать приложения, которые будут работать на разных операционных системах и устройствах с минимальными усилиями.

## 2.3 Создание графического пользовательского интерфейса

Чтобы отобразить форму необходимо дважды кликнуть левой кнопкой мыши по названию файла .ui на вкладке "Проекты"(см. рисунок 2.2).

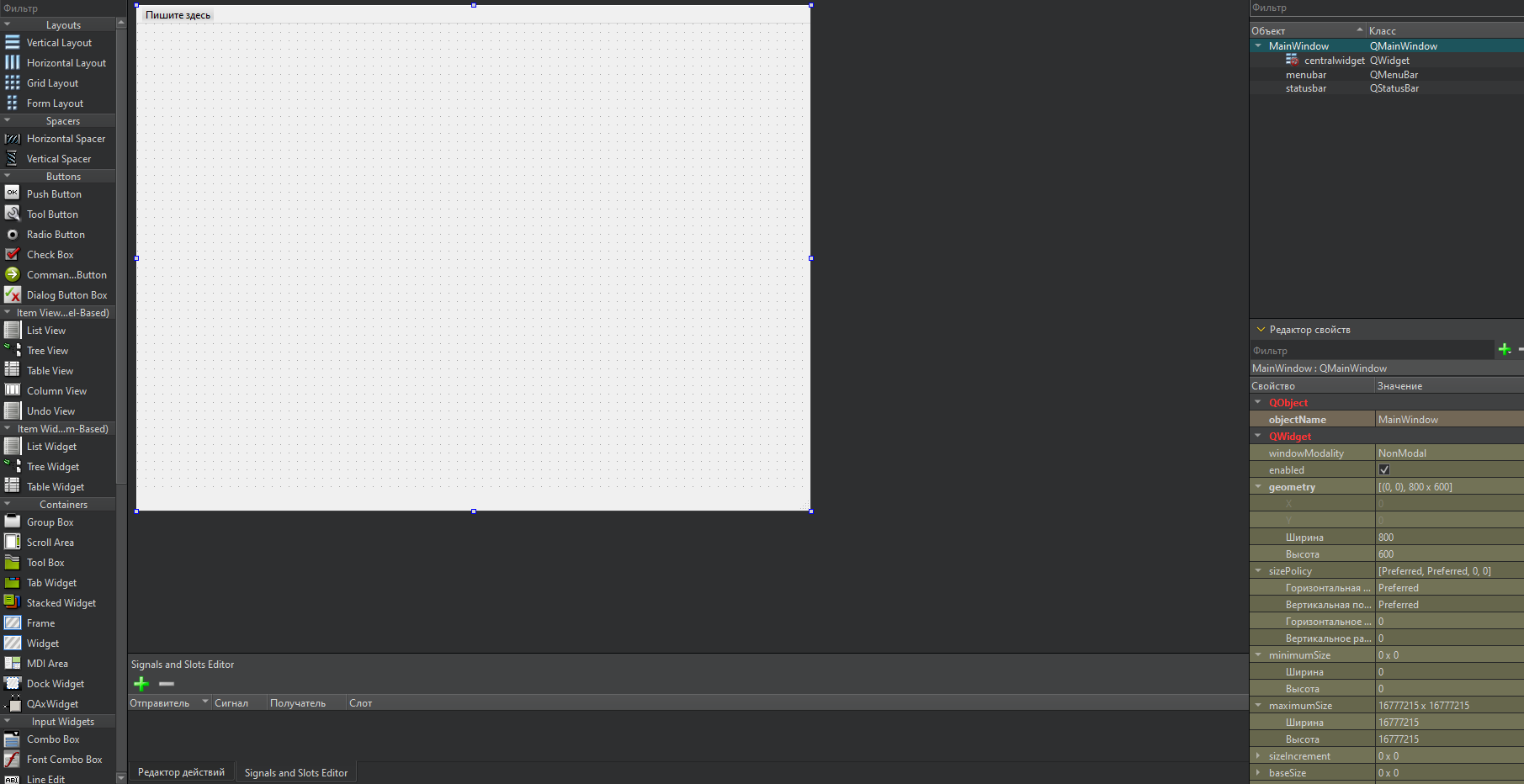


Рисунок 2.2 – Редактирование формы в режиме "Дизайн"

Добавим элементы "Label". Для этого в левой боковой панели выберем раздел "Display widgets" и найдем виджет "Label"(см. рисунок 2.3).

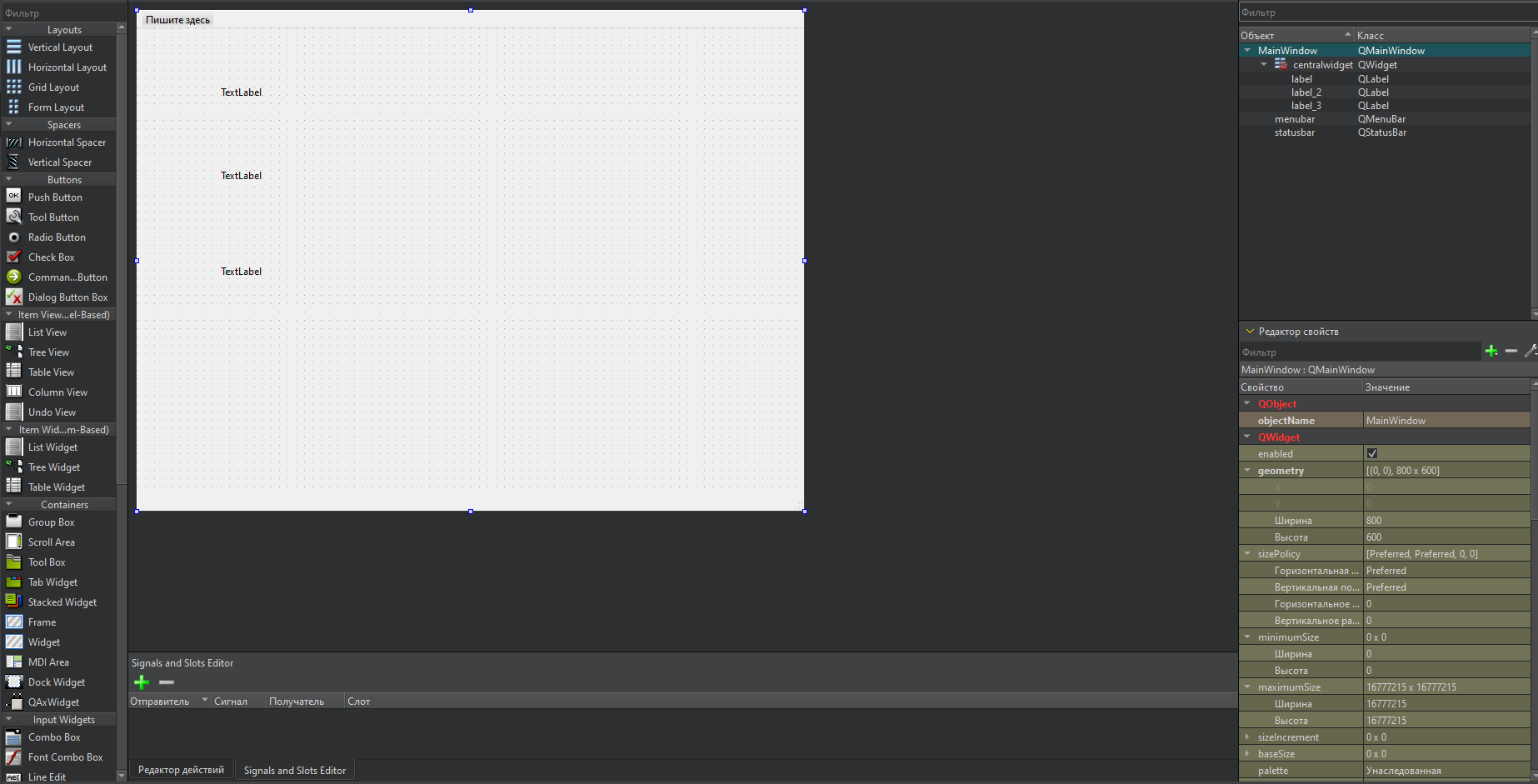


Рисунок 2.3 – Создание элементов "Label"

Для ввода значений нужно добавить виджеты "Line Edit". На левой боковой панели находим пункт "Input widgets", находим элемент "Line Edit". Размещаем необходимое количество на форме (см. рисунок 2.4).

