Московский государственный технический университет

имени Н.Э. Баумана

==========================================

Факультет «Информатика и системы управления»

Кафедра «Компьютерные системы и сети»

Г.С. Иванова, Т.Н. Ничушкина

**Создание приложений**

**с графическим интерфейсом**

**в среде Qt Creator под Linux ASTRA**

*Учебно-методическое пособие*

*по дисциплине Алгоритмизация и программирование*

Москва

(С) 2023 МГТУ им. Н.Э. БАУМАНА

УДК 004.432

ББК 32.973.26-018.1

И 21

**Иванова Г.С., Ничушкина Т.Н.**

Создание приложений с графическим интерфейсом в среде Qt Creator под Linux Astra. Учебно-методическое пособие по дисциплине Алгоритмизация и программирование. - М.: МГТУ имени Н.Э. Баумана, 2023. 41 с.

Издание содержит методический материал для выполнения домашнего задания по дисциплине «Алгоритмизация и программирование». Во время лабораторной работы студенты должны разобраться в особенностях создания графического интерфейса в среде Qt Creator под Linux Astra с использование Дизайнера и разработать две программы. Одну – полностью по методическим указаниям, вторую самостоятельно, но опираясь на текст программы, приведенный в методических указаниях.

Для студентов МГТУ имени Н.Э. Баумана, обучающихся по программе бакалавриата направления «Информатика и вычислительная техника».

*Рекомендовано учебно-методическим советом МГТУ им. Н.Э. Баумана в качестве учебного пособия*

*Учебное издание*

**Иванова Галина Сергеевна**

**Ничушкина Татьяна Николаевна**

**Создание приложений с графическим интерфейсом в среде Qt Creator**

**под Linux Astra**

© 2023 МГТУ имени Н.Э. Баумана

# Оглавление

[Введение 4](#_Toc152097962)

[1 Описание используемого программного обеспечения 5](#_Toc152097963)

[1.1 Основные принципы событийного программирования 5](#_Toc152097964)

[1.2 Особенности компиляции-сборки программ, использующих библиотеку интерфейсных элементов Qt 6](#_Toc152097965)

[1.3 Основные интерфейсные компоненты библиотеки Qt 7](#_Toc152097966)

[2 Задание 12](#_Toc152097967)

[Часть 1 Простейший калькулятор 12](#_Toc152097968)

[2.1.1 Ход разработки приложения в среде Qt Creator 12](#_Toc152097969)

[2.1.2 Требования к отчету 31](#_Toc152097970)

[2.1.3 Требования к защите 32](#_Toc152097971)

[2.1.4 Контрольные вопросы 32](#_Toc152097972)

[Часть 2 Записная книжка 34](#_Toc152097973)

[2.2.1 Особенности работы с многооконным приложением 35](#_Toc152097974)

[2.2.2 Требования к отчету 38](#_Toc152097975)

[2.2.3 Требования к защите 39](#_Toc152097976)

[2.2.4 Контрольные вопросы 39](#_Toc152097977)

[Литература 40](#_Toc152097978)

# Введение

Интегрированная среда программирования *Qt Creator* предназначена для создания 32х-разрядных консольных приложений и настольных (*desktop*) приложений с графическими интерфейсами в операционных системах *Windows*, *Linux* и *Mac OS*. Среда ориентирована на профессиональных программистов и относится к классу визуальных, в которых разработчику предоставляется возможность прямо на экране cформировать графический интерфейс разрабатываемого программного продукта из объектов стандартных классов элементов управления или объектов классов потомков классов этих интерфейсных элементов.

# Описание используемого программного обеспечения

## Основные принципы событийного программирования

Графические интерфейсы приложений взаимодействуют с остальной частью программы, используя принципы событийного программирования.

*Событийным* называется программирование, при котором основная часть программы представляет собой набор обработчиков выбранных разработчиком *событий*. В качестве событий при этом могут выступать как нажатие какой-либо “кнопки” в окне программы, так и некоторые ситуации в самой программе (например, визуализация окна приложения). Таким образом, цикл работы основной программы представляет собой ожидание какого-либо события, вызов соответствующего обработчика для обработки этого события, после чего вновь следует ожидание события, и цикл повторяется.

Программа, построенная по событийной технологии, не имеет алгоритма в традиционном смысле, так как связь между отдельными ее частями не задана жестко, и последовательность их выполнения зависит от порядка наступления тех или иных событий.

Графический интерфейс в среде *Qt Creator* строится из визуальных компонентов (виджетов) – объектов специальной библиотеки интерфейсных классов, наследуемых от класса *QWidjet*. Задачи разработчика в этом случае:

- разработать проект визуального представления окон приложения;

- определить множество обрабатываемых событий для реализации требуемой функциональности приложения;

- создать проект программы с графическим интерфейсом в среде *Qt Creator*;

- разместить соответствующие виджеты компонентов на виджете формы – в окне приложения, используя Дизайнер или без него;

- написать обработчики выбранных событий;

- связать события c их обработчиками.

## Особенности компиляции-сборки программ, использующих библиотеку интерфейсных элементов Qt

Среда *Qt Creator* предназначена для создания больших программ, элементы которых, включая интерфейсные, размещаются в разных файлах. Информация о включенных в программу файлах и некоторые настройки хранятся в *проекте –* текстовом файле с расширением .pro.

Помимо проекта программа обычно включает заголовочные файлы с расширением .h, содержащие объявления типов, переменных, классов интерфейсных элементов и предметной области, процедур и функций. Реализация объявленных в заголовочном файле методов классов и подпрограмм обычно осуществляется в файлах с расширением .cpp. По рекомендациям языка С++ пара файлов с одним именем и расширениями .h и .cpp при этом образуют *псевдомодуль*.

При создании библиотеки *Qt* разработчики несколько расширили возможности языка *С++*, добавив к стандартному набору операторов, макросов и директив языка специальные директивы и макросы, которые обеспечивают передачу и обработку сигналов, хранение информации о типе времени выполнения и динамические свойства объектов и т.п. Если при создании графического интерфейса приложения используется визуальное проектирование графического интерфейса в дизайнере среды, то каждому классу формы соответствует еще файл с расширением **.**ui, содержащий *xml*-описание формы.

Реализацию механизмов библиотеки *Qt*, отсутствующих в языке *С++*, обеспечивает специализированный препроцессор *–* Метаобъектный компилятор (МОК) (от англ .Meta Object Compiler или MOC) . Этот препроцессор обрабатывает исходный текст программы, подставляя вместо специальных макросов *Qt* реализацию заказанных свойств на языке *С++* (рисунок 1).



**Рисунок 1** - Схема сборки приложения при наличии в нем макросов Qt

Как следует из схемы, MOC обрабатывает только заголовочные файлы, содержащие описание классов, наследуемых от классов виджетов *Qt*. В результате работы *МОС* в описание объявляемых классов включаются вспомогательные методы классов *Qt*. Реализация этих методов помещается в файл *moс*\_<Имя класса>.*cpp*, который подключается к проекту посредством автоматически добавляемой директивы #include. Тексты основных методов в файле реализации класса при этом не затрагивается.

После обработки исходного текста метакомпилятором *MOC* на выходе получается исходный текст программы на «чистом» *С++*. Окончательная компиляция и сборка программы осуществляется компилятором *C++* и компоновщиком, доступными в рамках платформы, на которых осуществляется компиляция-сборка.

## Основные интерфейсные компоненты библиотеки Qt

При создании класса формы в организуемый базовым классом контейнер добавляются визуальные компоненты, реализующие требуемую функциональность интерфейса.

В качестве базового класса для формы (окна) приложения обычно используют классы:

* QWidget – организует простейший контейнер формы, который применяется в простых интерфейсах, не предусматривающих меню;
* QMainWindow – формирует заготовку главного окна с меню и применяется при создании сложных интерфейсов;
* QDialog – является заготовкой окна диалога и, соответственно, предусматривает набор кнопок для выбора ответа.

