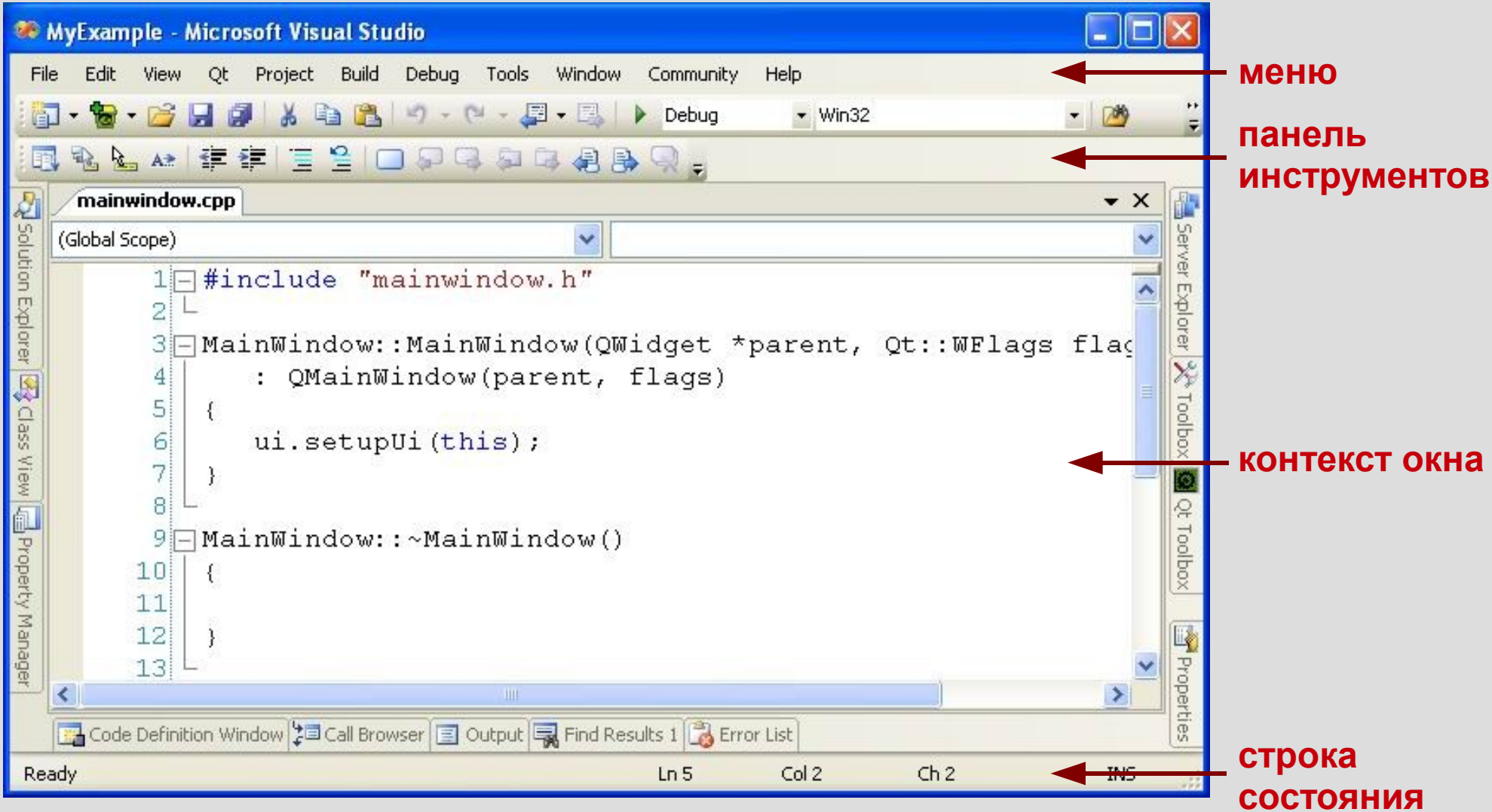
Организация многооконного интерфейса

- Понятие главного и диалогового окна программы
- Понятие модального и немодального окна
- Стандартные диалоги
- Создание диалогового окна
- Способ передачи данных между окнами

Понятие главного окна программы

- Большинство современных программ имеет одно главное окно и несколько диалоговых окон.
- Главное окно отличается от диалоговых тем, что имеет меню, панель инструментов и строку состояния.
- Оно появляется сразу после запуска программы, а его закрытие приводит к завершению всего приложения.

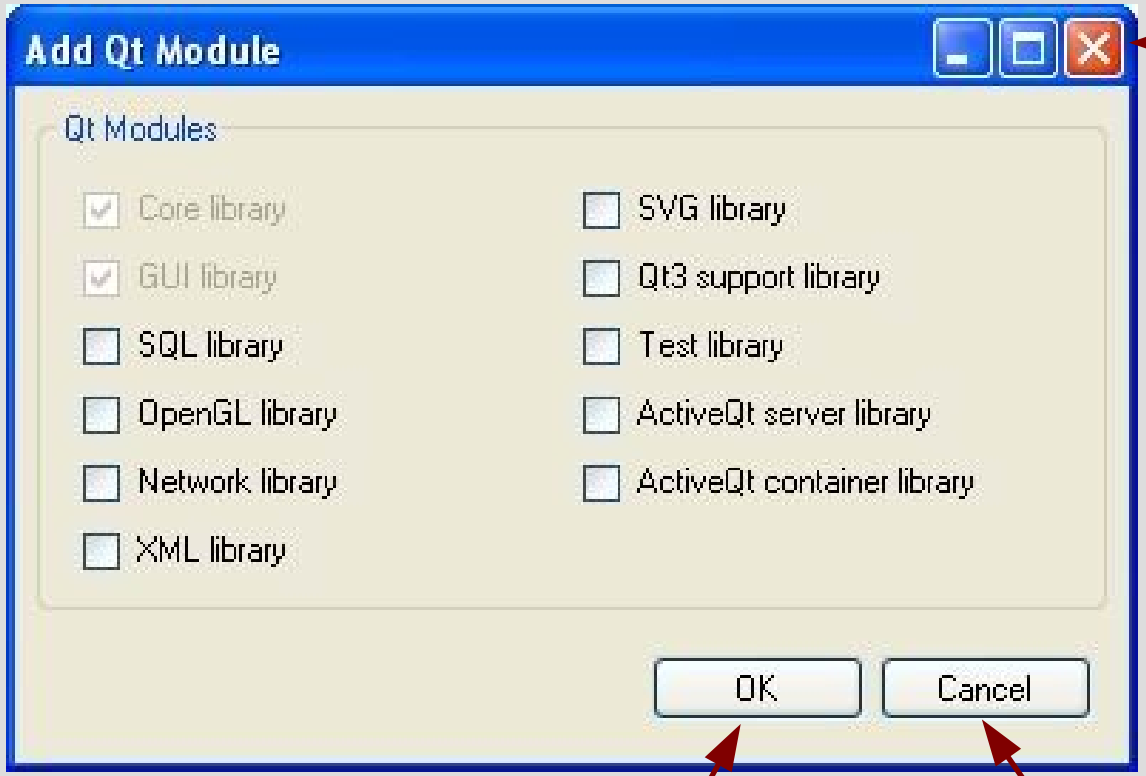
Понятие главного окна программы



Понятие диалогового окна программы

- При необходимости (например, с целью задания параметров программы или вывода сообщений) в программе могут появляться диалоговые окна.
- Диалоговые окна имеют одну или несколько кнопок, с помощью которых пользователь подтверждает или отменяет действия, сделанные им в диалоге.

Понятие диалогового окна программы



кнопка отмены

кнопка подтверждения + кнопка по умолчанию

кнопка отмены

Понятие диалогового окна программы

- Одна из кнопок диалога может быть кнопкой по умолчанию, что обеспечивает следующее поведение: если фокус расположен не на кнопке и нажата клавиша **Enter**, то диалоговое окно ведет себя так, как будто нажата кнопка по умолчанию.
- Для закрытия диалога с отменой выполненных действий также используется клавиша **Esc**.