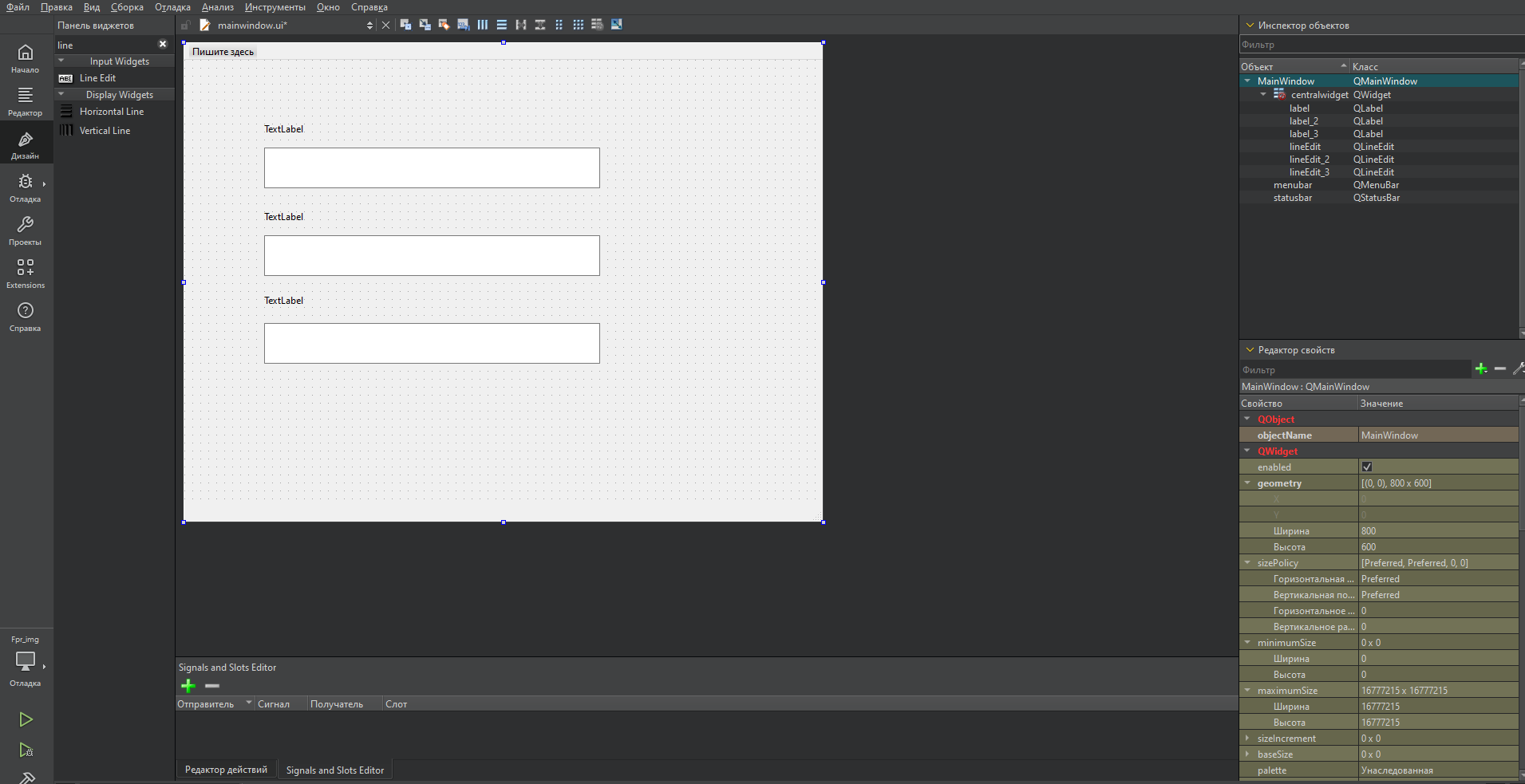


Рисунок 2.4 – Создание элементов "Line Edit"

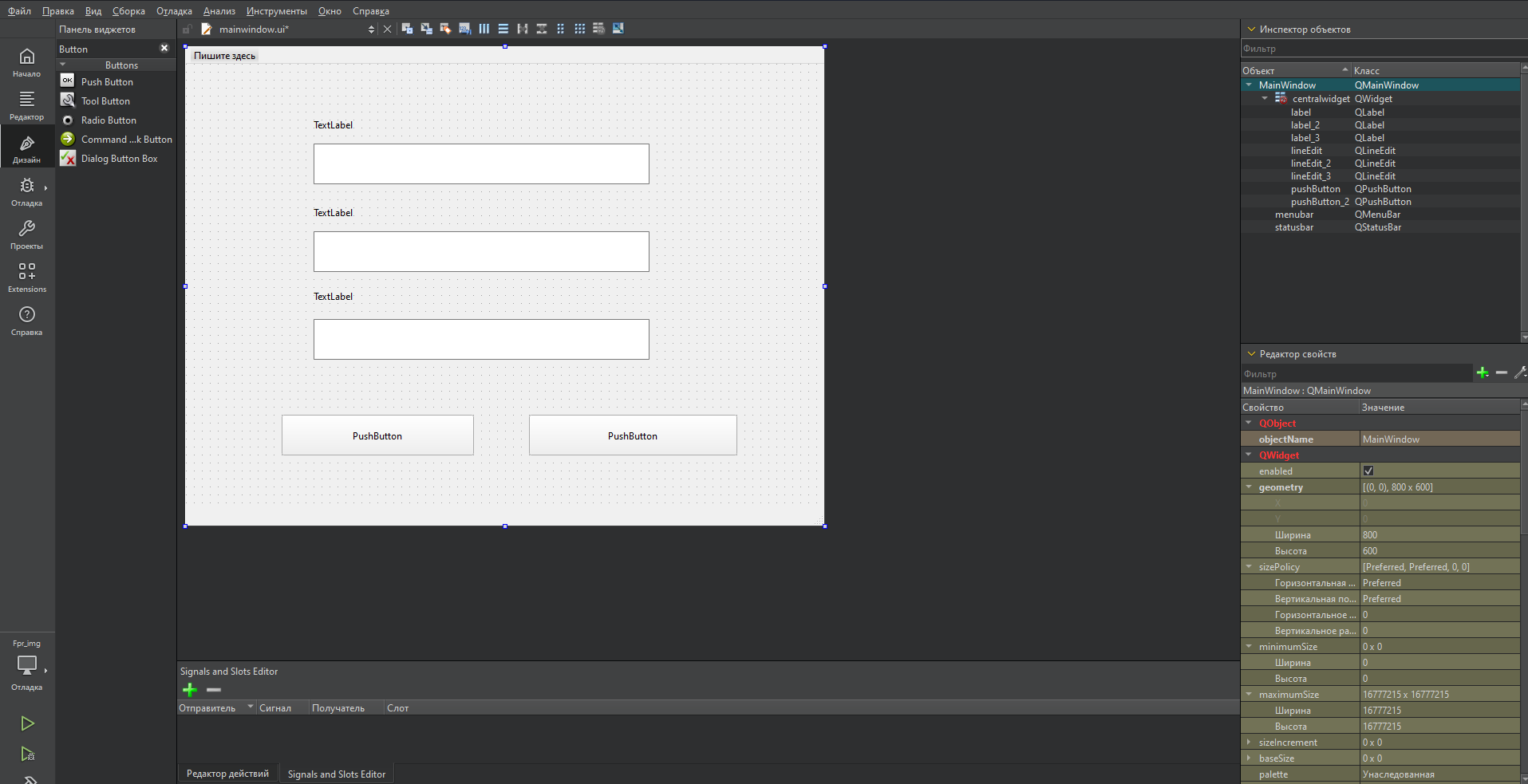
Добавим виджет "Button", который будет использоваться для начала обработки значений после нажатия. При нажатии на кнопку будет вызван определённый слот в классе, взаимодействующем с графическим интерфейсом. Это приведёт к активации конкретных методов, отвечающих за обработку значений (см. рисунок 2.5).

Рисунок 2.5 – Создание виджета "Button"

Теперь пропишем стили для наших элементов: "Line Edit", "Button", "Label". Для этого наведите курсор на ваш элемент "Label" и нажмите правой кнопкой мыши (см. рисунок 2.6).