Проанализировав особенности окна приложения, разработки выбирает нужный вариант базового класса.

В качестве визуальных компонентов для создания графических интерфейсов простейших программ чаще всего используют:

* метки QLabel – для вывода поясняющей информации на форму;
* однострочные текстовые редакторы (окна ввода-вывода) QLineEdit – для ввода данных или вывода результатов;
* кнопки QPushButton – для инициализации соответствующих действий программы.

При размещении виджетов в окне их необходимо настраивать. Настраиваются размеры, особенности функционирования, цвета фона и текста и т.д. Настройки можно выполнять при создании графического интерфейса в дизайнере – в Редакторе свойств дизайнера или, используя специальные методы, уже во время выполнения программы.

Настройки положения и размера обычно не выполняют, так как проще положение и размер виджетов задать, применяя компоновщики.

Также следует помнить, что все классы библиотеки виджетов наследуются от класса QWidget непосредственно, либо через своих предков, поэтому существуют общие для большинства виджетов настройки. Наиболее важные их низ настраиваются в Редакторе свойств. К таким настройкам, относятся:

objectName:QString – имя объекта в программе, автоматически объекты именуются в зависимости от типа, но со строчной буквы и с учетом номера в интерфейсе, так первая метка получит имя label, а начиная со второй, метки будут именоваться label\_2, label\_3 и т.д. Автоматически сформированное имя можно изменить в Редакторе свойств, но не из программы;

enabled:bool – определяет возможность принятия сигналов от клавиатуры и мышки: true – активный виджет, false – пассивный;

font– настройки шрифта;

text:QString – текст в поле виджета (метки, содержимое окна ввода-вывода, название кнопки и т.п.);

alignment – определяет, как располагается текст в поле виджета по горизонтали и вертикали: прижат к левой границе, правой или находится посередине, аналогично прижат к верху, к низу или находится посередине.

Для настройки этих и многих других свойств во время выполнения программыиспользуют методы, позволяющие читать или менять свойства:

void [setVisible](https://doc.qt.io/qt-6/qlabel.html#text-prop)(bool)– метод установки видимости виджета: true – виджет виден, false – скрыт;

QString text()const – метод чтения строки из поля text;

void [setText](https://doc.qt.io/qt-6/qlabel.html#text-prop)(const QString &)– метод записи строки в свойство text;

void [setEnable](https://doc.qt.io/qt-6/qlabel.html#text-prop)(bool)- метод настройки активности компонента: true – активный виджет, false – пассивный, т.е. не может принимать сообщения от клавиатуры и мышки.

Кроме этого, для окон ввода-вывода дополнительно используют свойство:

readОnly:bool – определяет, может ли пользователь вводить данные в окно: true – нет, false – может,

и соответствующий метод:

void [setReadOnly](https://doc.qt.io/qt-6/qlineedit.html#readOnly-prop)(bool) – метод установки возможности ввода данных в окно: true – вводить нельзя, false – вводить можно.

Для окон ввода-вывода и кнопок также существенно возможность установки фокуса на виджет. Для этого используют метод:

void setFocus()– метод установки фокуса; окно ввода-вывода, на которое установлен фокус, будет при сброшенном флаге readOnly обрабатывать ввод с клавиатуры, а кнопка, на которую установлен фокус, будет считаться нажатой при нажатии клавиши Enter на клавиатуре.

В процессе работы приложение обрабатывает сигналы, поступающие от пользователя через его взаимодействие с виджетом. Основной сигнал, который может быть сгенерирован кнопкой – сигнал нажатия на кнопку clicked(). Основной сигнал окна ввода-вывода в режиме ввода – завершение редактирования (нажатие клавиши Enter) [editingFinished](https://doc.qt.io/qt-6/qlineedit.html#editingFinished)().

В таблице 1 для удобства применения собраны основные свойства и методы, используемые для объектов классов QWidget, QLabel, QLineEdit, QPushButton.

Таблица 1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | QWidget | QLabel | QLineEdit | QPushButton |
| **Свойства** | | | | |
| objectName | + | + | + | + |
| enables | + | + | + | + |
| text |  | + | + | + |
| readOnly |  |  | + |  |
| alignment |  | + | + | + |
| **Методы** | | | | |
| setVisible() |  | + | + | + |
| setEnable() |  |  | + | + |
| setFocus() |  |  | + | + |
| setReadOnly() |  |  |  | + |
| Слоты | | | | |
| setText() |  | + | + |  |
| clear() |  | + | + |  |
| setNum() |  |  | + |  |
| **Сигналы** | | | | |
| clicked() |  |  |  | + |
| textEdited() |  |  | + |  |
| [**editingFinished**](https://doc.qt.io/qt-6/qlineedit.html#editingFinished)**()** |  |  | + |  |

# Задание

## Часть 1 Простейший калькулятор

***Задание***. *Следуя пояснениям ниже, разработать приложение Калькулятор, выполняющее основные арифметические действия над вводимыми числами.*

Для упрощения разработки не будем создавать на форме калькулятора кнопки цифр. Предусмотрим только окно ввода-вывода, кнопки очистки, операций сложения, вычитания, умножения, деления и получения результата, а также кнопку выхода (рисунок 2).



**Рисунок 2** – Эскиз окна

приложения

В процессе ввода число для контроля должно высвечивать в окне ввода. Ввод числа должен завершаться нажатием кнопки любой арифметической операции или операции получения результата. В последнем случае в окне ввода-вывода должен высвечиваться результат выполненных операций, а в остальных – введенное число должно запоминаться, а в окне должен отображаться ввод второго операнда.

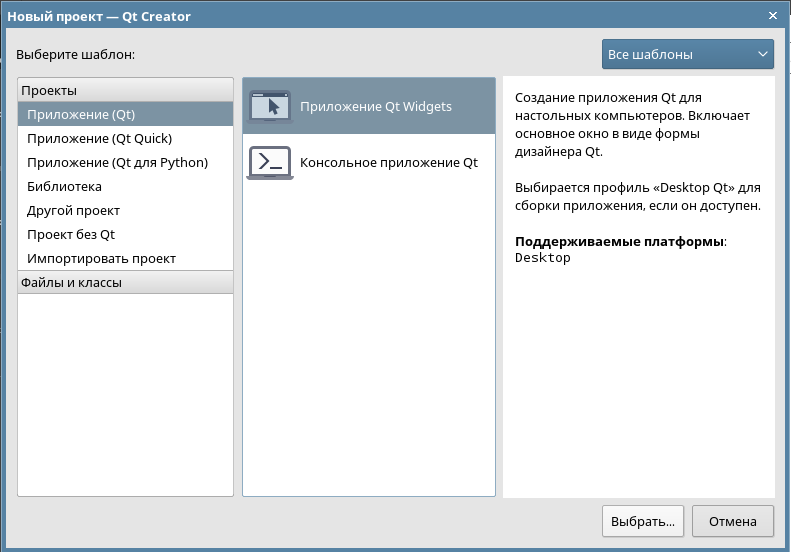
Кроме этого, предполагается, что, получив результат, пользователь может нажать кнопку любой арифметической операции (при этом результат сохранится в качестве первого операнда) и продолжить вычисления.

### 2.1.1 Ход разработки приложения в среде Qt Creator

1. *Создание нового проекта*. Проект можно создать двумя способами:

* нажав кнопку **+Создать** на странице приглашения среды;
* выбрав пункт меню **Файл \ Создать файл или проект...**

После нажатия кнопки или выбора пункта меню на экране появится окно диалога выбора шаблона проекта **Новый проект – Qt Creator** (рисунок 3), в котором надо выбрать тип проекта **Приложение (Qt)** и его подвид **Приложение Qt Widgets** и нажать кнопку **Выбрать**.