Понятие немодального окна

- В зависимости от того может ли пользователь работать сразу с несколькими окнами, различают модальные и не модальные окна.
- Немодальное окно – это окно, которое можно «покинуть», не закрывая его.
- В результате пользователь получает возможность работать с несколькими окнами, переключаясь между ними.

Пример немодального окна

- Диалог используется для поиска подстроки в документе, при этом пользователь может переходить от диалога к документу и наоборот без всяких ограничений.

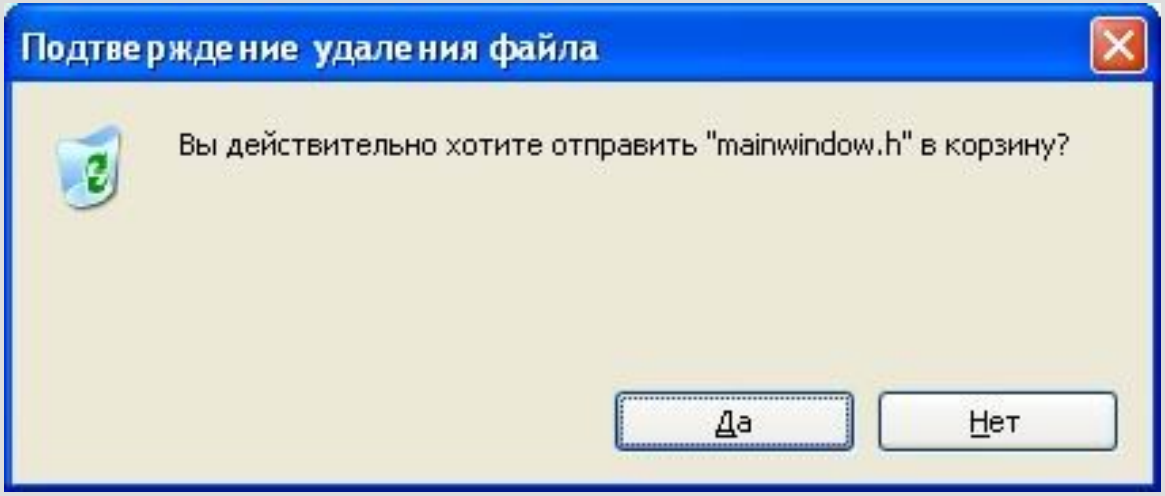
Диалог «**Найти и заменить**»

Понятие модального окна

- Модальное окно прерывает работу приложения и для продолжения работы окно должно быть закрыто.
- Такое окно обычно используется для осуществления действий, которые требуют обязательной реакции пользователя: только после того как пользователь выполнит необходимые действия, он сможет перейти к другому действию (в том числе вернуться к предыдущему).

Пример модального окна

- Диалог, который возникает при удалении файла: пока пользователь не подтвердит или не отменит удаление файла, никакие другие действия невозможны

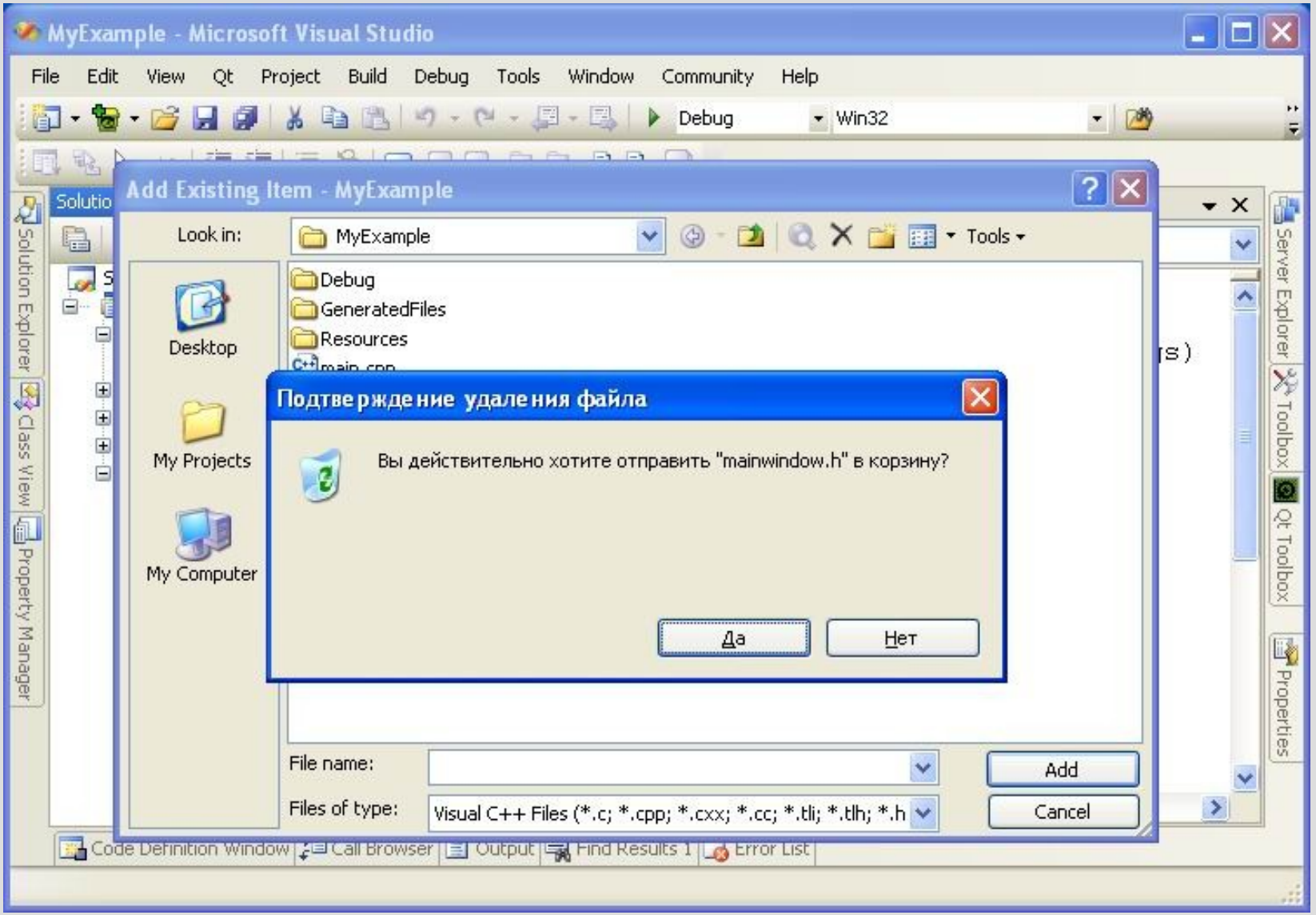


Диалог «**Подтверждение удаления файла**»

Стековая организация окон

- В программе одновременно могут использоваться как модальные, так и немодальные окна.
- Однако самым простым вариантом реализации многооконного интерфейса является стековая организация окон.
- При таком подходе все окна являются модальными, а пользователь всегда работает с «верхним» окном.

