Основы графического интерфейса пользователя

- Понятие графического интерфейса пользователя, окна и виджета
- Категории виджетов
- Создание макета окна в редакторе форм (QT-Designer). Задание свойств виджетов
- Проблема различных разрешений экрана. Управление размещением виджетов

Основы графического интерфейса пользователя

- Объекты, соответствующие виджетам, доступ к ним из программы
- Понятие программы, управляемой событиями
- Понятие сигналов и слотов
- Использование сигналов и слотов для реализации программы, управляемой событиями

Раздел I. Понятие графического интерфейса пользователя

- Графический интерфейс пользователя (GUI) — это интерфейс, основанный на представлении всех доступных пользователю функций в виде графических компонентов экрана (окон, значков, меню, кнопок, списков и т.п.).

- GUI-программа обычно представляет собой множество окон, каждое из которых содержит множество элементов управления.

Основные понятия

- Окно представляет собой «отдельный» экран со своим набором элементов управления или виджетов.

- Виджет (Widget = Window - окно + Gadget - приспособление) - это элемент управления, способный реагировать на действия пользователя.

Категории виджетов

- Виджеты отображения
- Виджеты ввода данных
- Виджеты выбора
- Кнопки
- Виджеты группировки

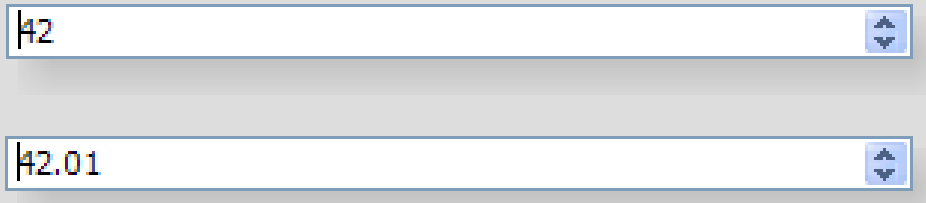
Далее рассматриваются виджеты из библиотеки QT Library

Виджеты отображения

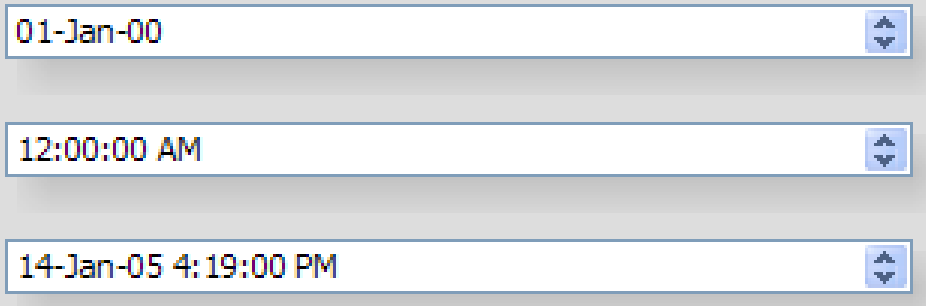
- Виджеты отображения не принимают активного участия в действиях пользователя, они используются только для информирования его о происходящем.

Виджеты ввода данных

- Счетчики (**SpinBox** и **DoubleSpinBox**) – используются для ввода чисел из ограниченного диапазона упорядоченных чисел.



- Элементы ввода даты и времени (**DateEdit**, **TimeEdit** и **DateTimeEdit**).

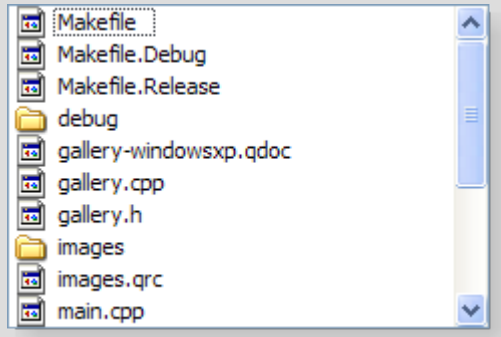


Виджеты выбора

- Виджеты выбора используются для выбора пользователем одного или нескольких значений из множества значений.