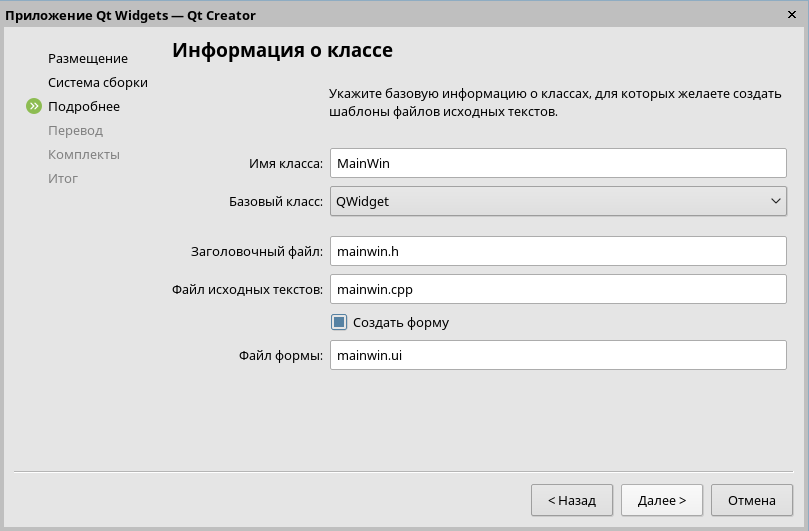
**Рисунок 3** – Выбор типа проекта и подвида проекта

В следующем окне вводим название проекта, например, Project1 и его местоположение, например, /home/user/projects и нажимаем кнопку **Далее>**.

На экране появляется окно выбора системы сборки. Как и при работе с консольным приложением выбираем *qmake* и опять нажимаем **Далее>**.

В следующем окне требуется выбрать базовый класс для главного окна приложения (рисунок 4). По умолчанию указан класс QMainWindow, однако для простого приложения без меню лучше использовать класс QWidget.

В качестве имени класса вводим MainWin и выбираем вариант **Создать форму**. Имена создаваемых заголовочного файла, файла исходных текстов и файла формы среда сформирует автоматически. После чего опять нажимаем кнопку **Далее>**.



**Рисунок 4** – Выбор базового класса для класса окна приложения и его имени

Следующее окно – окно **Файл переводов** содержит настройки, используемые для создания приложений со сменой языка интерфейса. В этом окне нажимаем **Далее>**.

В окне выбора комплекта оставляем комплект ***Desktop*** – настольное приложение и нажимаем **Далее>**.

Последнее окно создания приложения – окно **Управление проектом**, в нем перечислены файлы, создаваемые в папке **Project1**:

**/home/user/projects/Project1:**

**Project1.pro**

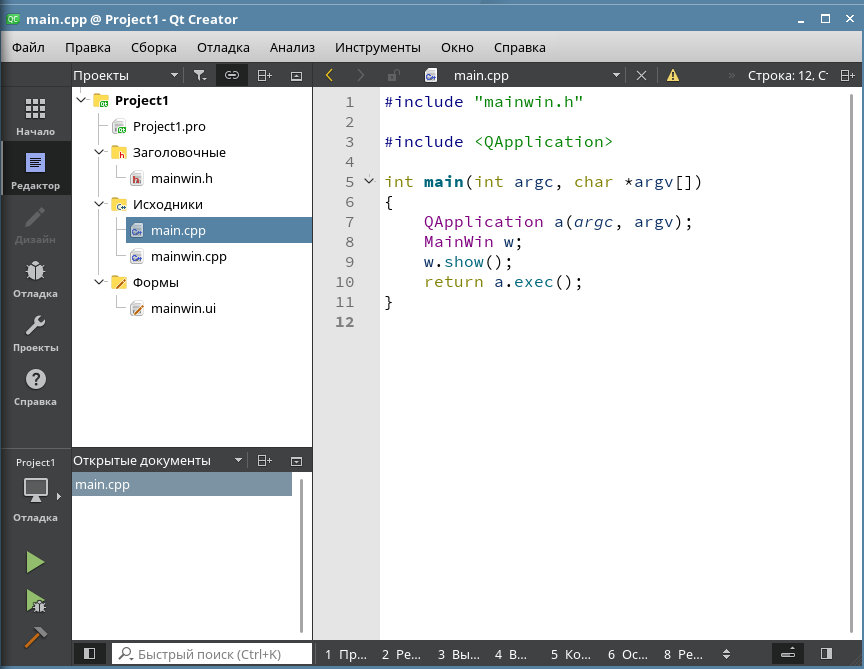
**main.cpp**

**mainwin.cpp**

**mainwin.h**

**mainwin.ui**

Нажав кнопку **Завершить**, получаем заготовку приложения, которая открывается на странице заготовки основной программы (рисунок 5).



**Рисунок 5** – Заготовка приложения (основная программа)

Код основной программы предусматривает: создание объекта-приложения, создание объекта класса главного окна, визуализацию главного окна и запуск цикла обработки сообщений.

Заголовочный файл mainwin.h содержит заготовку описания класса окна:

**#ifndef MAINWIN\_H**

**#define MAINWIN\_H**

**#include <QWidget>**

**QT\_BEGIN\_NAMESPACE**

**namespace Ui { class MainWin; }**

**QT\_END\_NAMESPACE**

**class MainWin : public QWidget**

**{**

**Q\_OBJECT**

**public:**

**MainWin(QWidget \*parent = nullptr);**

**~MainWin();**

**private:**

**Ui::MainWin \*ui;**

**};**

**#endif // MAINWIN\_H**

Как следует из текста заготовки, файл содержит объявление пространства имен Ui, в котором объявлен класс MainWin. Описание этого класса будет создаваться при визуальном проектировании интерфейса и будет содержать описание окна со всеми виджетами, в него входящими.

Основной класс окна также с именем MainWin объявлен в главном пространстве имен приложения ::. В этом классе предусмотрен указатель на объект класса окна из пространства имен Ui:

**Ui::MainWin \*ui;**

а также инициализирующий конструктор и деструктор. Конструктор традиционно отвечает за инициализацию поля ui (выделение памяти под объект и его инициализацию), а деструктор – освобождает динамическое поле.

Файл реализации mainwin.cpp содержит заготовки конструктора и деструктора:

**#include "mainwin.h"**

**#include "ui\_mainwin.h"**

**MainWin::MainWin(QWidget \*parent)**

**: QWidget(parent),ui(new Ui::MainWin)**

**{**

**ui->setupUi(this);**

**}**

**MainWin::~*MainWin*()**

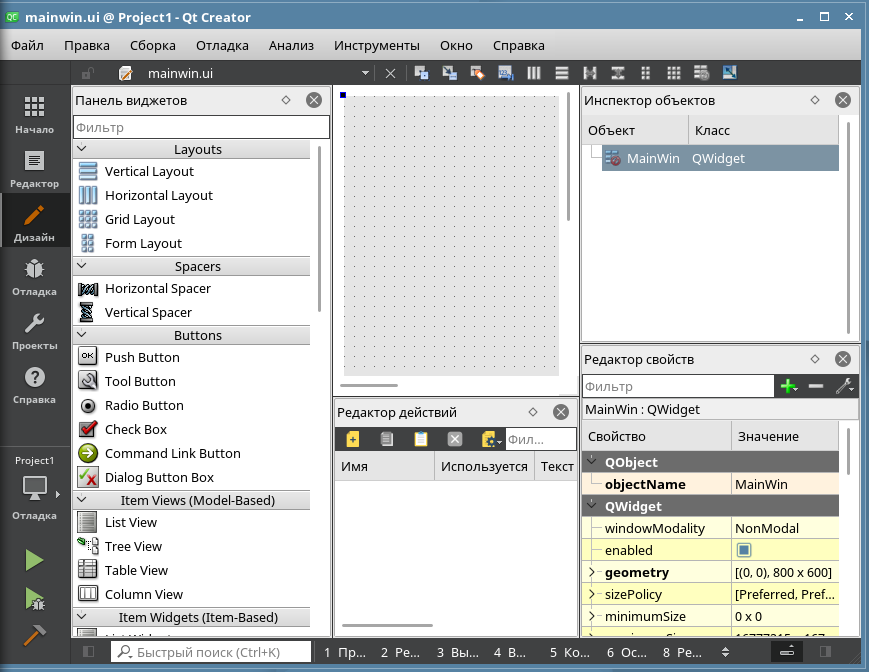
**{**

**delete ui;**

**}**

Конструктор, помимо вызова конструктора базового класса QWidget, выделяет память под включаемый через указатель объект описания окна и инициализирует его, вызывая метод setupUI(). Деструктор освобождает выделенную память.