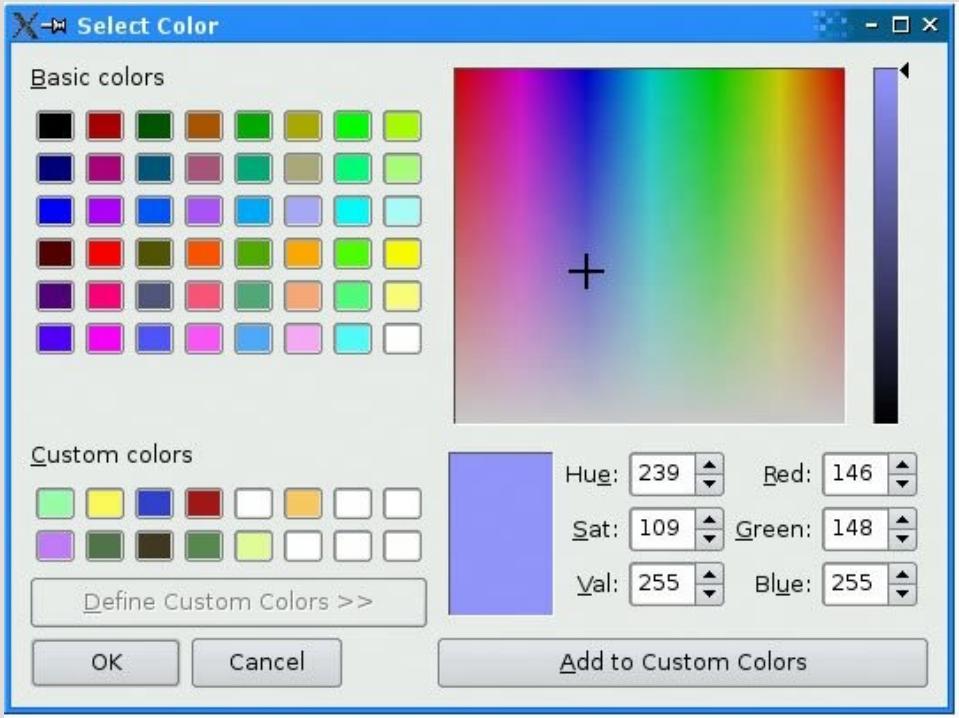
Пример стековой организации окон



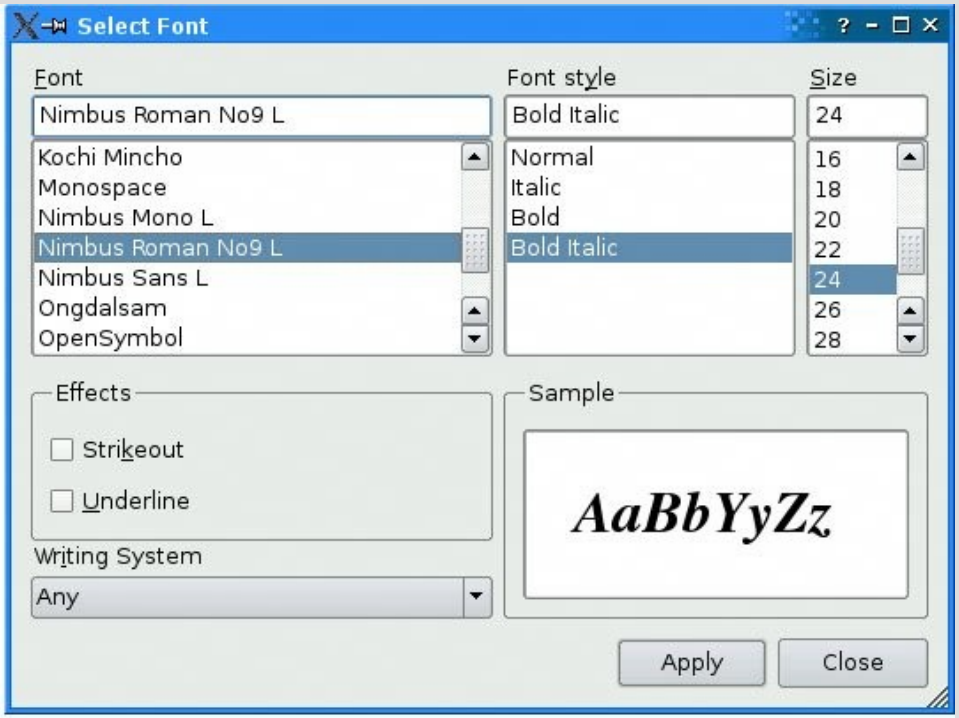
Стандартные диалоговые окна

- Использование стандартных диалоговых окон значительно ускоряет разработку приложений.
- В **QT Library** реализованы следующие стандартные диалоги:
  - диалоговое окно выбора цвета
  - диалоговое окно выбора шрифта
  - диалоговое окно настройки принтера
  - диалоговое окно выбора файлов
  - диалоговые окна «обратной связи»