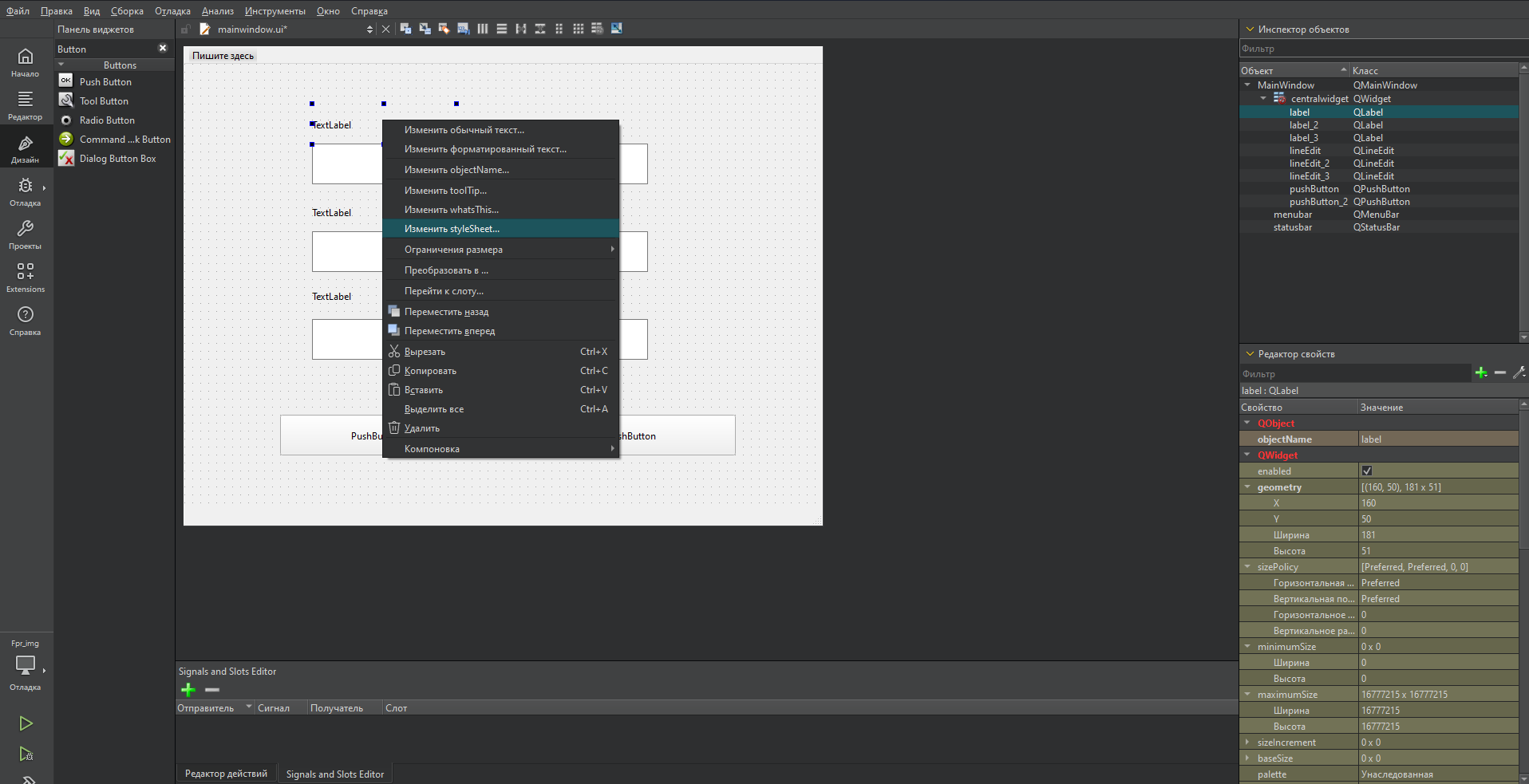


Рисунок 2.6 – Открытие панели со свойствами элемента

Далее нужно нажать на поле "изменить styleSheet". После нажатия вы перейдете на страницу изменения свойств элемента (см. рисунок 2.7).

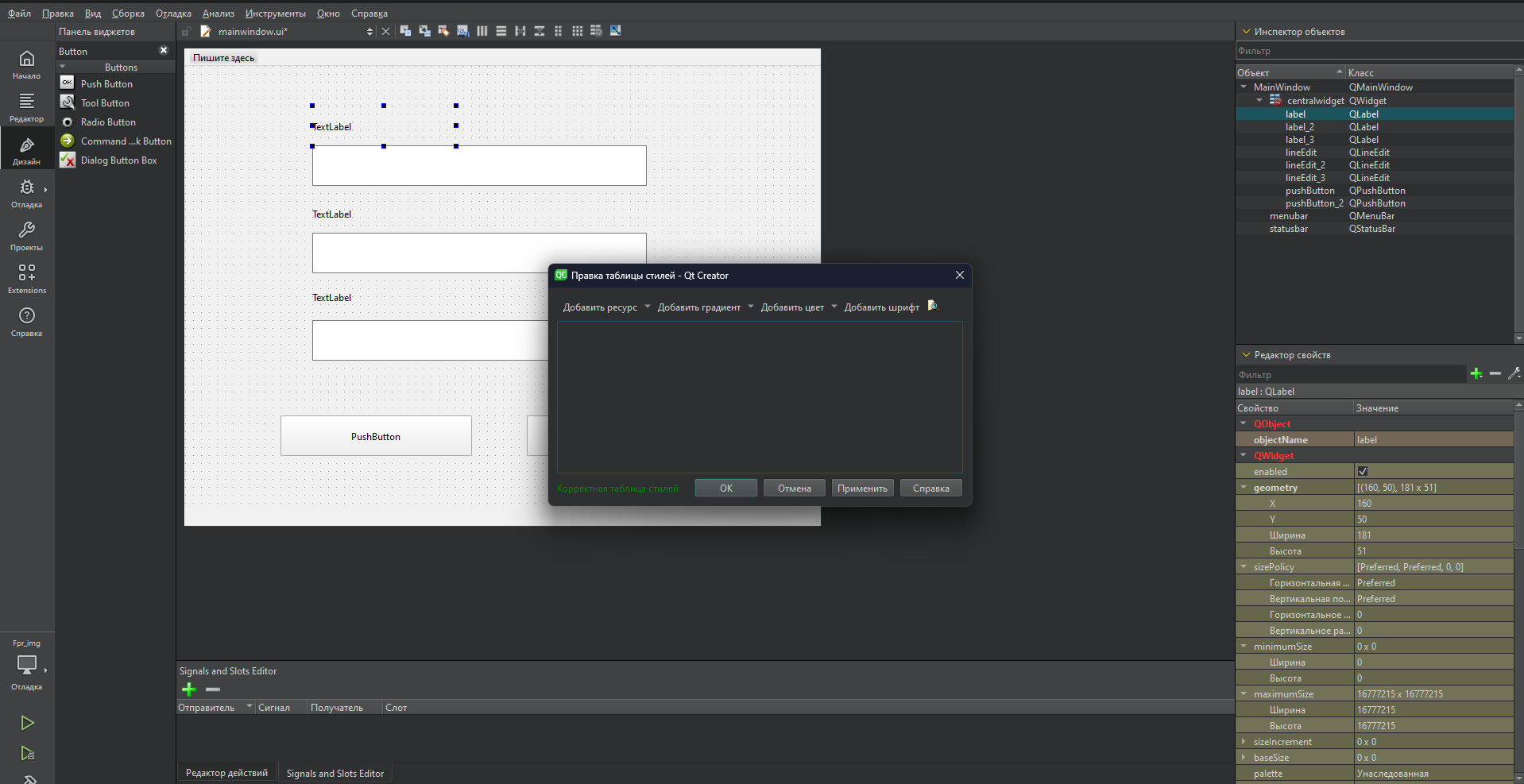


Рисунок 2.7 – Открытие панели со свойствами элемента

В этом поле нужно прописать свойства, которые вы хотите видеть для вашего элемента. Для этого ввидите команды в поле ввода (см. рисунок 2.8).

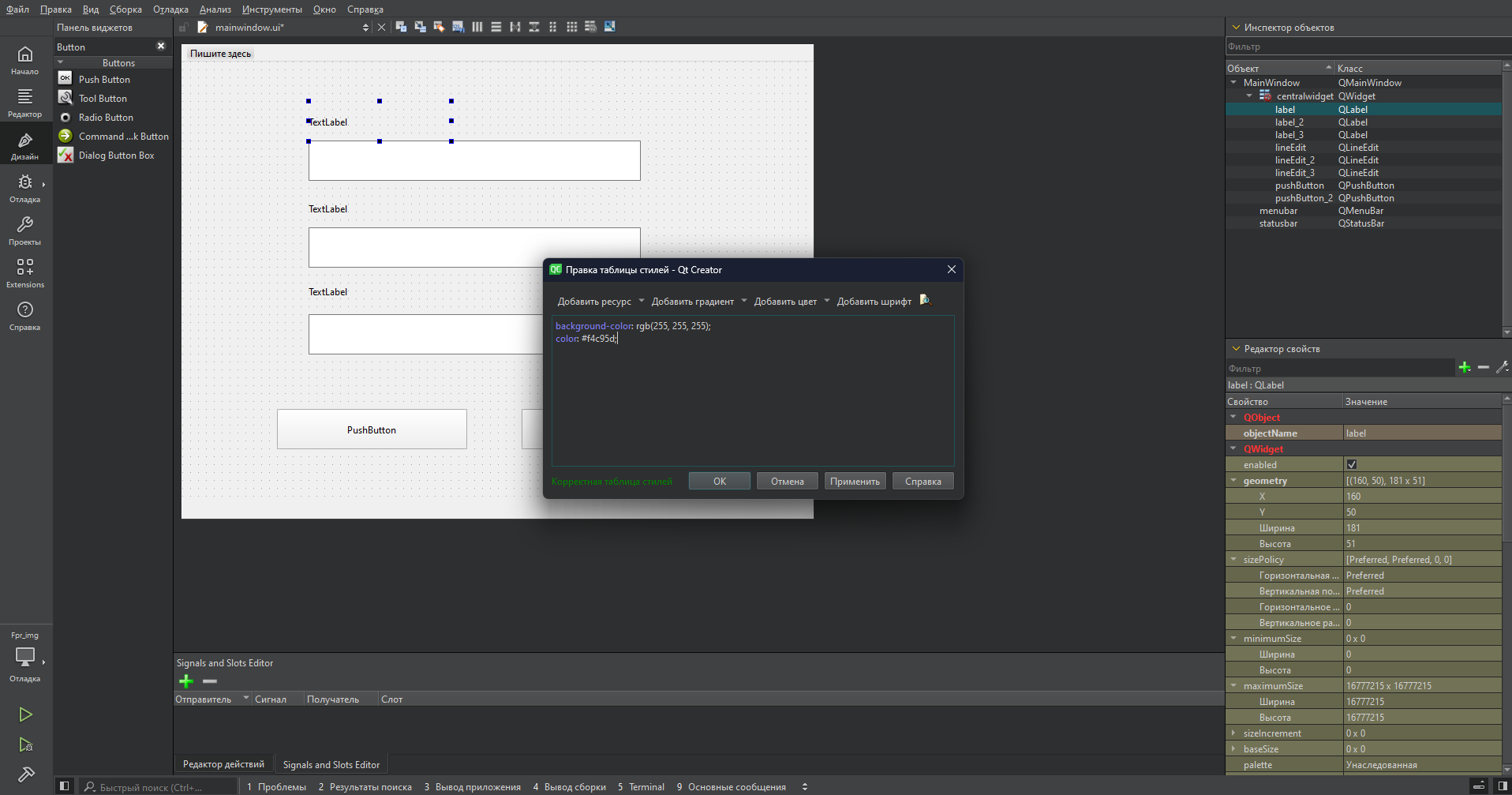


Рисунок 2.8 – Вставка свойств элемента в панель

Аналогично впишите все свойства для остальных элементов вашего окна. После того как вы это сделаете у вас уже будет примерный дизайн на экране

(см. рисунок 2.9).