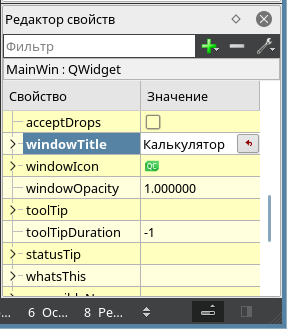
*2. Визуальное проектирование главного окна приложения.* При переключении на файл формы **.**ui мы переходим в режим Дизайнера и получаем возможность визуального проектирования интерфейса (рисунок 6).



**Рисунок 6** – Вкладка визуального проектирование главного окна приложения

В окне Дизайнера слева находится Панель виджетов, с которой можно мышкой «брать» нужные компоненты для установки на форму. Справа сверху располагается Инспектор объектов, используемый для переключения на требуемый виджет, справа внизу – Редактор свойств, позволяющий настраивать свойства виджета, выбранного в Инспекторе объектов, в центре сверху – проектируемая форма и в центре внизу – Редактор действий.

Прежде всего, зададим имя приложения, которое должно высвечивать в заголовке главного окна. Для этого при выбранном в Инспекторе объектов объекте MainWin в Редакторе свойств находим свойство windowTitle и вводим имя **Калькулятор** (рисунок 7).



**Рисунок 7** – Задание имени приложения

*Примечание*. При поиске свойства используется полоса прокрутки, расположенная в правой части окна.

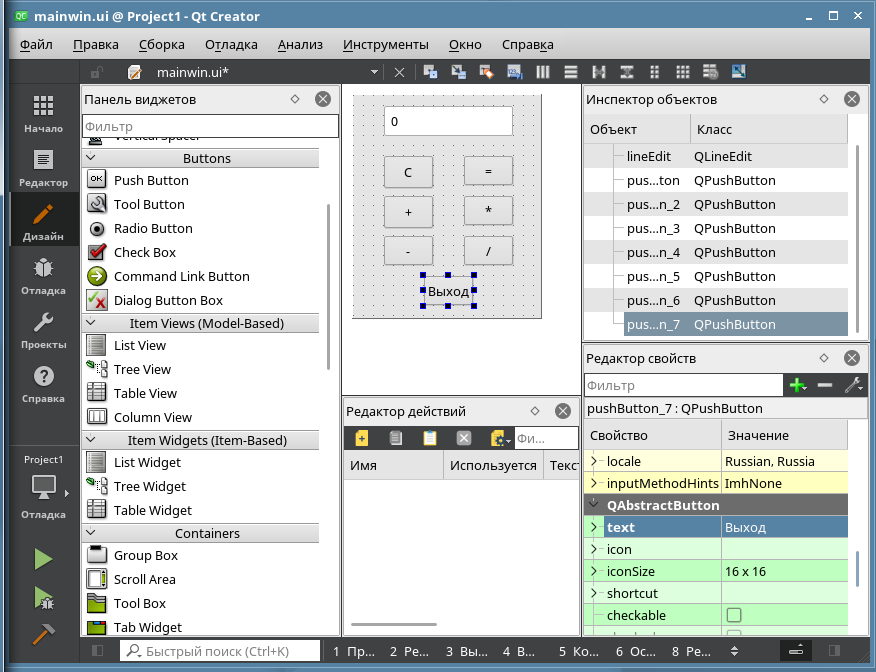
Затем, используя мышку, меняем размер формы и мышкой же устанавливаем на форме окно ввода-вывода чисел – объект класса QLineEdit и 7 кнопок – объектов класса QPushButton.

После этого в Инспекторе объектов появятся все перечисленные выше объекты с автоматически сформированными средой именами объектов.

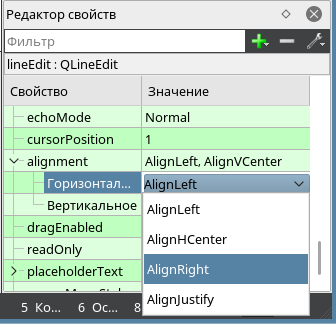
Формирование имен выполняется по следующим правилам: первый объект получает совпадающее с именем класса имя, но без буквы Q и записанное со строчной буквы, например, lineEdit, а последующие – то же имя, но с номером на конце, начиная с 2, например, pushButton\_2. При желании имена объектов-виджетов можно изменить в Редакторе свойств, задавая соответствующее значение свойству objectName.

Для задания названия кнопок и текста в поле ввода виджета окна ввода-вывода используется свойство text. Задавая это свойство в соответствии с проектируемым окном, получаем форму, показанную на рисунке 8.

Для наглядности в окно ввода введен 0, это позволяет увидеть, что числа в поле ввода будут прижиматься к левой границе, что непривычно. Перенос 0 вправо в окне ввода-вывода выполняем, настраивая свойство выравнивания alignment (рисунок 9).



**Рисунок 8** – Настройка внешнего вида кнопок



**Рисунок 9** – Настройка выравнивания по правой границе при выводе текста

3. *Компоновка виджетов.* Компоновка виджетов необходима для обеспечения автоматического изменения размеров и взаимоположения визуальных компонентов при изменении размеров формы. С этой целью в библиотеке Qt используются специальные компоненты – *компоновщики*. Дизайнер позволяет программисту выбрать нужный вариант компоновки: по горизонтали, по вертикали, в два ряда или по сетке.

Компонуем виджеты следующим образом:

1) выделяем кнопки «С» и «=» и нажимаем кнопку Скомпоновать по горизонтали (см. панель инструментов в верхней области окна);

2) выделяем кнопки «+» и «\*» и нажимаем кнопку Скомпоновать по горизонтали;

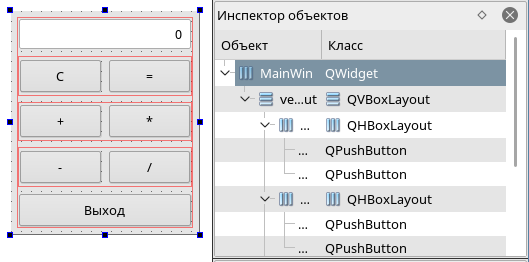
3) выделяем кнопки «-» и «/» и нажимаем кнопку Скомпоновать по горизонтали;

4) выделяем все виджеты и компоновщики и нажимаем кнопку Скомпоновать по вертикали (там же);

5) выделяем форму и нажимаем кнопку Скомпоновать по вертикали (или по горизонтали).

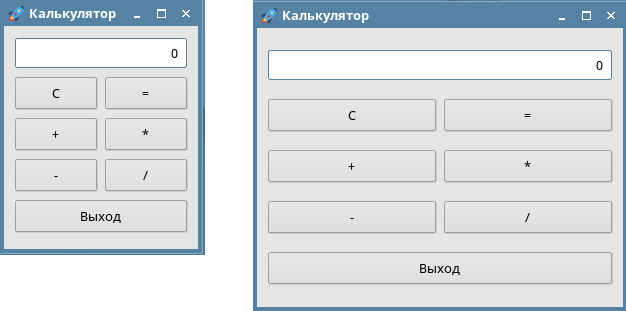
Последнее действие связывает форму с компоновщиками.