Диалоговые окна выбора цвета и шрифта

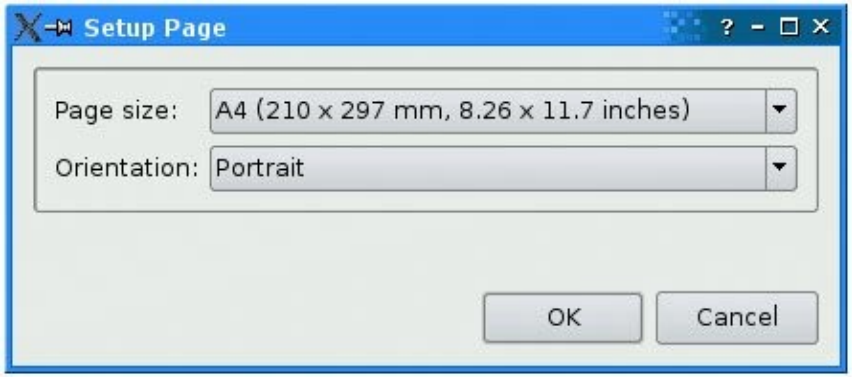


QColorDialog

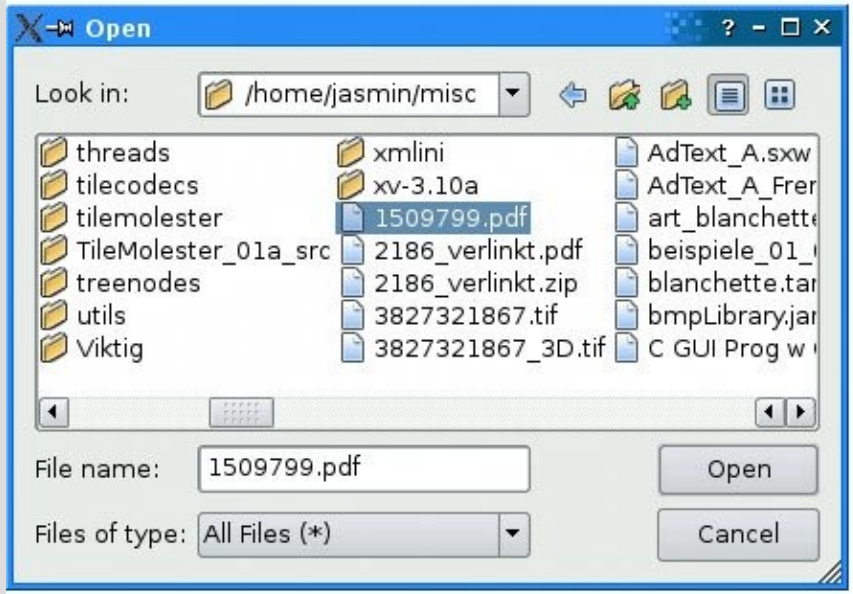


QFontDialog

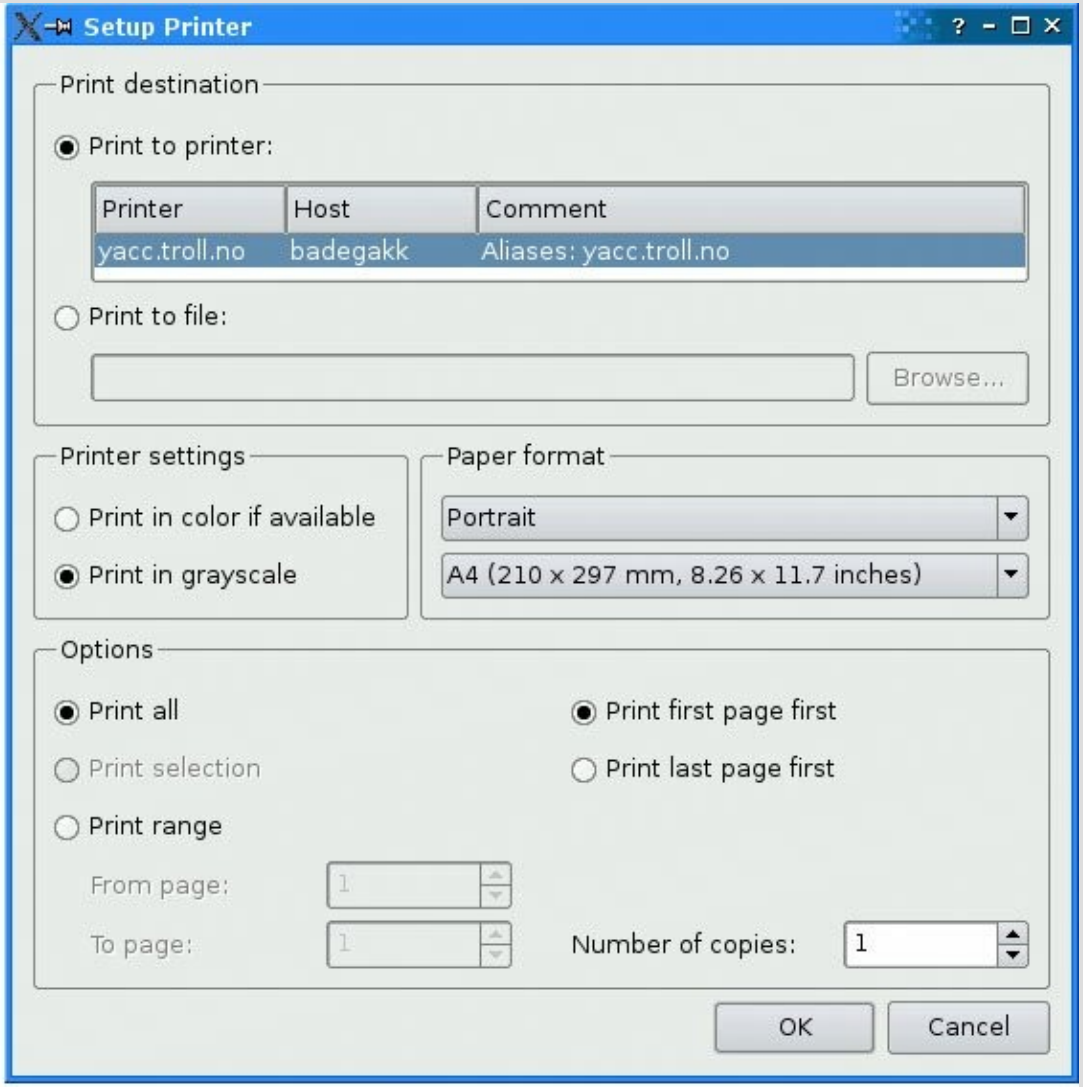
Диалоговые окна настройки принтера и выбора файлов



QPageSetupDialog



QFileDialog



QPrintDialog

Диалоговые окна «обратной связи»

- Диалоговое окно ввода строки или числа (модальное окно) — **`QInputDialog`**
- Окно сообщений (модальное окно) — **`QMessageBox`** — отображает текстовое сообщение и ожидает реакции со стороны пользователя

Диалоговые окна «обратной связи»

- Окно сообщения об ошибке (немодальное окно) — **`QErrorMessage`**
- Диалоговое окно прогресса (модальное окно) — **`QProgressDialog`** — информирует пользователя о начале продолжительной операции и дает ему возможность визуально оценить время работы.

Диалоговые окна «обратной связи»

QInputDialog

QProgressDialog

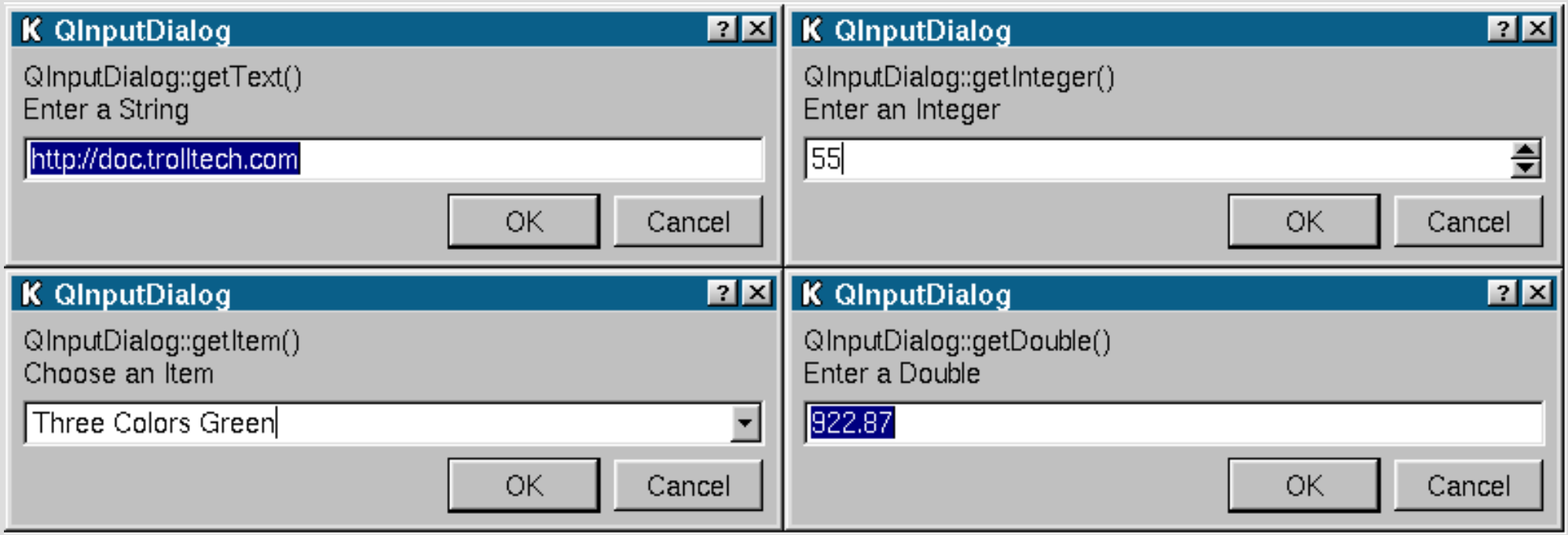
QMessageBox

QErrorMessage

Диалоговое окно ввода данных `QInputDialog`

- Для создания и отображения окна используются 4 статических метода класса:
  - **`getText()`** - для ввода текста
  - **`getInteger()`** - для ввода целых чисел
  - **`getDouble()`** - для ввода чисел с плавающей точкой двойной точности
  - **`getItem()`** - для выбора элемента из списка строк

Диалоговое окно ввода данных
QInputDialog



Диалоговое окно ввода данных `QInputDialog`

- Каждый из методов возвращает запрашиваемое значение, если пользователь подтвердил свои действия (нажал кнопку «OK»)
- Для того чтобы определить, подтвердил или отменил пользователь свои действия , используется параметр, передаваемый в метод по адресу:

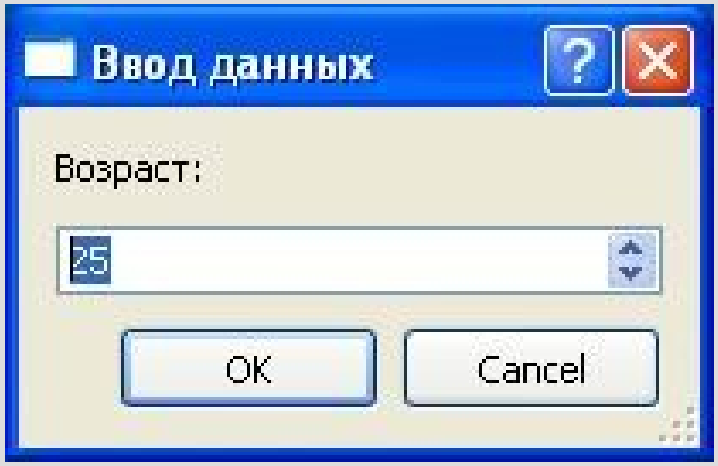
```
bool * ok
```

Задание

Создайте диалог для ввода возраста человека, если метод имеет следующий заголовок

```
int getInteger (
 QWidget * parent,          // окно-родитель
 const QString & title, // заголовок окна
 const QString & label, // поясняющий текст
 int value = 0,             // исходное значение
 int minValue = -2147483647,// мин. значение
 int maxValue = 2147483647, // макс.значение
 int step = 1,    // шаг изменения значения
 bool * ok = 0)   // признак подтверждения
```