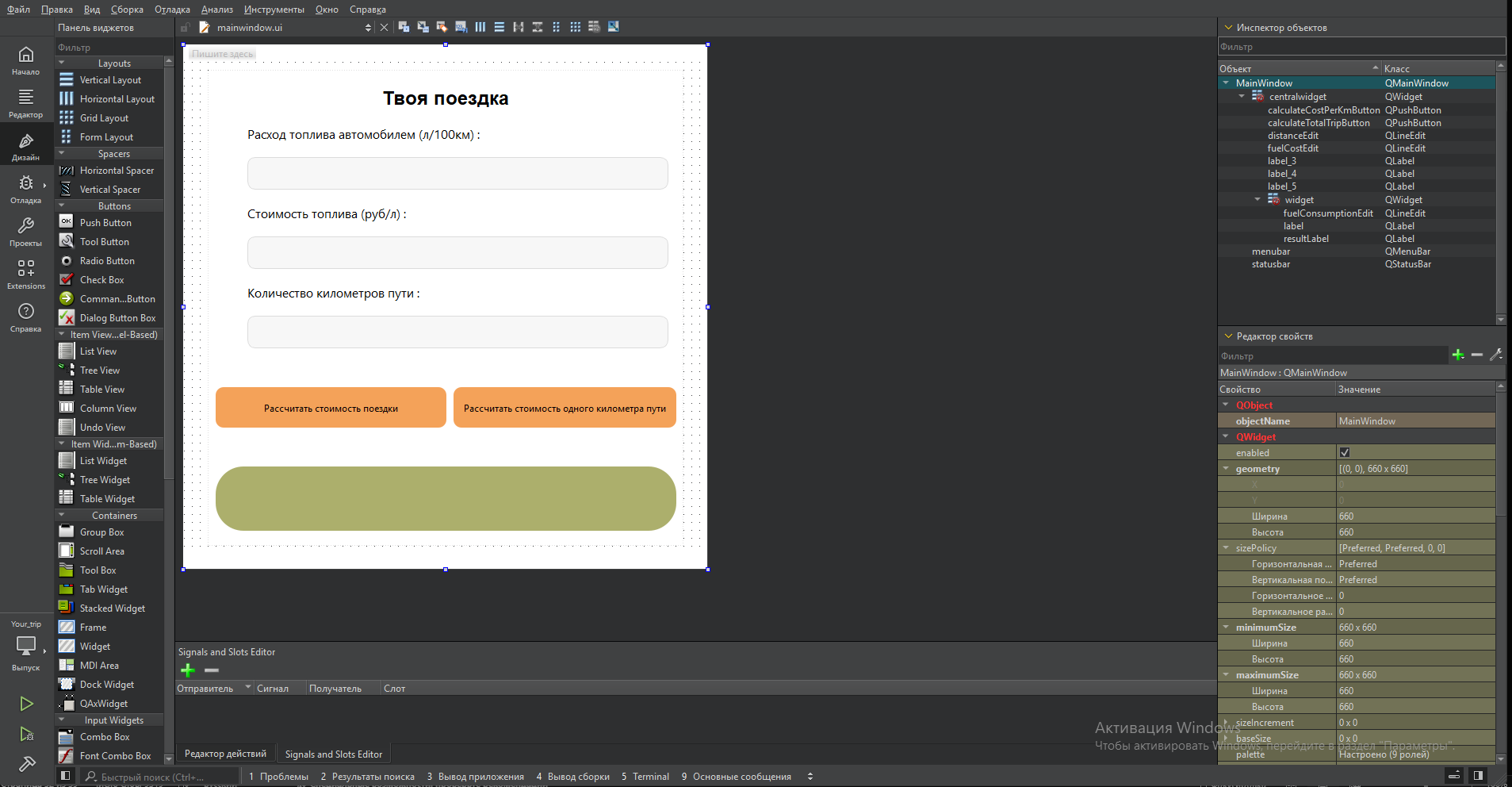


Рисунок 2.9 – Дизайн экрана после вставки стилей в панель

Теперь нужно изменить фон и стандартную иконку приложения. Для этого переместите необходимые изображения в папку проекта, затем создайте файл ресурсов и добавьте в него эти изображения. После этого создайте ещё один "Label" и загрузите в него изображение из файла ресурсов, используя свойство "pixmap". Кроме того, измените иконку приложения, установив её с помощью свойства "icon"(см. рисунок 2.10).

Рисунок 2.10 – Создание иконки приложения

В этом разделе мы рассмотрели концепцию графического пользовательского интерфейса и утилиту Qt Designer, которая значительно облегчает процесс разработки интерфейсов и их интеграцию с программным кодом. Благодаря мощным возможностям фреймворка Qt разработчики могут создавать интуитивно понятные и эстетически привлекательные интерфейсы, что, в свою очередь, улучшает взаимодействие пользователей с приложениями.

Использование Qt Designer не только ускоряет процесс проектирования, но и обеспечивает высокое качество и функциональность интерфейсов, что является ключевым аспектом успешного программного обеспечения. Этот инструмент позволяет разработчикам сосредоточиться на логике приложения, минимизируя время, затрачиваемое на создание визуальных элементов. В результате, проекты становятся более организованными, а конечные пользователи получают более приятный и удобный опыт работы с программой.

# 3 РАЗРАБОТКА ПРОГРАММНОГО СРЕДСТВА

## 3.1 Unified Modeling Language (UML)

Unified Modeling Language (UML) представляет собой мощный язык графического описания, предназначенный для объектного моделирования в различных областях, включая разработку программного обеспечения, моделирование бизнес-процессов, проектирование систем и отображение организационных структур. UML был создан с целью облегчить разработку сложных программных решений, позволяя разработчикам и архитекторам представлять систему в виде наглядных диаграмм, которые описывают ее компоненты, их взаимодействия, а также поведение и состояние объектов. UML используется для того, чтобы обеспечить четкость и прозрачность на всех этапах разработки, начиная от концептуального проектирования и заканчивая подробной реализацией.

Одна из отличительных особенностей UML заключается в его способности работать на различных уровнях абстракции. Это означает, что с помощью UML можно моделировать как общие, высокоуровневые структуры системы, так и более детализированные элементы, включая ее компоненты, поведение и взаимодействия. Например, диаграммы классов идеально подходят для представления статической структуры системы и ее объектов, в то время как диаграммы последовательности более подходят для моделирования динамических процессов, например, последовательности вызовов методов и сообщений между объектами. Такой подход обеспечивает ясность и позволяет одновременно видеть как статическую архитектуру системы, так и ее динамическое поведение, что особенно важно в сложных проектах, где взаимодействия между объектами играют ключевую роль.

UML активно поддерживает использование различных методологий разработки, включая как традиционные водопадные модели, так и гибкие подходы, такие как Agile или Scrum. В методологиях, ориентированных на гибкость и быстрые итерации, таких как Agile, важным аспектом является быстрая адаптация к изменениям требований и возможность оперативного реагирования на изменения. UML идеально подходит для таких методологий, так как позволяет моделировать систему на разных уровнях детализации – начиная с диаграмм прецедентов, которые дают общее представление о взаимодействии пользователей с системой, и заканчивая более детализированными диаграммами, такими как диаграммы классов или компонентов. Такой подход дает разработчикам возможность работать с необходимым уровнем детализации на каждом этапе разработки, одновременно упрощая процесс изменений и позволяя быстро пересматривать проект при изменении бизнес-требований.

Другим значимым преимуществом UML является его универсальность при работе с распределёнными командами. Диаграммы UML представляют собой графическое представление системы, что позволяет членам команды с различным уровнем знаний и компетенций понимать, как работает система. Например, диаграммы прецедентов, которые показывают взаимодействие пользователей с системой, будут полезны бизнес-аналитикам и заказчикам, в то время как диаграммы классов и компонентов подойдут разработчикам и архитекторам для проектирования технической части системы. Такое разделение диаграмм по уровням абстракции позволяет избежать недоразумений и способствует лучшему взаимодействию между различными участниками проекта, обеспечивая единую картину системы и её компонентов.

UML также является отличным инструментом для моделирования сложных распределённых, многослойных и многокомпонентных систем. В таких проектах, где нужно интегрировать множество сервисов, подсистем и внешних компонентов, UML помогает спроектировать систему таким образом, чтобы компоненты взаимодействовали друг с другом на основе чётких контрактов и соглашений. Например, диаграмма компонентов может быть использована для описания взаимодействия между модулями системы, диаграмма развертывания – для отображения того, как эти модули будут размещаться на различных серверах и платформах. Использование UML помогает предсказать возможные проблемы на стадии проектирования, а также выработать архитектуру, которая будет гибкой и легко адаптируемой.