Результат компоновки показан на рисунке 10.



**Рисунок 10** – Компоновка приложения

Полученное приложение можно запустить на выполнение. После указанной выше компоновки можно менять размеры окна, что приведет к соответствующему масштабированию виждетов (рисунок 11).



**Рисунок 11** – Вид формы приложения при его запуске и изменении размера

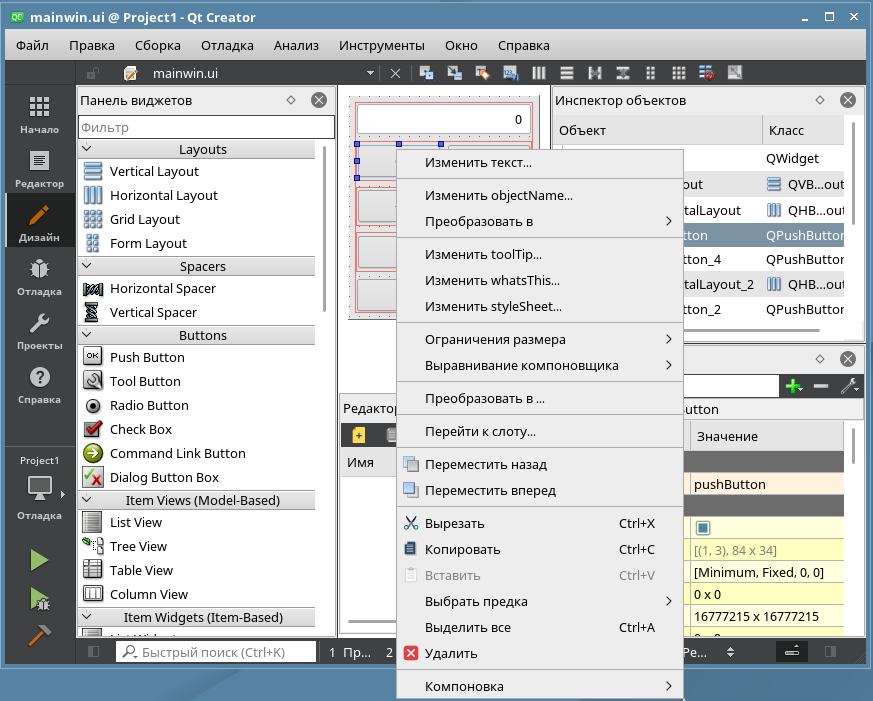
Естественно приложение еще «не умеет» делать ничего полезного, однако обработка нажатия стандартных кнопок, таких как Закрыть приложение («x») вверху справа, уже заложена в базовом классе главного окна.

4. *Создание слотов*

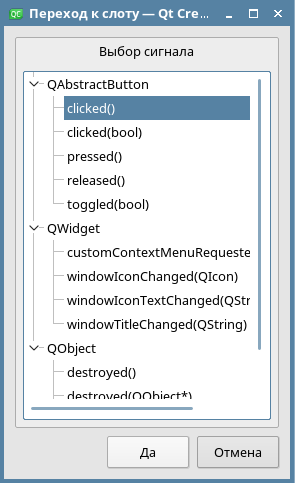
Для обеспечения требуемой функциональности необходимо создать обработчики сигналов и связать их с сигналами – нажатием кнопок. Заготовку слота, связанного с сигналом, можно получить, щелкнув правой кнопкой мышки по кнопке, нажатие которой необходимо обработать, и в появившемся контекстном меню выбрать пункт **Перейти к слоту...** (рисунок 12).

В ответ на экране появится окно выбора сигнала **Переход к слоту** (рисунок 13). Нас интересует сигнал clicked(), т.е. нажатие кнопки.

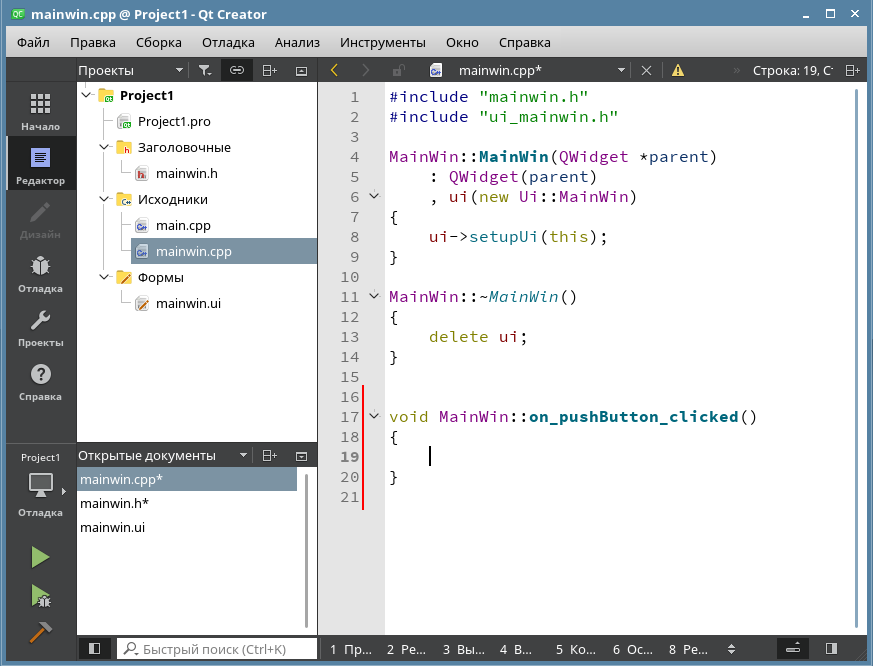
После нажатия кнопки **Да** в окне выборасреда демонстрирует нам заготовку слота обработки нажатия кнопки в файле mainwin.cpp после текстов конструктора и деструктора объектов класса MainWin (рисунок 14).



**Рисунок 12** – Контекстное меню виджета кнопки



**Рисунок 13** – Выбор обрабатываемого сигнала кнопки



**Рисунок 14** – Заготовка слота нажатия кнопки

Тело слота по логике работы приложения должно содержать очистку строки ввода, обнуление результата и отменять ранее сохраненную операцию, устанавливая фокус на окно ввода. Соответственно необходимо определить переменную для хранения накопленного результата res и переменную для запоминания символа операции oper. Обе переменные должна быть доступны из всех методов класса MainWin главного пространства имен, поэтому их можно объявить в секции private этого класса:

private:

Ui::MainWin \*ui;

**double res; QChar oper;**

Указанные переменные необходимо инициализировать в конструкторе.