Пример использования диалога для ввода целого числа




```
bool ok;

// Вводим возраст человека
int age = QInputDialog::getInteger(this,
                    "Ввод данных","Возраст:",
                    25, 0, 100, 1, &ok);

// Если пользователь ничего не ввел, то
// задаем неопределенное значение
if (ok) age = -1;
```

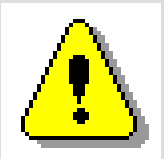
Диалоговое окно выбора файлов

- Класс `QFileDialog` отвечает за создание и работоспособность сразу трех диалоговых окон:
  - диалогового окна выбора файла для открытия
  - диалогового окна выбора места и файла для его сохранения
  - диалогового окна для выбора директории

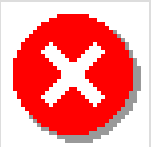
Диалоговое окно отображения сообщений `QMessageBox`

- Для создания и отображения окна используются 4 статических метода класса:

- **`information()`** - для вывода информационного сообщения



- **`warning()`** - для вывода предупреждающего сообщения



- **`critical()`** - для вывода критического сообщения

- **`question()`** - для вывода вопроса

Диалоговое окно отображения сообщений `QMessageBox`

- В каждый из методов передается выводимое сообщение и набор отображаемых кнопок
- Каждый метод возвращает индекс выбранной пользователем кнопки

Задание

Создайте диалог для запроса у пользователя подтверждения на удаление текущей записи. Кнопка по умолчанию - «Нет». Используйте метод:

```
StandardButton question (
  QWidget * parent,          // окно-родитель
  const QString & title,     // заголовок окна
  const QString & text,      // текст вопроса
  StandardButtons buttons,// набор кнопок
  StandardButton defaultButton)// кнопка по
                             // умолчанию
```

Задание

Константы для кнопок:

`QMessageBox::Yes` — кнопка «Да»

`QMessageBox::No` — кнопка «Нет»

Набор кнопок задается через побитовую операцию «ИЛИ»:

`<кнопка>|<кнопка>|<кнопка>`

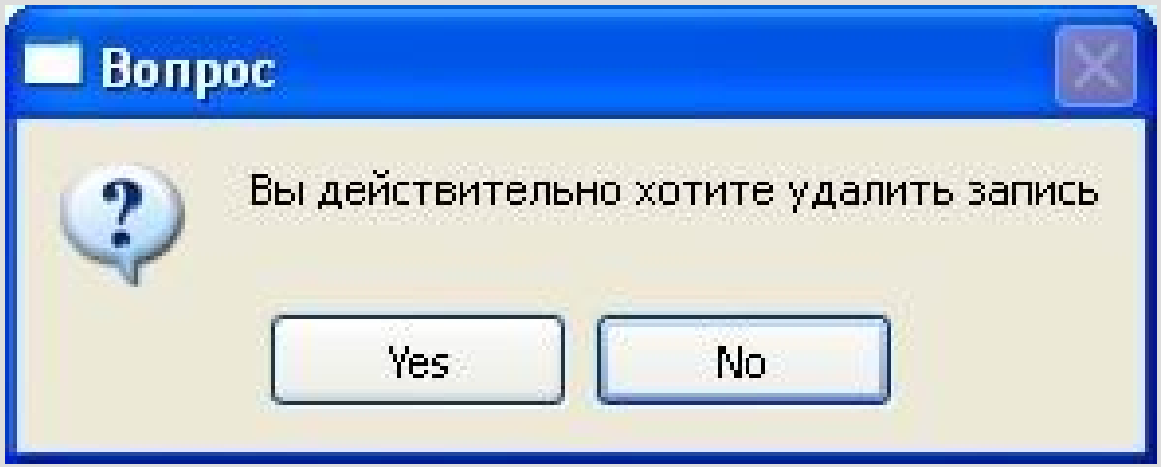
Пример использования диалога для задания вопроса

```
// кнопка, выбранная пользователем
QMessageBox::StandardButton btnAnswer;

// Задаем вопрос
btnAnswer = QMessageBox::question(this,
    "Вопрос",
    "Вы действительно хотите удалить запись",
    QMessageBox::Yes | QMessageBox::No,
    QMessageBox::No);

// Если пользователь готов удалить запись
if (btnAnswer == QMessageBox::Yes)
{ /* Удаляем запись */ }
```

Пример использования диалога для задания вопроса



Создание собственного диалога

- Если стандартные диалоги имеют недостаточную функциональность, то создаются собственные диалоги. Например, диалоги для задания настроек приложения.
- В общем случае работа с диалоговыми окнами включает в себя три шага:

  (1) создание макета диалога

  (2) задание поведения диалога

  (3) создание диалога в окне-родителе и «запуск» диалога

Создание макета диалога

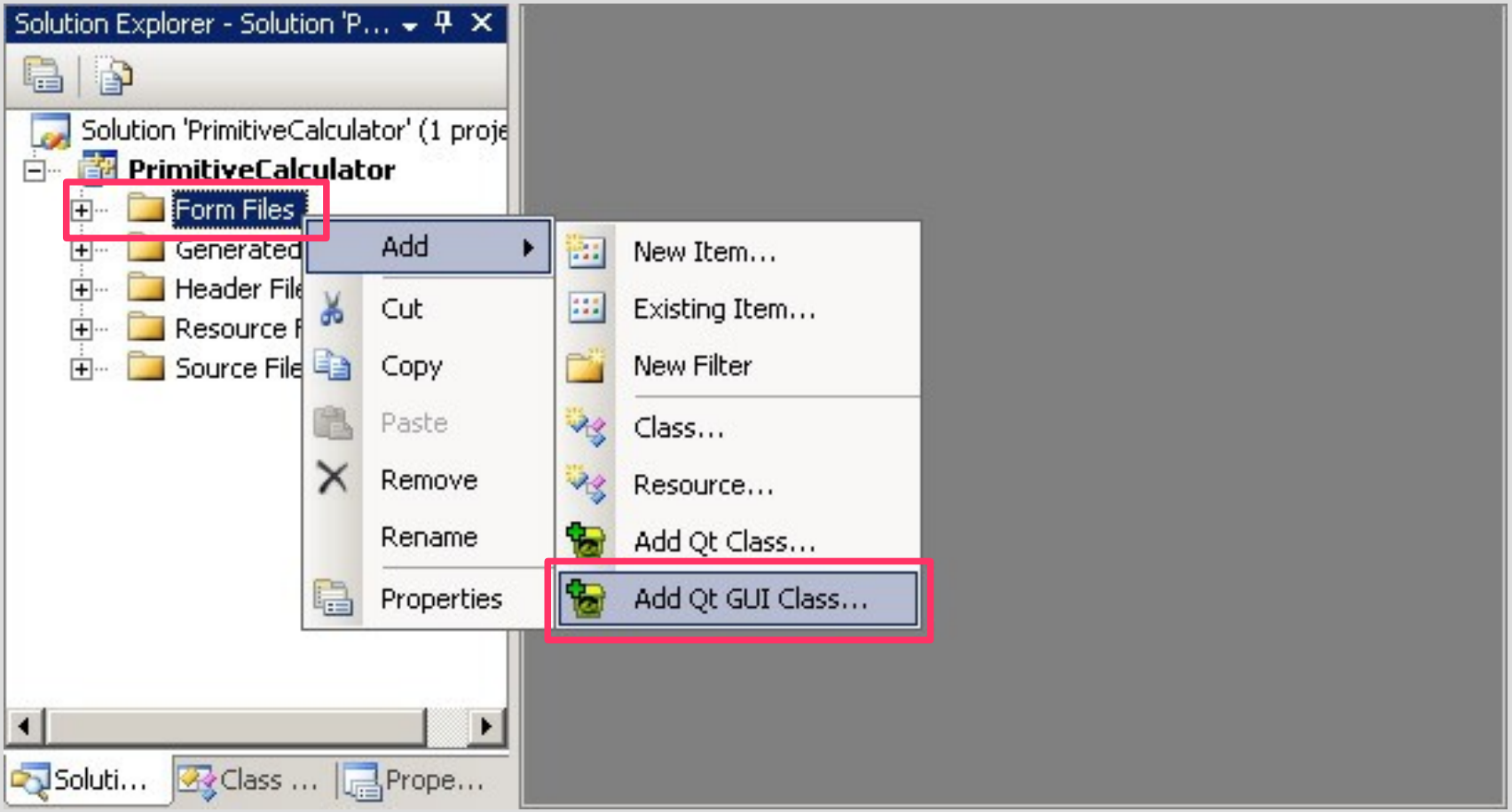
(1) Создаем новый GUI-класс, соответствующий диалоговому окну.