В последние годы, с развитием концепции микро-сервисов, UML продолжает сохранять свою актуальность. В микро-сервисной архитектуре каждый сервис представляет собой независимый модуль, который взаимодействует с другими через стандартизированные интерфейсы. В таком контексте UML помогает моделировать взаимодействие между сервисами, их API, а также потоки данных. Например, диаграммы компонентов и диаграммы взаимодействий позволяют чётко отразить все эти взаимодействия, что является важным для успешной разработки и эксплуатации таких систем. UML помогает гарантировать, что все сервисы правильно взаимодействуют между собой, а их коммуникация не нарушает принципов архитектуры.

Еще одной значимой чертой UML является его способность интегрироваться с инструментами автоматической генерации кода. Современные средства разработки, такие как Enterprise Architect или Rational Rose, позволяют автоматически генерировать код на основе UML-диаграмм, что значительно ускоряет процесс разработки. Это помогает избежать ошибок, которые могут возникнуть при ручном написании кода, а также экономит время разработчиков, давая им возможность сосредоточиться на логике приложения, а не на создании инфраструктурного кода. С помощью UML можно автоматически создавать основные классы, интерфейсы и даже целые компоненты, что значительно ускоряет процесс разработки.

Также стоит отметить, что UML используется в процессе тестирования программного обеспечения. Диаграммы, такие как диаграммы последовательности, диаграммы активности и диаграммы состояний, позволяют моделировать поведение системы в различных ситуациях, что помогает создать тестовые сценарии и случаи для проверки работоспособности системы. Тестирование, основанное на моделях UML, гарантирует, что система будет работать корректно и в соответствие с заданными требованиями. Эти диаграммы помогают охватить как обычные потоки выполнения, так и исключительные случаи, которые могут возникнуть в процессе эксплуатации системы, что делает процесс тестирования более полным и эффективным.

Особое внимание стоит уделить возможностям UML для локализации программных продуктов. Современные приложения часто требуют поддержки нескольких языков и культур, и UML помогает моделировать такие аспекты системы, как изменяющиеся форматы данных, языковые предпочтения и локализация пользовательского интерфейса. Диаграммы прецедентов и классов, создаваемые с использованием UML, могут учитывать эти особенности, позволяя команде разрабатывать и тестировать локализованные версии программного обеспечения. Встроенные функции локализации позволяют значительно упростить работу с многоязычными проектами и ускорить выход продукта на международные рынки.

Кроме того, UML обеспечивает платформенную независимость, что является одним из его основных преимуществ. Созданные с помощью UML модели могут быть использованы для разработки программного обеспечения, которое будет работать на различных операционных системах и устройствах, от мобильных телефонов до серверов и ПК. Это особенно важно в условиях современной разработки кросс–платформенных приложений, когда необходимо обеспечить совместимость между различными операционными системами, такими как Windows, Linux, macOS и другие. UML помогает гарантировать, что все части системы будут правильно взаимодействовать, независимо от того, на какой платформе они работают.

Интеграция UML с современными инструментами для разработки и управления проектами также является важной чертой этого языка. Современные системы управления проектами, такие как JIRA, GitHub и другие, могут работать с UML-моделями, позволяя разработчикам отслеживать изменения в проекте, управлять задачами и координировать работу разных членов команды. Это делает процесс разработки более прозрачным и организованным, помогает избежать недоразумений и ошибок, а также позволяет легко отслеживать версии проектируемой системы.

Наконец, использование UML помогает улучшить общее качество создаваемых систем. Моделирование на ранних стадиях разработки помогает выявить потенциальные проблемы и риски, которые могут возникнуть в процессе создания приложения. Благодаря этому можно избежать ошибок, которые могли бы существенно повлиять на проект в дальнейшем. UML также служит отличной основой для последующего обслуживания и модификации системы, так как диаграммы могут быть обновлены по мере изменений требований или архитектуры, что значительно упрощает поддержку проекта в долгосрочной перспективе.

Таким образом, Unified Modeling Language продолжает оставаться важным и универсальным инструментом в арсенале разработчиков. Его использование помогает эффективно проектировать сложные системы, улучшать коммуникацию внутри команды, минимизировать ошибки и обеспечивать высокий уровень качества разрабатываемого программного обеспечения. Независимо от того, какой методологии разработки придерживается команда – будь то классическая водопадная модель или гибкие методологии, такие как Agile – UML помогает упорядочить процесс создания системы, обеспечивая четкую и понятную картину того, как система будет функционировать.

UML-диаграмма классов программного средства представлена в приложении А.

## 3.2 Классы разработанного программного средства

Класс Trip представляет собой модель поездки, содержащую информацию о расходе топлива и стоимости топлива. Поля класса, такие как fuelConsumption, который обозначает расход топлива на 100 километров, и fuelCost, который указывает на стоимость топлива за литр, имеют атрибут доступа protected. Это значит, что они могут быть доступны только в самом классе и его производных.

Класс включает конструктор, который инициализирует эти поля при создании объекта. Метод info формирует строку с информацией об объекте, позволяя пользователю получить представление о параметрах поездки. Метод costPerKilometer предназначен для вычисления стоимости одного километра пути, основываясь на значениях, заданных в полях класса. Таким образом, класс Trip предоставляет полезные инструменты для расчета затрат на поездку и управления данными о топливе(см. рисунок 3.1). Описание класса Trip, его полей и методов находится в файле Trip.h.

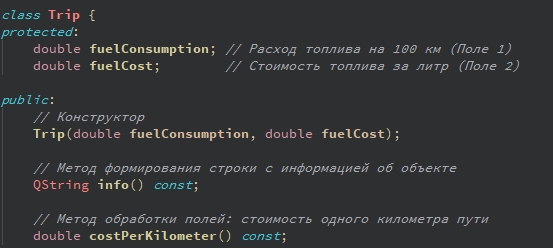


Рисунок 3.1 – Описание класса Trip

Класс LongTrip является производным от класса Trip и наследует его поля и методы. В дополнение к полям базового класса, он имеет собственное поле distance, которое представляет собой количество пройденных километров и имеет атрибут доступа private. Это означает, что информация о расстоянии доступна только внутри класса LongTrip.

Конструктор класса LongTrip инициализирует объект, принимая параметры для расхода топлива, стоимости топлива и расстояния, что позволяет создавать экземпляры с полным набором данных, необходимым для расчета затрат на поездку. Метод totalTripCost вычисляет общую стоимость всей поездки, основываясь на расстоянии, расходе топлива и его стоимости, что делает класс полезным для более сложных расчетов, связанных с длительными поездками (см. рисунок 3.2).

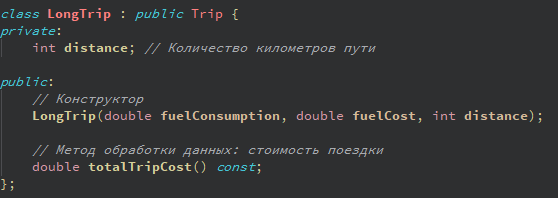


Рисунок 3.2 – Описание класса LongTrip

Класс MainWindow представляет собой основной интерфейс приложения и управляет пользовательским взаимодействием. Все поля этого класса имеют атрибут доступа private, что обеспечивает инкапсуляцию данных и предотвращает их несанкционированный доступ извне. Методы класса являются публичными, что позволяет другим компонентам приложения взаимодействовать с ним.

Класс содержит два слота: первый из них предназначен для расчета общего дохода, который отображается в пользовательском интерфейсе при нажатии соответствующей кнопки. Второй слот отвечает за расчет чистого дохода с учетом налогов, что позволяет пользователю получить более точную информацию о своих финансах. Конструктор класса MainWindow и деструктор отвечают за инициализацию необходимых ресурсов при создании объекта и освобождение памяти при его уничтожении, соответственно.

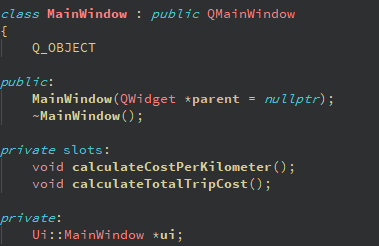
Описание полей, методов и слотов класса находится в заголовочном файле mainwindow.h, что позволяет организовать код и сделать его более структурированным. Таким образом, класс MainWindow выполняет важную роль в управлении пользовательским интерфейсом и обработке бизнес-логики приложения (см. рисунок 3.3).

Рисунок 3.3 – Описание класса MainWindow

## 3.3 Директивы

Перед выполнением программы всегда выполняются директивы препроцессора, в основном они используются для подключения к программе заголовочных и библиотечных файлов. Препроцессор в языках C и C++ выполняет важнейшую роль в подготовке исходного кода к компиляции, обрабатывая различные директивы, которые управляют процессом сборки и настройкой программы. Эти директивы позволяют настраивать поведение компилятора, модифицировать код на стадии препроцессинга и обеспечить гибкость при разработке сложных систем. Препроцессор предоставляет механизмы для включения внешних файлов, условной компиляции, оптимизации и управления различными аспектами процесса компиляции, что делает разработку более гибкой и эффективной. Однако, помимо основных директив, таких как #include, #define и #if, существует целый ряд других директив, каждая из которых решает свои специфические задачи и дополняет функциональность препроцессора.