Кроме этого в конструктор следует добавить установку фокуса на окно ввода, чтобы сразу после запуска калькулятора можно было вводить числа:

MainWin::MainWin(QWidget \*parent)

: QWidget(parent),

ui(new Ui::MainWin),

**res(0.0), oper('@')**

{

ui->setupUi(this);

**ui->lineEdit->setFocus();**

}

С учетом вышесказанного и необходимости следить за установкой курсора в окно ввода слот нажатия кнопки «С» необходимо определить как:

void MainWin::on\_pushButton\_clicked()

{

**ui -> lineEdit -> clear();**

**ui -> lineEdit -> setFocus();**

**res = 0.0; oper = '@';**

}

В соответствии с заданием по нажатию на кнопки «+», «-», «\*», «/», «=» должно выполняться следующее:

* если переменная oper содержит символ **'@'**, то следует получить и запомнить введенное до нажатия кнопки число в переменной res,
* иначе – необходимо выполнить операцию из переменной oper над переменной res и последним введенным числом,

запомнить новую операцию и, если нажата клавиша «=», то соответствующим образом изменить значение в окне результата на форме:

void MainWin::on\_pushButton\_2\_clicked()

{

**double r = ui->lineEdit->text().toDouble();**

**if (oper == '+') res+=r;**

**else**

**if (oper == '-') res-=r;**

**else**

**if (oper == '\*') res\*=r;**

**else**

**if (oper == '/') res/=r;**

**else res = r;**

// определяем виджет, отправившый сигнал – sender()

**QPushButton\* buttonSender =**

**qobject\_cast<QPushButton\*>(sender());**

**oper = (buttonSender->text())[0];**

**QString str;**

**if (oper == '=')**

**ui->lineEdit->setText(str.setNum(res));**

**else ui->lineEdit->setText("");**

**ui->lineEdit->setFocus();**

}

Заготовку для этого слота можно создать, используя контекстное меню одной из перечисленных кнопок, однако вызываться он должен по нажатию любой кнопки, поэтому необходимо связать их нажатия со слотом.

5. *Связывание сигналов и слотов*

Связать сигналы со слотами можно, добавив в конструктор класса макросы connect:

**connect(ui->pushButton,SIGNAL(clicked(bool)),**

**this,SLOT(on\_pushButton\_clicked()));**

**connect(ui->pushButton\_2,SIGNAL(clicked(bool)),**

**this,SLOT(on\_pushButton\_2\_clicked()));**

**connect(ui->pushButton\_3,SIGNAL(clicked(bool)),**

**this,SLOT(on\_pushButton\_2\_clicked()));**

**connect(ui->pushButton\_4,SIGNAL(clicked(bool)),**

**this,SLOT(on\_pushButton\_2\_clicked()));**

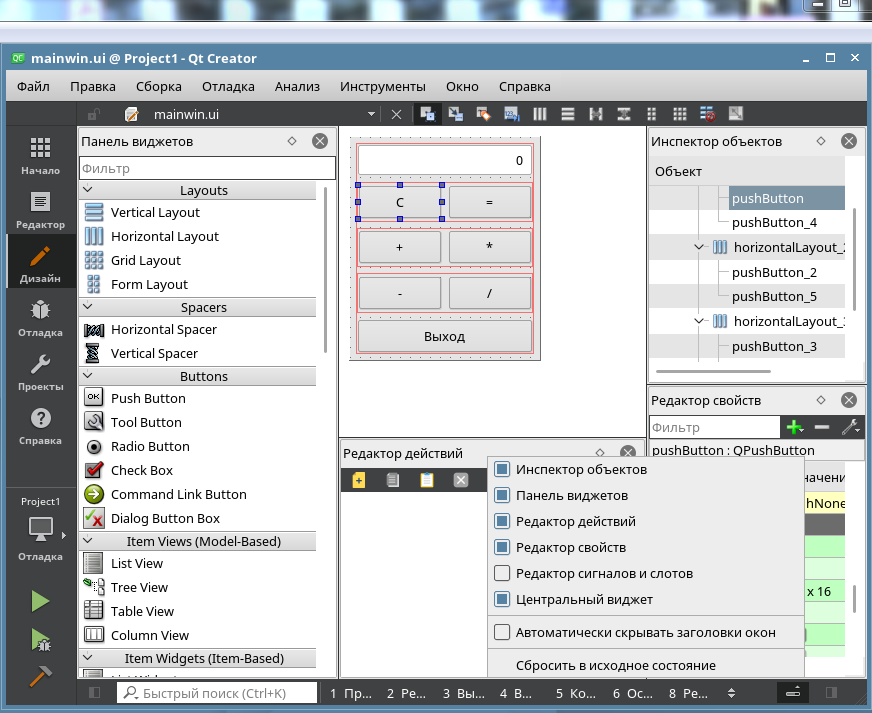
**connect(ui->pushButton\_5,SIGNAL(clicked(bool)),**

**this,SLOT(on\_pushButton\_2\_clicked()));**

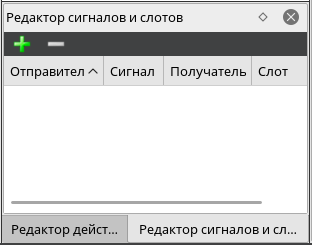
**connect(ui->pushButton\_6,SIGNAL(clicked(bool)),**

**this,SLOT(on\_pushButton\_2\_clicked()));**

Для связывания сигнала от кнопки «Выход» со стандартным слотом закрытия окна close(), также можно использовать Редактор сигналов и слотов. Чтобы этот редактор появился в окне Дизайнера, следует щелкнуть правой клавишей мышки по заголовку панели Редактор действий. В ответ появится настройка (рисунок 15), в которой следует выбрать Редактор сигналов и слотов, после чего соответствующая панель высветится внизу в середине окна Дизайнера (рисунок 16).



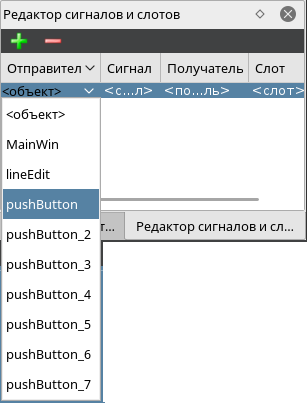
**Рисунок 15** – Получение доступа к Редактору сигналов и слотов



**Рисунок 16** – Редактор сигналов и слотов

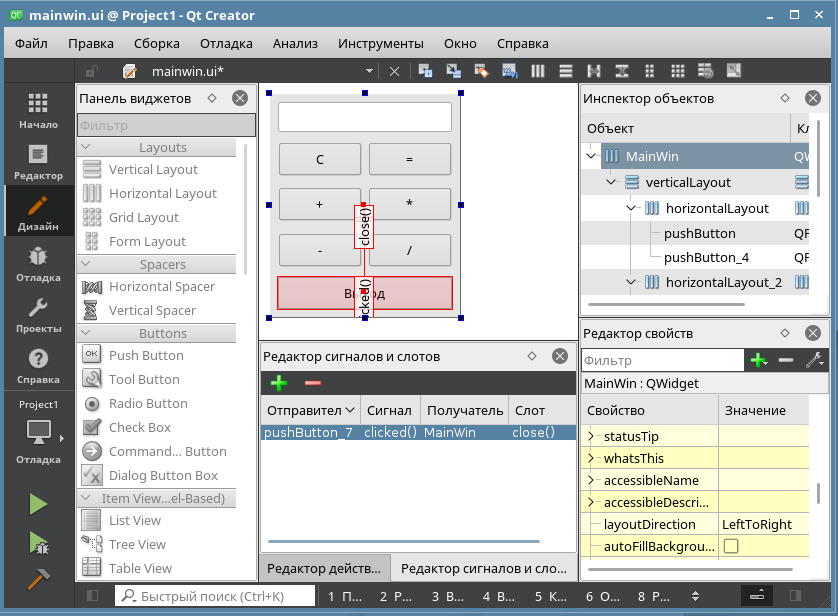
Для добавления связи в этом редакторе используется зеленый плюс. В каждом поле появившейся строки описания связи двойным щелчком мышки вызываем выпадающие списки и последовательно выбираем:

* отправителя (рисунок 17) – кнопку *pushButton*,
* сигнал *clicked()*,
* получателя *MainWin*,
* слот *close()*.



**Рисунок 17** – Выбор отправителя сигнала

После чего в режиме Изменение сигналов и слотов, подключаемом через меню **Правка**, можно увидеть установленную связь (рисунок 18).



**Рисунок 18** – Связывание сигнала со слотом

6. *Запуск приложения на выполнение*

Запускаем программу на выполнение и убеждаемся, что мы получили работоспособное «настольное» приложение.