(2) Создаем макет диалогового окна с помощью редактора форм: размещаем на нем требуемые виджеты и задаем их основные свойства.

(3) Добавляем в диалоговое окно одну или несколько кнопок. Задаем одну кнопку по умолчанию.

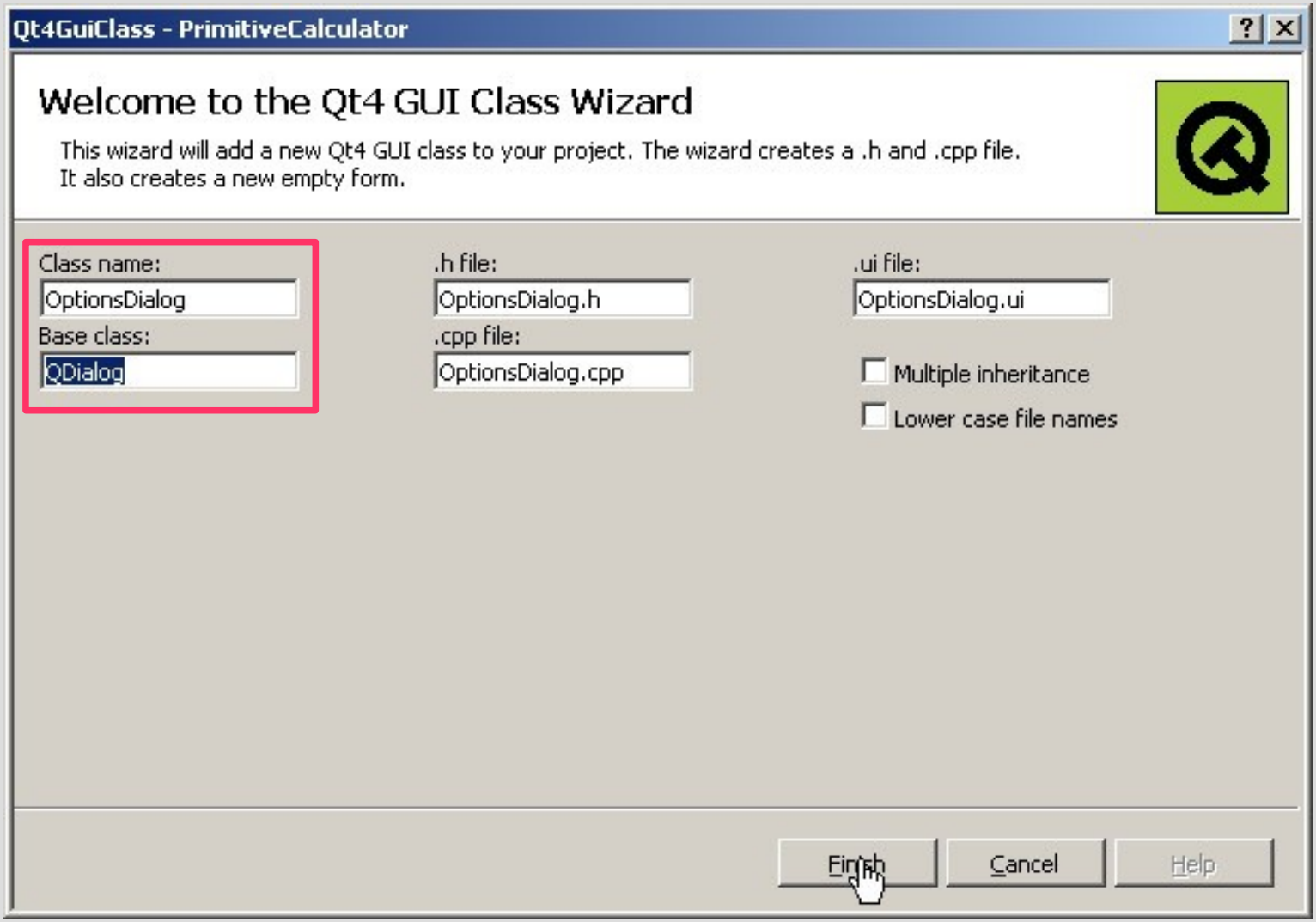
(4) Настраиваем кнопки на закрытие диалогового окна.

Создание нового GUI-класса диалогового окна



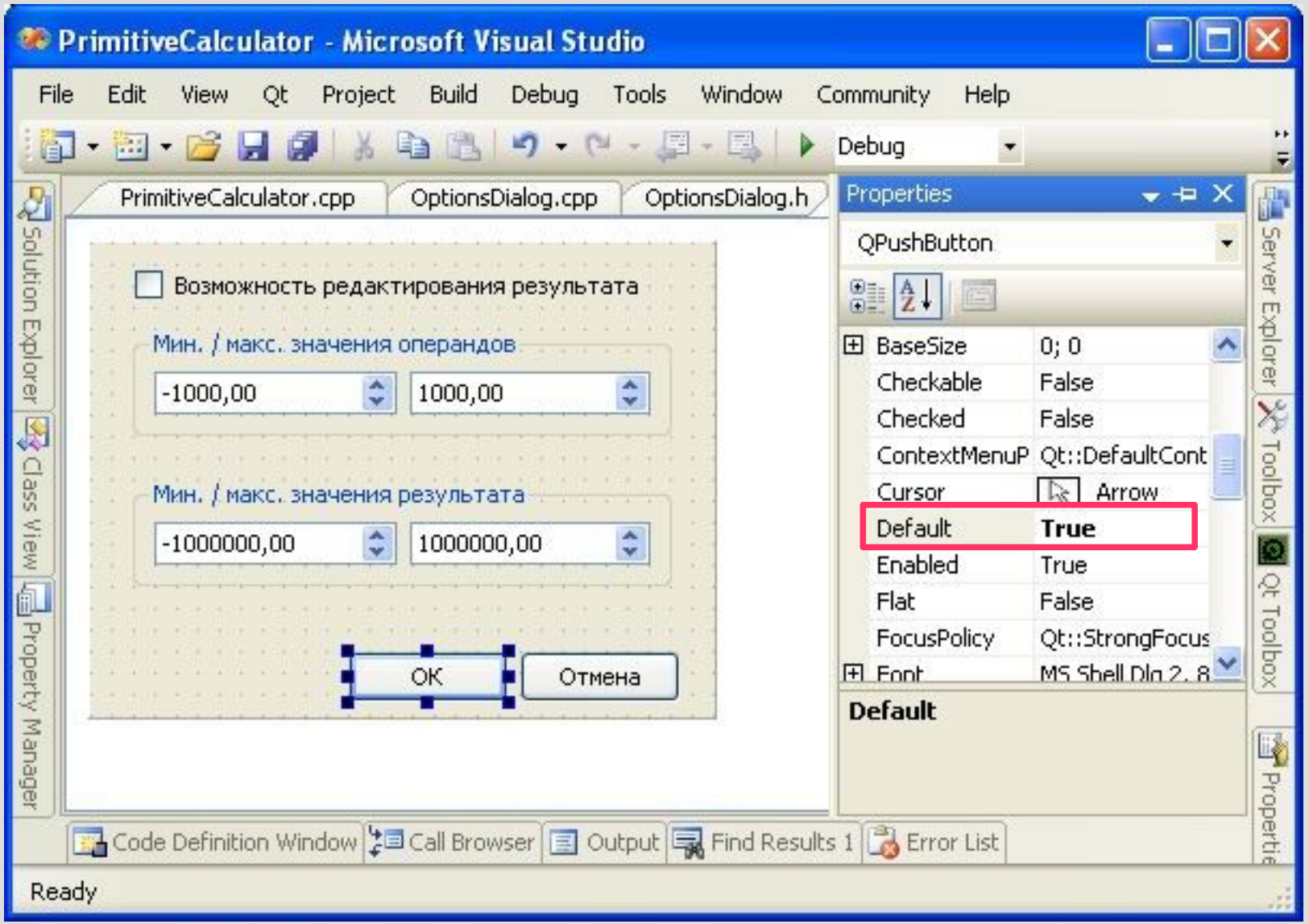
Начало создания GUI-класса диалогового окна