Одной из таких директив является #undef, которая используется для отмены ранее определённого макроса. Это может быть полезно, если в процессе работы над проектом возникла необходимость изменить определение макроса или удалить его использование. После применения директивы #undef компилятор больше не будет воспринимать этот макрос, и он больше не будет заменяться на соответствующее значение в коде. Например, это может быть полезно в случае, когда макрос должен изменяться в зависимости от платформы, и необходимо гарантировать, что старые значения макросов больше не будут использоваться после изменения конфигурации.

Директива #ifdef позволяет включать в код только те фрагменты, которые зависят от определения определённого макроса. Препроцессор проверяет, был ли определён этот макрос, и если это так, то включает соответствующий код. Если же макрос не определён, то соответствующая часть кода будет проигнорирована. Директивы #ifndef и #endif работают аналогично, но в противоположном смысле: они включают код, только если макрос не был определён. Это полезно при работе с кроссплатформенными программами, когда необходимо адаптировать код для различных операционных систем, компиляторов или конфигураций.

Препроцессор также предоставляет директиву #pragma, которая используется для отправки инструкций непосредственно компилятору. Она не является обязательной и зависит от компилятора, но она позволяет более тонко настраивать поведение компилятора, включая различные аспекты оптимизации и предупреждений. Например, с помощью #pragma можно указать компилятору, чтобы он отключил определённые предупреждения, или наоборот, включил их для более строгой проверки кода. Другим примером является директива #pragma once, которая используется для предотвращения многократного включения одного и того же заголовочного файла. Эта директива является альтернативой традиционному механизму защиты от повторного включения через #ifndef и позволяет компилятору гарантировать, что файл будет включён только один раз в процессе компиляции.

Также стоит отметить директиву #error, которая используется для генерации пользовательских сообщений об ошибках на стадии препроцессинга. Программисты могут использовать её для проверки условий, которые должны быть выполнены перед компиляцией, например, проверка версии компилятора или наличия необходимых библиотек. Если условие не выполняется, препроцессор генерирует ошибку, и компиляция останавливается с заданным сообщением. Эта директива полезна для разработки многофункциональных и кроссплатформенных решений, где необходимо выполнять проверки условий конфигурации системы перед началом компиляции.

Директива #line позволяет изменить номер строки или имя файла в исходном коде, который обрабатывается компилятором. Это может быть полезно для генерации кода автоматически или для создания отчетности о местоположении ошибок в сгенерированном коде. Например, если сгенерированный код автоматически вставляется в проект, использование #line помогает компилятору правильно отслеживать местоположение ошибок, отображая их в соответствующих исходных файлах.

Кроме того, препроцессор в C и C++ поддерживает директиву #include, которая позволяет подключать внешние файлы в программу. Это важный инструмент для разделения программы на более мелкие части и использования уже написанных и протестированных модулей кода. Однако подключение внешних файлов требует аккуратности, чтобы избежать циклических зависимостей и конфликтов между различными частями кода. Для этого используется механизм защиты от многократного включения, который предотвращает повторное включение одного и того же заголовочного файла. Директивы #ifndef, #define и #endif помогают гарантировать, что заголовочные файлы подключаются лишь один раз, что улучшает производительность компиляции и предотвращает ошибки.

Особое внимание стоит уделить директивам, связанным с условной компиляцией. Когда программа разрабатывается для нескольких платформ, важно, чтобы код мог адаптироваться под различные операционные системы или архитектуры. Для этого препроцессор предоставляет возможность использовать различные директивы условной компиляции, такие как #if, #else, #elif и #endif. Эти директивы позволяют программисту включать или исключать определённые фрагменты кода в зависимости от того, какие макросы или флаги были определены в процессе сборки программы. Например, условная компиляция может использоваться для включения кода, специфичного для Windows, в одном случае, и для Linux – в другом. Это позволяет создавать кросс–платформенные приложения, которые будут корректно работать на разных операционных системах.

Особенность использования препроцессора заключается в его способности настраивать код на уровне его обработки до этапа компиляции, что позволяет избежать множества ошибок и ускоряет процесс разработки. Препроцессор действует как фильтр, который подготавливает исходный код для дальнейшей обработки компилятором, что позволяет создать более чистый и структурированный код, а также значительно ускоряет процесс компиляции. Важно понимать, что несмотря на всю мощь препроцессора, его использование требует тщательной проверки и внимания к деталям, поскольку неконтролируемое использование директив может привести к сложным ошибкам, которые будут трудно диагностировать.

Еще одной важной функцией препроцессора является создание простых макросов, которые могут быть использованы для замены значений и выражений на этапе препроцессинга. Однако макросы имеют свои особенности, и они не проверяются на корректность как обычные функции. Они подставляют код в местах вызова, что иногда может приводить к ошибкам, если макросы используются неправильно. Например, использование макросов в выражениях без дополнительных скобок может привести к неожиданным результатам из-за особенностей приоритета операций.

В заключение, препроцессор предоставляет разработчикам огромное количество инструментов для гибкой настройки и оптимизации исходного кода. Он позволяет выполнять операции на этапе препроцессинга, которые значительно ускоряют компиляцию, обеспечивают кроссплатформенность и управляют зависимостями между различными частями кода. Применение препроцессора помогает уменьшить вероятность ошибок, улучшить структуру программы и создать более эффективные и масштабируемые приложения. Однако для правильного использования препроцессора важно внимательно относиться к каждой директиве, чтобы избежать сложных для диагностики ошибок и обеспечить корректную работу программы на всех этапах компиляции.

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

## 3.4 Механизм сигналов и слотов в Qt

В Qt используется механизм сигналов и слотов. Сигнал вырабатывается, когда происходит определенное событие. Слот – это функция, которая вызывается в ответ на определенный сигнал. В системе сигналов и слотов в Qt реализован четкий механизм взаимодействия между пользовательским интерфейсом и логикой приложения. Основной класс MainWindow, который наследуется от QMainWindow, содержит два слота: calculateCostPerKilometer() и calculateTotalTripCost(). Эти слоты предназначены для обработки нажатий кнопок, которые позволяют пользователю вычислять стоимость поездки.

Когда пользователь нажимает кнопку для расчета стоимости за километр, срабатывает сигнал от кнопки calculateCostPerKmButton, который подключен к слоту calculateCostPerKilometer(). В этом слоте происходит проверка введенных пользователем данных о расходе топлива и стоимости топлива. Если данные корректны, создается объект класса Trip, который находится в файле Trip.h, и вызывается метод costPerKilometer(), рассчитывающий стоимость одного километра. Результат выводится на экран в элемент resultLabel.

Аналогично, при нажатии кнопки для расчета общей стоимости поездки активируется сигнал от кнопки calculateTotalTripButton, подключенный к слоту calculateTotalTripCost(). Этот слот также выполняет проверку введенных данных, включая расстояние. Если все данные валидны, создается объект класса LongTrip, который наследуется от класса Trip и находится в файле LongTrip.h. Затем вызывается метод totalTripCost(), который рассчитывает общую стоимость поездки. Результат отображается на экране в том же элементе resultLabel.

Таким образом, ваша система обеспечивает мгновенную реакцию на действия пользователя, благодаря чему приложение становится интерактивным и удобным в использовании. Сигналы и слоты эффективно обрабатывают события, обеспечивая связь между элементами интерфейса и логикой приложения, что делает код более структурированным и легким для поддержки.

В Qt механизм сигналов и слотов представляет собой одну из самых элегантных и мощных особенностей, позволяющих организовать взаимодействие между объектами. Этот механизм основан на паттерне проектирования «наблюдатель», который обеспечивает эффективную связь между компонентами приложения, позволяя им динамично реагировать на изменения состояния. Сигналы и слоты становятся основой для создания интерактивных интерфейсов, где пользователи могут легко взаимодействовать с приложением, а разработчики – сосредотачиваться на важной бизнес-логике.

Сигнал – это своего рода уведомление, которое вырабатывается объектом в момент наступления определенного события. Например, в пользовательском интерфейсе сигнал может возникать при нажатии кнопки, изменении значения ползунка или закрытии окна. Сигналы не несут в себе никакой логики, их главная задача – сообщить окружающим об изменениях состояния. Часто сигналы имеют параметры, которые передают дополнительную информацию о событии. Например, при нажатии кнопки можно передать координаты курсора, что позволяет точно определить, где произошло взаимодействие.

Слоты, в свою очередь, представляют собой функции, которые вызываются в ответ на определенные сигналы. Каждый слот может принимать параметры, соответствующие тем, которые указаны в сигналах. Это делает их мощным инструментом для обработки событий, позволяя выполнять действия, такие как обновление пользовательского интерфейса, изменение состояния объекта или выполнение бизнес-логики. Слоты можно определять как обычные методы класса, и они могут быть как публичными, так и приватными, что дает разработчику гибкость в организации кода.

Чтобы сигнал и слот могли взаимодействовать, необходимо установить соединение между ними с помощью метода connect(). Это связывает сигнал с конкретным слотом. Например, когда кнопка нажата, вырабатывается соответствующий сигнал, и Qt автоматически вызывает связанный слот для обработки этого события. Такой подход позволяет разработчикам создавать четкие и лаконичные интерфейсы, где не нужно беспокоиться о том, как именно происходит вызов слота – это делает система интуитивно понятной и простой в использовании.