Для контроля ниже приведены исходные файлы программы:

- файл проекта Project1.pro;

- файл основной программы main.cpp;

- заголовочного файла окна mainwin.h;

- файла реализации окна mainwin.cpp.

Текст файла окна, созданного дизайнером mainwin.ui, приводить не имеет смысла, так как он создавался автоматическ.

Файл проекта Project1.pro *Qt Creator* построил автоматически:

QT += core gui

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

CONFIG += c++11

DEFINES += QT\_DEPRECATED\_WARNINGS

SOURCES += main.cpp mainwin.cpp

HEADERS += mainwin.h

FORMS += mainwin.ui

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

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

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

Файл основной программы main.cpp с момента создания не изменился:

#include "mainwin.h"

#include <QApplication>

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

{

QApplication a(argc, argv);

MainWin w;

w.show();

return a.exec();

}

Заголовочный файл окна mainwin.hвыглядит следующим образом:

#ifndef MAINWIN\_H

#define MAINWIN\_H

#include <QWidget>

QT\_BEGIN\_NAMESPACE

namespace Ui { class MainWin; }

QT\_END\_NAMESPACE

class MainWin : public QWidget

{

Q\_OBJECT

public:

MainWin(QWidget \*parent = nullptr);

~MainWin();

private slots:

void on\_pushButton\_clicked();

void on\_pushButton\_2\_clicked();

private:

Ui::MainWin \*ui;

double res;

QChar oper;

};

#endif // MAINWIN\_H

Файл реализации mainwin.cpp содержит тексты конструктора и деструктора:

#include "mainwin.h"

#include "ui\_mainwin.h"

MainWin::MainWin(QWidget \*parent)

: QWidget(parent),ui(new Ui::MainWin),

res(0.0),oper('@')

{

ui->setupUi(this);

ui->lineEdit->setFocus();

connect(ui->pushButton,SIGNAL(clicked(bool)),

this,SLOT(on\_pushButton\_clicked()));

connect(ui->pushButton\_2,SIGNAL(clicked(bool)),

this,SLOT(on\_pushButton\_2\_clicked()));

connect(ui->pushButton\_3,SIGNAL(clicked(bool)),

this,SLOT(on\_pushButton\_2\_clicked()));

connect(ui->pushButton\_4,SIGNAL(clicked(bool)),

this,SLOT(on\_pushButton\_2\_clicked()));

connect(ui->pushButton\_5,SIGNAL(clicked(bool)),

this,SLOT(on\_pushButton\_2\_clicked()));

connect(ui->pushButton\_6,SIGNAL(clicked(bool)),

this,SLOT(on\_pushButton\_2\_clicked()));

connect(ui->pushButton\_7,SIGNAL(clicked(bool)),

this,SLOT(close()));

}

MainWin::~MainWin()

{

delete ui;

}

void MainWin::on\_pushButton\_clicked()

{

ui->lineEdit->setText("");

ui->lineEdit->setFocus();

res = 0.0; oper = '@';

}

void MainWin::on\_pushButton\_2\_clicked()

{

double r = ui->lineEdit->text().toDouble();

if (oper == '+') res+=r;

else

if (oper == '-') res-=r;

else

if (oper == '\*') res\*=r;

else

if (oper == '/') res/=r;

else res = r;

QPushButton\* buttonSender =

qobject\_cast<QPushButton\*>(sender());

oper = (buttonSender->text())[0];

QString str;

if (oper == '=')

ui->lineEdit->setText(str.setNum(res));

else ui->lineEdit->setText("");

ui->lineEdit->setFocus();

}

### 2.1.2 Требования к отчету

Отчет должны иметь титульный лист, на котором указывается:

а) наименование факультета и кафедры;

б) название дисциплины;

в) номер и тема домашнего задания;

г) фамилия преподавателя, ведущего занятия;

д) фамилия, имя и номер группы студента.

Отчет должен содержать:

1) текст задания;

2) диаграмму состояний интерфейса;

3) диаграмму классов приложения;

4) текст программы с комментариями;

5) вручную посчитанные тесты;

4) результаты, полученные при выполнении программы;

5) выводы.

Все записи в отчете должны быть либо напечатаны на принтере, либо разборчиво выполнены от руки синей или черной ручкой (карандаш – не допускается). Схемы и таблицы также должны быть напечатаны при помощи компьютера или нарисованы с использованием чертежных инструментов, в том числе карандаша.

### 2.1.3 Требования к защите

Защита лабораторной предполагает ответы на вопросы преподавателя. При этом работа считается защищенной, если студент правильно и уверенно отвечает на 2-3 вопроса преподавателя, может указать в программе те или иные операции, внести изменения в программу, а также поясняет работу указанного фрагмента.

### 2.1.4 Контрольные вопросы

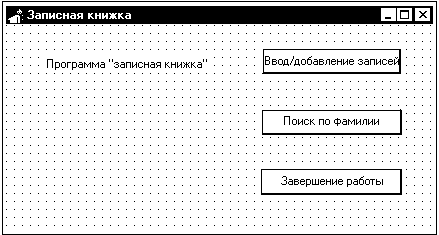
1. Какие файлы создаются при создании проекта приложения с графическим интерфейсом для визуального проектирования интерфейса?
2. Какие панели включает Дизайнер и для чего они используются?
3. Как регулируются размеры заготовки главного окна?
4. Для чего используются компоновщики, и какие виды компоновщиков доступны в Дизайнере?
5. Как получить заготовку слота?
6. Как выполняется связывание сигналов и слотов?
7. Какая информация представлена на диаграмме состояний интерфейса?
8. Какая схема строится при проектировании классов? Какая информация на ней представлена?
9. Зачем нужна диаграмма компоновки приложения? Какие обозначения на ней используются?

## Часть 2 Записная книжка

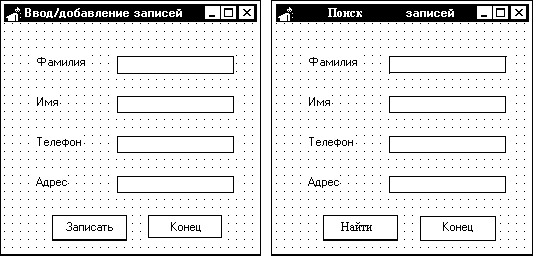
***Задание.*** *Используя Дизайнер среды Qt Creator,**разработать приложение «Записная книжка».*

*Приложение должно обеспечивать запись и хранение телефонов абонентов пользователя, поиск абонента по фамилии или имени с учетом возможности наличия информации об абонента с одинаковыми именами или фамилиями, удаление записей о заданных абонентах.*

*Примерный вид окон приложения представлен на рисунках 19 и 20.*



**Рисунок 19**  – Эскиз главного окна приложения



**Рисунок 20 –** Эскизыокон дополнения и поиска записей

*Возможность удаления записей и переход к следующей найденной записи встроить в форму поиска.*

*Примечание*. В случае возникновения трудностей обратитесь к приложению А, которое содержит возможный текст более простого варианта предложенного проекта.