Создание нового GUI-класса диалогового окна



Завершение создания GUI-класса диалогового окна

Создание макета диалогового окна, задание кнопки по умолчанию



Закрытие диалогового окна

- Под закрытием диалога понимается его «исчезновение» с экрана. Однако, в памяти диалог продолжает существовать и может быть повторно «вызван» на экран.
- Диалог может быть «закрыт» разными способами:
  - без указания того, что пользователь подтвердил/опроверг свои действия (характерно для немодальных диалогов)
  - с подтверждением/опровержением действий (характерно для модальных диалогов)

Закрытие диалогового окна

- Для закрытия диалога используются стандартные слоты диалога:
  - **`bool close ()`** - закрывает диалог без признака того, что пользователь подтвердил или отменил свои действия
  - **`void accept ()`** - закрывает диалог с признаком того, что пользователь подтвердил свои действия
  - **`void reject ()`** - закрывает диалог с признаком того, что пользователь отменил свои действия

Настройка кнопок на закрытие диалогового окна

- Для закрытия диалога с помощью кнопок, необходимо «связать» кнопки со стандартными слотами диалога `close()`, `accept()` или `reject()`.
- Кнопки можно «связать» со слотами либо программным способом (с помощью метода `connect()`), либо визуальным способом (с помощью редактора форм)

Задание

Задайте в конструкторе диалогового окна связь между кнопкой «OK» и слотом **`accept()`** этого диалога, если известно, что:

- объектное имя кнопки **`btnOK`**;
- диалог задается классом **`OptionsDialog`**

Подсказка: в результате нажатия кнопки испускается сигнал

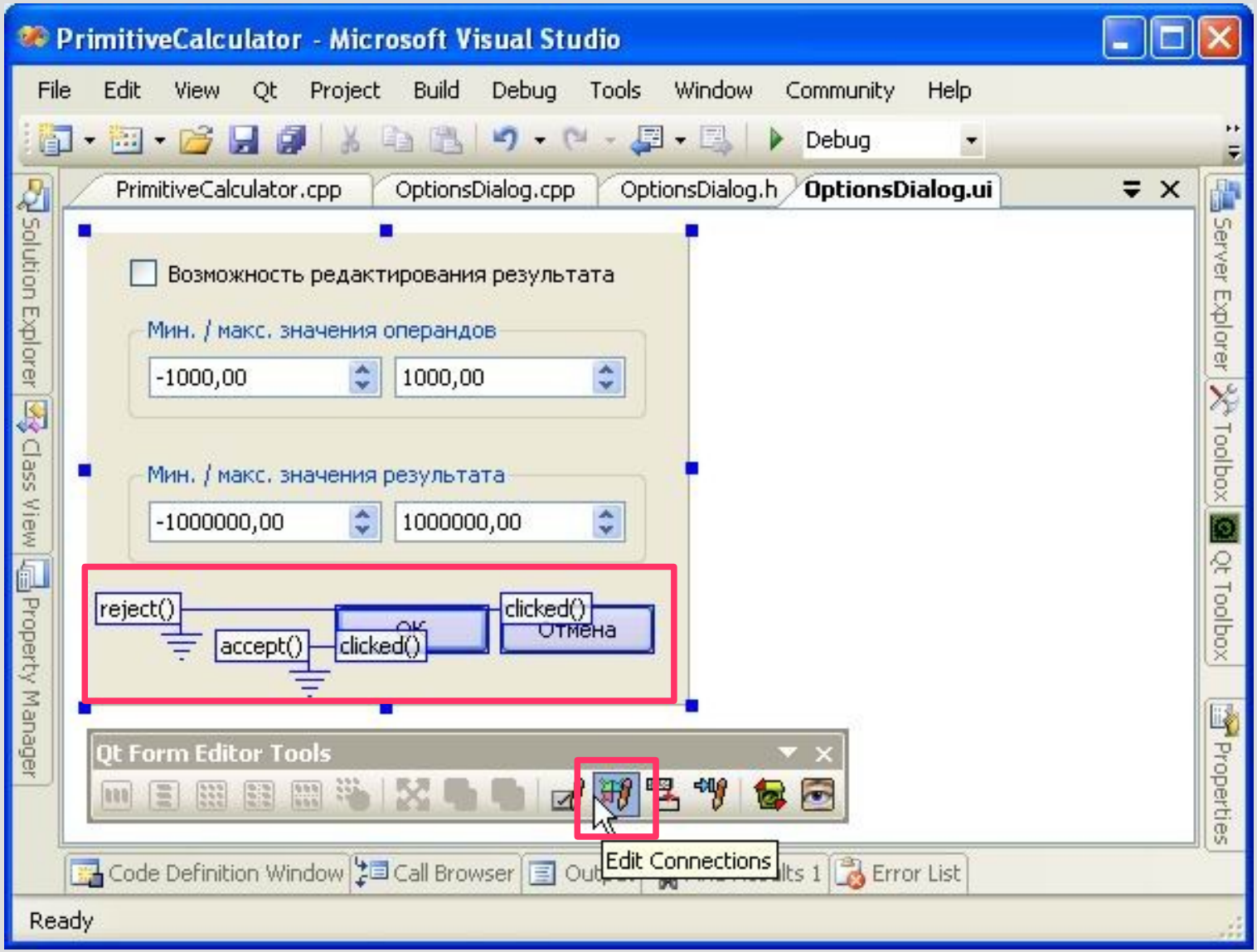
```
void clicked ( )
```

Связывание кнопок со слотами диалога программным способом

```
// Конструктор диалога
OptionsDialog::OptionsDialog(
                    QWidget *parent)
   : QDialog(parent)
{
    ui.setupUi(this);

    // Кнопка "OK" закрывает диалог с
    // подтверждением
    connect(ui.btnOK, SIGNAL(clicked()),
            this, SLOT(accept()));
}
```

Связывание кнопок со слотами диалога визуальным способом



Создание диалога в окне-родителе и «запуск» диалога

- Создание диалога не подразумевает его отображение на экране.
- Для создания диалога необходимо создать экземпляр того класса, который описывает диалог, т.е. объект диалога.
- Для отображения диалога необходимо вызвать один из методов **`show()`** или **`exec()`** применительно к созданному объекту диалога.
- Создание и «вызов» диалога выполняется в окне-родителе.