Механизм сигналов и слотов также поддерживает возможность подключения нескольких слотов к одному сигналу. Это дает возможность одному и тому же событию вызывать несколько действий в различных частях приложения. Например, при закрытии окна можно одновременно сохранить данные и освободить ресурсы, что делает код более эффективным и удобным. Кроме того, слоты могут быть отключены в зависимости от состояния приложения, что позволяет динамически менять поведение программы и адаптироваться к требованиям пользователя.

Одной из ключевых особенностей механизма сигналов и слотов является его асинхронность. Это особенно полезно для предотвращения блокировок пользовательского интерфейса, так как обработка сигналов и вызов слотов могут происходить в отдельных потоках. Это критически важно для приложений, требующих высокой отзывчивости, таких как графические интерфейсы или системы, обрабатывающие большие объемы данных. Разработчики могут использовать сигналы и слоты для взаимодействия между потоками, обеспечивая безопасный доступ к ресурсам и избегая потенциальных проблем.

Qt также предоставляет возможность использовать лямбда-функции в качестве слотов, что значительно упрощает код и делает его более читаемым. Вместо создания отдельных методов, разработчики могут использовать лямбда-функции для быстрой обработки событий. Это особенно удобно для простых операций, которые не требуют создания отдельного слота, что делает код более лаконичным и понятным.

К тому же, для повышения удобства работы разработчиков в Qt предусмотрены различные инструменты для отладки сигналов и слотов. Например, можно использовать методы, позволяющие отслеживать, какие сигналы и слоты подключены, а также проверять наличие ошибок в соединениях. Это облегчает диагностику проблем и помогает избежать распространенных ошибок, связанных с неправильным использованием механизма. Такой уровень поддержки делает Qt особенно привлекательным для разработчиков, стремящихся к качеству и надежности.

Кроме того, механизм сигналов и слотов в Qt поддерживает работу с объектами, которые могут быть удалены во время выполнения программы. Это достигается за счет использования специального механизма, который автоматически отключает слоты от сигналов, когда объект, к которому они подключены, уничтожается. Это предотвращает возникновение ошибок, связанных с вызовом слотов для несуществующих объектов, что добавляет надежности в систему.

В целом, механизм сигналов и слотов в Qt предоставляет мощный и гибкий инструмент для организации взаимодействия между компонентами приложений. Он позволяет разработчикам создавать более структурированный и поддерживаемый код, упрощает процесс обработки событий и обеспечивает высокую степень модульности. Благодаря этому механизму, разработка приложений с графическим интерфейсом становится более интуитивной и эффективной, позволяя сосредоточиться на реализации бизнес-логики и пользовательского опыта, а не на низкоуровневых деталях взаимодействия между объектами.

Система сигналов и слотов активно используется в различных типах приложений, от простых настольных программ до сложных систем управления и обработки данных. Эта универсальность позволяет разработчикам быстро реагировать на изменения в пользовательском интерфейсе и выполнять необходимые действия, что делает приложения более интерактивными и удобными для пользователей. Благодаря своей простоте и гибкости, механизм сигналов и слотов остается одним из самых популярных методов обработки событий в Qt, обеспечивая высокую производительность и надежность.

Механизм сигналов и слотов не только упрощает обработку событий, но и способствует созданию более модульного и гибкого кода. Этот подход позволяет разработчикам разделять логику приложения на независимые компоненты, которые могут взаимодействовать друг с другом без жесткой связи. Такой подход значительно упрощает тестирование и отладку, поскольку каждый компонент можно протестировать отдельно, а затем интегрировать в общую систему.

Одним из ключевых преимуществ использования сигналов и слотов является возможность динамического управления поведением приложения. В процессе выполнения программы разработчики могут изменять логику обработки событий, подключая или отключая слоты в зависимости от состояния приложения. Это особенно полезно в сложных интерфейсах, где пользователю могут быть доступны различные функции в зависимости от контекста. Такой уровень гибкости делает приложение более отзывчивым и адаптированным к действиям пользователя, что создает более комфортный пользовательский опыт.

Более того, механизм сигналов и слотов позволяет легко реализовывать паттерны проектирования, такие как MVC (Model-View-Controller). В таком подходе модель может генерировать сигналы, уведомляя представление об изменениях, в то время как контроллер может реагировать на действия пользователя, инициируя изменения в модели. Это создает четкое разделение между логикой приложения и пользовательским интерфейсом, что упрощает поддержку и развитие проекта, а также делает код более организованным.

Сигналы и слоты также хорошо подходят для работы в многопоточной среде. Qt предоставляет специальные механизмы для обеспечения безопасного взаимодействия между потоками. Например, при соединении сигнала с слотом, который выполняется в другом потоке, Qt автоматически управляет очередями сообщений и синхронизацией, что позволяет избежать проблем с конкурентным доступом к данным. Это значительно упрощает разработку многопоточных приложений, поскольку разработчику не нужно заботиться о низкоуровневых деталях потоковой безопасности, что освобождает время для более важной разработки.

Кроме того, Qt предоставляет возможность использования временных сигналов, например, с помощью класса QTimer, который может генерировать сигналы по истечении заданного времени. Это может быть полезно для реализации периодических задач, таких как обновление интерфейса или выполнение фоновой обработки данных. Использование временных сигналов позволяет разработчикам создавать более динамичные и интерактивные приложения, способные адаптироваться к требованиям пользователей.

Сигналы и слоты обеспечивают высокую степень инкапсуляции. Объекты могут взаимодействовать друг с другом, не зная внутренней структуры друг друга. Это снижает связанность компонентов и упрощает их замену или модификацию без необходимости вносить изменения в другие части приложения. Такой уровень абстракции делает код более понятным и легким для сопровождения, что особенно важно в больших проектах.

Важно отметить, что механизм сигналов и слотов поддерживает не только стандартные типы данных, но и пользовательские типы, что делает его универсальным инструментом. Разработчики могут определять свои собственные сигналы и слоты с параметрами, основанными на пользовательских структурах или классах. Это позволяет адаптировать систему взаимодействия под специфические нужды приложения и расширяет возможности по обработке событий, что делает Qt особенно мощным инструментом для создания сложных интерфейсов.

Использование сигналов и слотов также может помочь в реализации системы логирования или отслеживания событий. Например, можно создать специальный слот, который будет записывать информацию о каждом вызове сигнала, фиксируя, когда и в каких условиях произошло событие. Это может быть полезно для анализа производительности приложения или для выявления проблем в его работе, обеспечивая разработчикам ценные данные для оптимизации своих приложений.

Кроме этого, механизм сигналов и слотов активно используется в сочетании с другими компонентами Qt, такими как модели и представления (Model/View). Это позволяет создавать сложные пользовательские интерфейсы, где изменения в модели автоматически отражаются в представлениях благодаря сигналам и слотам. Такой подход значительно упрощает управление данными и обновление интерфейса, так как разработчикам не нужно вручную отслеживать изменения и обновлять интерфейс – это происходит автоматически, что делает разработку более эффективной.

Применение сигналов и слотов также нашло свое место в тестировании приложений. Разработчики могут использовать этот механизм для создания тестов, которые проверяют корректность работы сигналов и слотов. Это помогает выявлять ошибки на ранних стадиях разработки и значительно улучшает качество конечного продукта, что является критически важным в условиях современных требований к надежности и производительности программного обеспечения.

В заключение, механизм сигналов и слотов в Qt представляет собой мощный инструмент, который не только упрощает разработку, но и улучшает структуру приложения. Он позволяет создавать гибкие, модульные и легко поддерживаемые системы, которые могут динамически адаптироваться к изменениям состояния и пользовательскому взаимодействию. Понимание и правильное использование этого механизма является важным шагом для любого разработчика, работающего с Qt, и открывает новые возможности для создания сложных и интуитивно понятных приложений. Сигналы и слоты помогают не только сократить время разработки, но и повысить качество программного обеспечения, делая его более отзывчивым и удобным для пользователей, что в конечном итоге обеспечивает успех любого приложения.

Листинг кода программного средства представлен в приложении В.

# ЗАКЛЮЧЕНИЕ

В рамках курсового проекта была выполнена разработка программного средства на языке C++ с использованием фреймворка Qt, направленного на расчёт стоимости одного километра пути и стоимости поездки на основе введённых данных о машине и поездке. В процессе работы были успешно применены принципы объектно-ориентированного программирования, что позволило разделить функционал программы на логические классы и обеспечить гибкость и расширяемость кода.

Разработанная программа содержит два основных класса: Trip и LongTrip. Класс Trip реализует базовую функциональность для расчёта стоимости одного километра пути, используя два поля: стоимость топлива за один литр и расход топлива автомобилем на 100 километров пути.

Дочерний класс LongTrip расширяет функционал базового класса Trip, добавляя поле количество километров пути. В этом классе реализован метод для расчёта конечной стоимости поездки.

Был разработан графический интерфейс с применением Qt Widgets, предоставляющий удобные поля ввода и кнопки для запуска расчётов. В интерфейсе реализованы проверки корректности ввода данных.

В ходе выполнения курсового проекта были достигнуты следующие цели:

1 Изучены основы объектно-ориентированного программирования и применены принципы ООП в проектировании структуры классов.

2 Получены практические навыки работы с библиотекой Qt, её механизмами сигналов и слотов, а также основными виджетами для создания пользовательского интерфейса.

3 Разработан функциональный код, а также алгоритмы для обработки пользовательских данных, что позволило создать программное средство, соответствующее целям и задачам проекта.