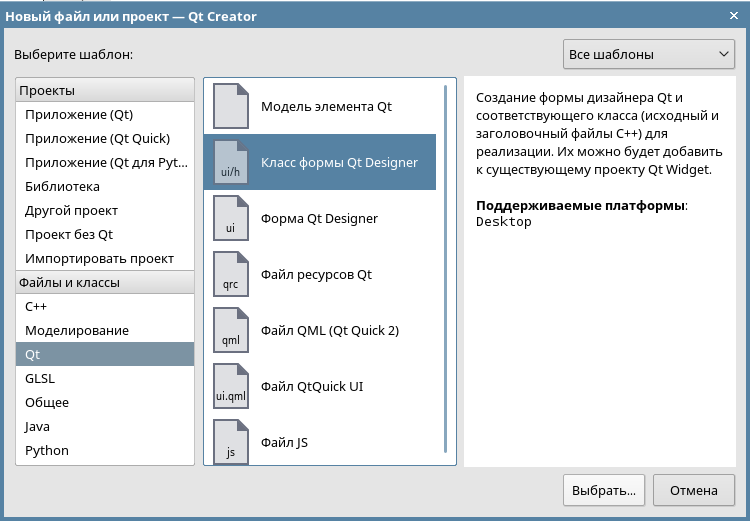
### 2.2.1 Особенности работы с многооконным приложением

При создании многооконного приложения программист сталкивается с необходимостью

1) описать несколько классов-форм;

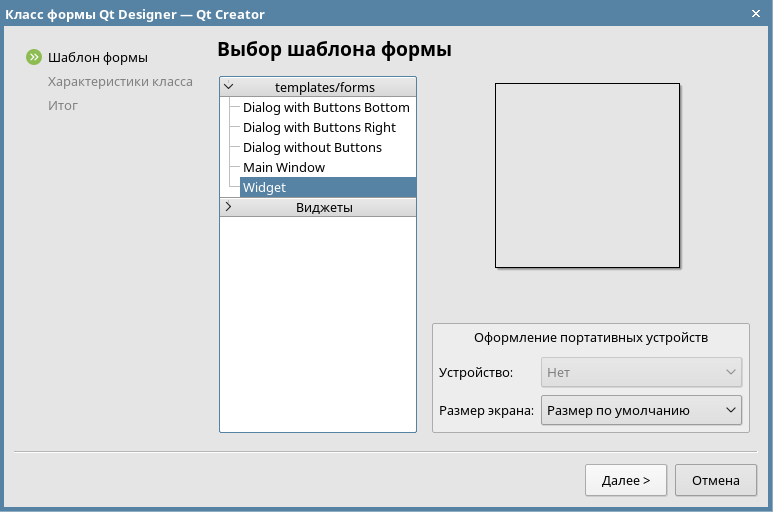
2) создавать объекты классов-форм, визуализировать их, передавать им управление и возвращать его обратно.

Для создания дополнительных классов-форм можно использовать пункт меню **Файл/Создать файл или проект...** и выбрать в появившемся окне Новый Файл или проект – Qt Creator пункты Qt и Класс формы Qt Designer (рисунок 21).

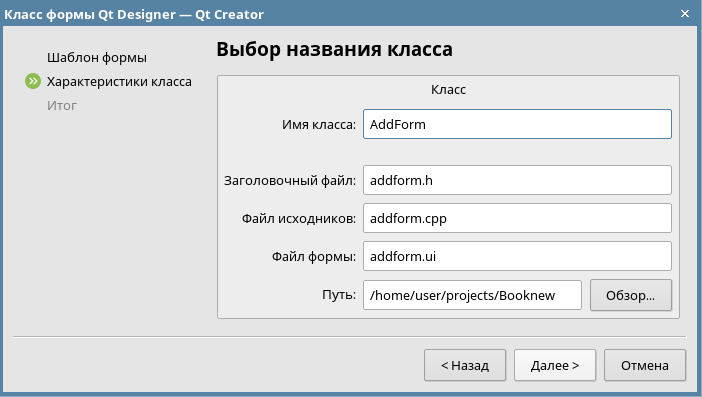


**Рисунок 21** – Создание класса-формы

Затем последовательно необходимо выбрать шаблон формы (рисунок 22) и название класса (рисунок 23).

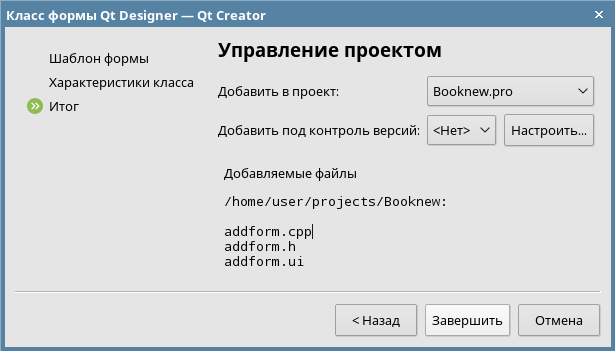


**Рисунок 22** – Выбор шаблона формы



**Рисунок 23** – Выбор названия класса

Завершающее окно создания класса формы подтверждает создание трех файлов, описывающих проект (рисунок 24).



**Рисунок 23** – Выбор названия класса

В перечисленных файлах создаются заготовки класса формы, его конструктора и деструктора, а также описания формы окна.

Вид окна конструируется с помощью Дизайнера и дополняется слотами, обеспечивающими работоспособность формы.

Переменную класса формы удобнее всего объявлять в секции private главного окна, например, так:

private:

Ui::Widget \*ui;

**AddForm add;**

Для того чтобы это не вызвало синтаксических ошибок, при этом нужно не забыть включить заголовочный файл формы в заголовочный файл главного окна:

**#include <addform.h>**

Добавленное в класс главного окна объектное поле add доступно всем методам этого класса, соответственно, разрабатывая слот кнопки вызова дополнительной формы можно указать:

void Widget::on\_pushButton\_clicked()

{

**add.show();**

}

После чего вторая форма появится поверх первой.

Интерес представляет собой также работа с файлом. Выбор библиотек для выполнения операций остается на усмотрение студента.

### 2.2.2 Требования к отчету

Отчет должны иметь титульный лист, на котором указывается:

а) наименование факультета и кафедры;

б) название дисциплины;

в) номер и тема домашнего задания;

г) фамилия преподавателя, ведущего занятия;

д) фамилия, имя и номер группы студента.

Отчет должен содержать:

1) текст задания;

2) реализованные формы интерфейса;

3) диаграммы состояний интерфейса для переключения окон и для окон добавления и поиска записей;

4) диаграммы классов интерфейса для тех же форм;

5) результаты, полученные при выполнении программы;

6) выводы.

Все записи в отчете должны быть либо напечатаны на принтере, либо разборчиво выполнены от руки синей или черной ручкой (карандаш – не допускается). Схемы и таблицы также должны быть напечатаны при помощи компьютера или нарисованы с использованием чертежных инструментов, в том числе карандаша.

### 2.2.3 Требования к защите

Защита лабораторной предполагает ответы на вопросы преподавателя. При этом работа считается защищенной, если студент правильно и уверенно отвечает на 2-3 вопроса преподавателя, может указать в программе те или иные операции, внести изменения в программу, а также поясняет работу указанного фрагмента.

### 2.2.4 Контрольные вопросы

1. Зачем в проекте используются заголовочные файлы?
2. Как создается класс формы? Какие файлы при этом используются?
3. Где объявляется переменная класса формы? Почему именно в этом месте?
4. Как визуализировать дополнительную форму приложения? Как ее спрятать?
5. Если дополнительную форму включать в главную через указатель, то в каких местах программы можно выделять под нее память?

# Литература

1. Иванова Г.С. C++. Часть 3. Создание графических интерфейсов пользователя с использованием библиотеки Qt 4.7: Учебное пособие. – М.: МГТУ им. Н.Э. Баумана, 2014. – 52 с. – URL: https://e-learning.bmstu.ru/iu6/course/view.php?id=322.
2. Иванова Г. С., Ничушкина Т. Н. Объектно-ориентированное программирование: учебник для вузов / Иванова Г. С., Ничушкина Т. Н. ; общ. ред. Иванова Г. С. - М.: Изд-во МГТУ им. Н. Э. Баумана, 2014. - 455 с. : ил. - ISBN 978-5-7038-3921-8. – URL: https://ibooks.ru/bookshelf/364125.
3. Шлее М. Qt 5.10. Профессиональное программирование на С++. СПб.: БХВ-Петербург, 2018. 1072 с.: ил. - ISBN 978-5-9775-3678-3 – URL: https://ibooks.ru/bookshelf/358881.