Создание модального диалога

- Диалог создается в одном из методов окна-родителя как локальный объект. Это объясняется тем, что модальный диалог необходим только внутри вызвавшего его метода.
- Для того чтобы указать, что окно является родителем диалога, в конструктор диалога передается указатель на окно-родитель.
- Синтаксис создания диалога:

```
<имя класса> <объект диалога>(this);
```

«Запуск» модального диалога

- Для отображения диалога вызывается метод

```
int exec ()
```

который возвращает признак того, что пользователь подтвердил (`1`) или опроверг (`0`) свои действия.

Задание

- Создайте и запустите модальный диалог, заданный классом **`OptionsDialog`**.
- Модальный диалог создается и запускается внутри метода **`updateOptions`**`()` класса главного окна **`PrimitiveCalc`**.
- Выставить флаг **`isUpdate`**, если пользователь подтвердил свои действия

Пример создания и запуска модального диалога

```
void PrimitiveCalc::updateOptions()
{
    int dialogCode; // как закрылся диалог
    bool isUpdate;  // флаг

    // Создаем диалог
    OptionsDialog dialog(this);

    // Запускаем диалог
    dialogCode = dialog.exec();

    // Анализируем, как закрылся диалог
    isUpdate = (dialogCode == 1);

} // здесь диалог будет уничтожен
```

Создание немодального диалога

- Диалог создается в одном из методов окна-родителя как динамический объект. Это объясняется тем, что немодальный диалог будет существовать и после выхода из метода.
- Для того чтобы указать, что окно является родителем диалога, в конструктор диалога передается указатель на окно-родитель.
- Синтаксис создания диалога:

```
<имя класса> * <указатель на объект диалога> = new <имя класса>(this);
```

«Запуск» немодального диалога

- Для отображения диалога вызывается метод

```
void show ()
```

Задание

- Создайте и запустите немодальный диалог, заданный классом **`OptionsDialog`**.
- Модальный диалог создается и запускается внутри метода **`updateOptions`**`()` класса главного окна **`PrimitiveCalc`**.

Пример создания и запуска немодального диалога

```
void PrimitiveCalc::updateOptions()
{
    // Создаем диалог
    OptionsDialog *dialog =
                        new OptionsDialog(this);

    // Запускаем диалог
    dialog->show();

} // здесь диалог будет существовать
```

Задание поведения диалогового окна

- Для того чтобы в родительском окне можно было создать диалоговое окно, окно-родитель должно «знать» о диалоговом окне.
- Поэтому в исходном файле окна-родителя должен быть указан заголовочный файл диалогового окна. Например, в файле **PrimitiveCalc.cpp** должна быть указана следующая строчка

```
#include "OptionsDialog.h"
```

Задание поведения диалогового окна

- Диалоговое окно обычно реализует один из следующих вариантов поведения:
  - отображает данные, переданные из окна-родителя;
  - осуществляет ввод данных и передает их окну-родителю;
  - редактирует данные, переданные из окна-родителя, с возвратом результатов окну-родителю.

Задание поведения диалогового окна

- Поведение диалога реализуется посредством реагирования на сигналы, которые порождают виджеты, расположенные на макете диалогового окна.
- Для этого слоты, принадлежащие диалоговому окну, связываются с сигналами виджетов (как в главном окне).

Способы организации взаимодействия между окнами

- Особенностью диалоговых окон является то, что они получают или передают данные окну-родителю.
- При этом возможны различные способы организации многооконного взаимодействия, отличающиеся между собой:
  - по месту хранения данных;
  - по способу передачи данных в диалоговое окно;
  - по расположению кода, выполняющего связанные с диалогом действия.

Место хранения данных

(1) Данные хранятся в окне-родителе, а в диалоговом окне хранится указатель на эти данные (как правило, хранится указатель на главное окно).

(2) Данные хранятся в в диалоговом окне (если эти данные хранятся также и в окне-родителе, то в диалоговом окне хранится их копия).

Способ передачи данных в диалоговое окно

(1) Данные передаются как дополнительные параметры конструктора диалогового окна (первый параметр является обязательным указателем на предка окна)

(2) Данные передаются через специальную функцию инициализации, вызываемую после создания объекта окна но до его показа на экран

По месту расположения действия

- Если окно не просто отображает данные, а вводит параметры некоторого действия (поиска, удаления записи, сохранения в файл и т.д.), то окна различают по расположению кода, выполняющего это действие:

(1) в главном окне

(2) в диалоговом окне

По способу связи между главным окном и немодальным диалогом

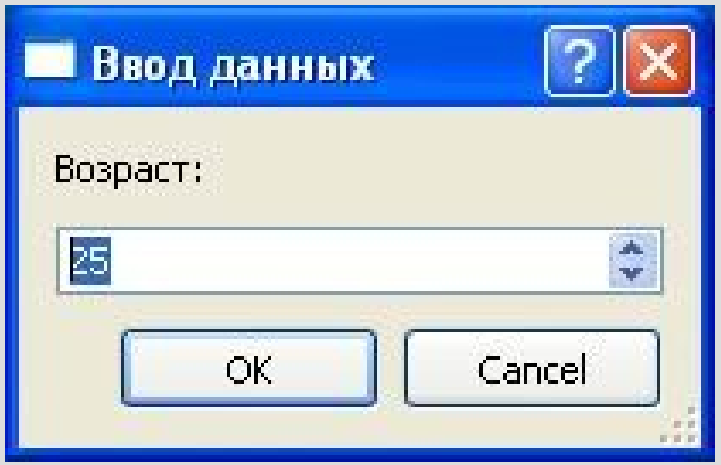
- Если диалог немодальный, то возникает необходимость в передачи информации о действиях пользователя между ним и главным окном в процессе существования окон. Это может быть сделано двумся способами:

(1) с помощью вызова функции

(2) с помощью испускания сигнала

Задание

- Создайте модальное диалоговое окно **`MyInputDialog`** равнозначное стандартному диалогу для ввода целого числа.
- Создание и запуск диалога выполните в методе **`answerAge()`** главного окна программы **`MainWindow`**.



Задание

1) Выберите способ взаимодействия окон

2) Объявите и реализуйте класс **`MyInputDialog`**

3) Реализуйте метод **`answerAge`**`()` главного окна программы **`MainWindow`**, который запрашивает у пользователя его возраст.

Выбор способа взаимодействия окон

- Исходное значения числа будет храниться в главном окне, а в диалоговое окно будет передаваться его копия, которая и будет модифицироваться пользователем.
- Исходное значения числа будет передаваться как параметр конструктора диалогового окна.
- Модифицированное пользователем число будет возвращаться из диалога с помощью метода

```
int getNumber()
```

Объявление класса MyInputDialog

```
// Класс диалогового окна
class MyInputDialog : public QDialog
{
...
public:
   // Конструктор
   MyInputDialog(QWidget *parent = 0,
        const QString & title,
        const QString & label, int number);

   // Метод чтения текущего значения числа
    int getNumber();
...
};
```

Реализация класса MyInputDialog

```
// Конструктор
MyInputDialog::MyInputDialog(
        QWidget *parent = 0,
        const QString & title,
        const QString & label, int number)
    ...
{
    ...
   // Фиксируем исходное значение числа
    ui.NumberBox->setValue(number);
    // Отображаем заголовок и метку
    setTitle(title);
    ui.Label->setText(label);
}
```

Реализация класса MyInputDialog

```
// Метод чтения текущего значения числа
int MyInputDialog::getNumber()
{
   return ui.NumberBox->value();
}
```

Использование класса MyInputDialog

```
// Метод главного окна
void MainWindow::answerAge()
{
    int age = -1; // возраст пользователя

    // Создаем диалоговое окно
    MyInputDialog dialog(this, "Ввод данных",
      "Возраст:", 18);

    // Отображаем диалоговое окно и
    // фиксируем возраст пользователя
    if(dialog.exec())age =dialog.getNumber();
    ....
}
```