Проект продемонстрировал успешное применение методов программирования и разработки графического интерфейса, а также показал возможности фреймворка Qt для создания интерактивных приложений.

# СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1 Страуструп, Б. Язык программирования С++ : Пер. с англ. – М. : Бином, 2023. – 1216 с.

2 Qt documentation. All Classes [Электронный ресурс]. – Режим доступа : https://doc.qt.io/qt-6/classes.html.

3 Система электронного обучения [Электронный ресурс]. – Режим доступа :  <https://lms.bsuir.by>.

5 Учебное пособие по UML: как нарисовать диаграмму классов UML [Электронный ресурс]. – Режим доступа : <https://www.youtube.com/watch?v=ao1ESgIy2Ws>.

6 Qt Company. Qt Widgets [Электронный ресурс]. – Режим доступа : <https://doc.qt.io/qt-6/qtwidgets-index.html>.

7 Qt Company. Qt Widgets [Электронный ресурс]. – Режим доступа : <https://doc.qt.io/qt-6/qtwidgets-index.html>.

8 Бенедикт, Р. Основы Qt для начинающих. – М. : Питер, 2021. – 320 с

9 Шилдт, Х. C++: Полное руководство. – М. : Вильямс, 2018. – 960 с.

# ПРИЛОЖЕНИЕ А (обязательное) UML-диаграмма классов программного средства

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

# ПРИЛОЖЕНИЕ В (обязательное) Листинг кода программного средства

QT += core gui

greaterThan(QT\_MAJOR\_VERSION, 4): QT += widgets

CONFIG += c++17

# You can make your code fail to compile if it uses deprecated APIs.

# In order to do so, uncomment the following line.

#DEFINES += QT\_DISABLE\_DEPRECATED\_BEFORE=0x060000 # disables all the APIs deprecated before Qt 6.0.0

SOURCES += \

longtrip.cpp \

main.cpp \

mainwindow.cpp \

trip.cpp

HEADERS += \

longtrip.h \

mainwindow.h \

trip.h

FORMS += \

mainwindow.ui

win32:RC\_FILE = car.rc

# Default rules for deployment.

qnx: target.path = /tmp/$${TARGET}/bin

else: unix:!android: target.path = /opt/$${TARGET}/bin

!isEmpty(target.path): INSTALLS += target

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#include <QMainWindow>

QT\_BEGIN\_NAMESPACE

namespace Ui { class MainWindow; }

QT\_END\_NAMESPACE

class MainWindow : public QMainWindow {

Q\_OBJECT

public:

explicit MainWindow(QWidget \*parent = nullptr);

~MainWindow();

private slots:

void calculateCostPerKilometer();

void calculateTotalTripCost();

void checkInput();

private:

Ui::MainWindow \*ui;

double convertToDouble(const QString& text, bool\* ok);

};

#endif // MAINWINDOW\_H

#include "mainwindow.h"

#include "ui\_mainwindow.h"

#include "LongTrip.h"

#include <QMessageBox>

#include <QDoubleValidator>

MainWindow::MainWindow(QWidget \*parent)

: QMainWindow(parent)

, ui(new Ui::MainWindow)

{

ui->setupUi(this);

connect(ui->fuelConsumptionEdit, &QLineEdit::textChanged, this, &MainWindow::checkInput);

connect(ui->fuelCostEdit, &QLineEdit::textChanged, this, &MainWindow::checkInput);

connect(ui->distanceEdit, &QLineEdit::textChanged, this, &MainWindow::checkInput);

ui->resultLabel->setAlignment(Qt::AlignCenter);

QDoubleValidator \*fuelConsumptionValidator = new QDoubleValidator(0.0, 1000.0, 3, this);

fuelConsumptionValidator->setNotation(QDoubleValidator::StandardNotation);

ui->fuelConsumptionEdit->setValidator(fuelConsumptionValidator);

QDoubleValidator \*fuelCostValidator = new QDoubleValidator(0.0, 1000.0, 3, this);

fuelCostValidator->setNotation(QDoubleValidator::StandardNotation);

ui->fuelCostEdit->setValidator(fuelCostValidator);

QDoubleValidator \*distanceValidator = new QDoubleValidator(0.0, 10000.0, 3, this);

distanceValidator->setNotation(QDoubleValidator::StandardNotation);

ui->distanceEdit->setValidator(distanceValidator);

connect(ui->calculateCostPerKmButton, &QPushButton::clicked, this, &MainWindow::calculateCostPerKilometer);

connect(ui->calculateTotalTripButton, &QPushButton::clicked, this, &MainWindow::calculateTotalTripCost);

}

MainWindow::~MainWindow()

{

delete ui;

}

void MainWindow::checkInput()

{

QList<QLineEdit\*> inputs = {ui->fuelConsumptionEdit, ui->fuelCostEdit, ui->distanceEdit};

for (QLineEdit\* input : inputs) {

QString text = input->text();

if (text.startsWith('0') && text.length() > 1) {

QMessageBox::warning(this, "Ошибка", "Числа не могут начинаться с 0\nЧисло заменено на ваше второе число");

text = text.mid(1);

input->setText(text);

}

}

}

double MainWindow::convertToDouble(const QString& text, bool\* ok) {

QString modifiedText = text;

modifiedText.replace(',', '.');

return modifiedText.toDouble(ok);

}

void MainWindow::calculateCostPerKilometer() {

checkInput();

bool fuelConsumptionOk, fuelCostOk;

double fuelConsumption = convertToDouble(ui->fuelConsumptionEdit->text(), &fuelConsumptionOk);

if (!fuelConsumptionOk || fuelConsumption <= 0) {

QMessageBox::warning(this, "Некорректные данные",

"Пожалуйста, введите положительное число для расхода топлива (Поле 1).");

return;

}

double fuelCost = convertToDouble(ui->fuelCostEdit->text(), &fuelCostOk);

if (!fuelCostOk || fuelCost <= 0) {

QMessageBox::warning(this, "Некорректные данные",

"Пожалуйста, введите положительное число для стоимости топлива (Поле 2).");

return;

}

LongTrip trip(fuelConsumption, fuelCost, 1); // Используем 1 км для расчета

double cost = trip.costPerKilometer();

QFont font = ui->resultLabel->font();

font.setPointSize(15);

font.setBold(true);

ui->resultLabel->setFont(font);

ui->resultLabel->setText(QString("Стоимость одного километра: %1 руб.").arg(cost));

}

void MainWindow::calculateTotalTripCost() {

checkInput(); // Проверка ввода

bool fuelConsumptionOk, fuelCostOk;

double fuelConsumption = convertToDouble(ui->fuelConsumptionEdit->text(), &fuelConsumptionOk);

if (!fuelConsumptionOk || fuelConsumption <= 0) {

QMessageBox::warning(this, "Некорректные данные",

"Пожалуйста, введите положительное число для расхода топлива (Поле 1).");

return;

}

double fuelCost = convertToDouble(ui->fuelCostEdit->text(), &fuelCostOk);

if (!fuelCostOk || fuelCost <= 0) {

QMessageBox::warning(this, "Некорректные данные",

"Пожалуйста, введите положительное число для стоимости топлива (Поле 2).");

return;

}

bool distanceOk;

double distance = convertToDouble(ui->distanceEdit->text(), &distanceOk);

if (!distanceOk || distance <= 0) {

QMessageBox::warning(this, "Некорректные данные",

"Пожалуйста, введите положительное число для расстояния (Поле 3).");

return;

}

LongTrip longTrip(fuelConsumption, fuelCost, distance);

double totalCost = longTrip.totalTripCost();

QFont font = ui->resultLabel->font();

font.setPointSize(15);

font.setBold(true);

ui->resultLabel->setFont(font);

ui->resultLabel->setText(QString("Общая стоимость поездки: %1 руб.").arg(totalCost));

}

#ifndef LONGTRIP\_H

#define LONGTRIP\_H

#include "Trip.h"

class LongTrip : public Trip {

private:

int distance; // Количество километров пути

public:

LongTrip(double fuelConsumption, double fuelCost, int distance);

double totalTripCost() const;

};

#endif // LONGTRIP\_H

#include "LongTrip.h"

LongTrip::LongTrip(double fuelConsumption, double fuelCost, int distance)

: Trip(fuelConsumption, fuelCost), distance(distance) {}

double LongTrip::totalTripCost() const {

return costPerKilometer() \* distance;

}

#ifndef TRIP\_H

#define TRIP\_H

#include <QString>

class Trip {

protected:

double fuelConsumption;

double fuelCost;

public:

Trip(double fuelConsumption, double fuelCost);

QString info() const;

double costPerKilometer() const;

};

#endif // TRIP\_H

#include "Trip.h"

Trip::Trip(double fuelConsumption, double fuelCost)

: fuelConsumption(fuelConsumption), fuelCost(fuelCost) {}

QString Trip::info() const {

return QString("Расход топлива: %1 л/100км, Стоимость топлива: %2 руб/л")

.arg(fuelConsumption)

.arg(fuelCost);

}

double Trip::costPerKilometer() const {

return (fuelConsumption / 100) \* fuelCost;

}

#include "mainwindow.h"

#include <QApplication>

int main(int argc, char \*argv[])

{

QCoreApplication::setAttribute(Qt::AA\_EnableHighDpiScaling);

QApplication a(argc, argv);

QLocale::setDefault(QLocale(QLocale::Russian, QLocale::Russia));

MainWindow w;

w.show();

return a.exec();

}