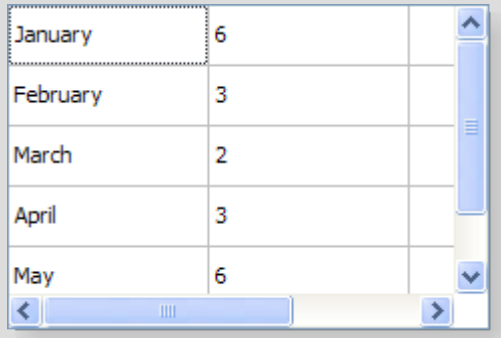
Виджеты выбора

- Простой список (**ListWidget**) – поддерживает одноэлементный и множественный выбор.

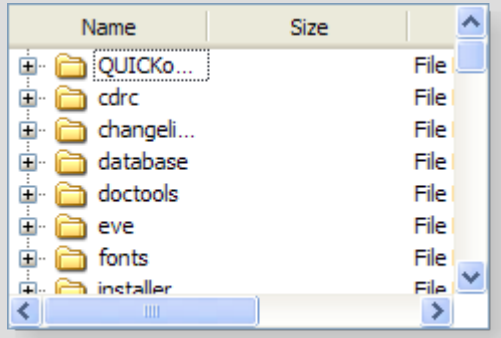


- Выпадающий список (**ComboBox**).

- Таблица (**TableWidget**) – в отличие от простого списка имеет несколько информационных колонок.



- Дерево (**TreeWidget**) - позволяет отображать список в иерархической форме (например, содержимое дисков и каталогов).

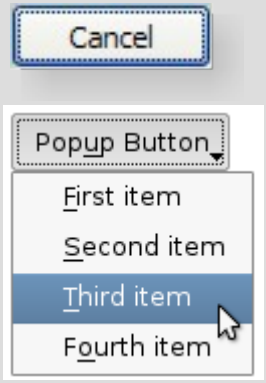


Применение виджетов для отображения и редактирования данных

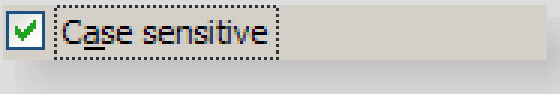
Тип данных	Виджет
целое число	SpinBox
вещественное число	DoubleSpinBox
строка	LineEdit
текст	TextEdit
дата	DateEdit
время	TimeEdit
дата и время	DateTimeEdit
флаг	CheckBox
элемент множества	ComboBox, ListWidget, группа RadioButton
несколько элементов множества	ListWidget, группа CheckBox

Кнопки

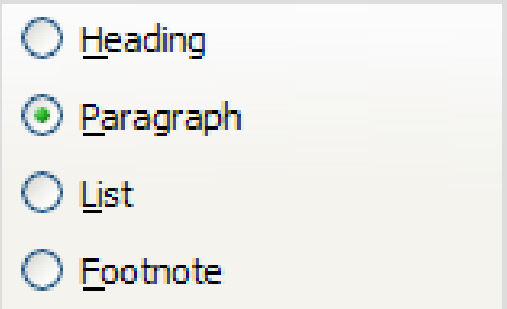
- Командная кнопка (**PushButton** - кнопка нажатия) – активизирует выполнение некоторого действия. Различают обычную кнопку, кнопку-выключатель и кнопку-меню.



- Флажок (**CheckBox**)– обычно используется для задания настроек программы. Может иметь третье, неопределенное, состояние.



- Переключатель (**RadioButton**)– обеспечивает выбор только одной опции из нескольких.

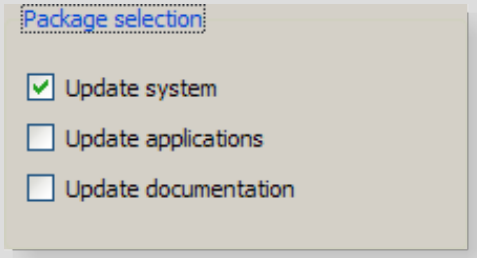


Виджеты группировки

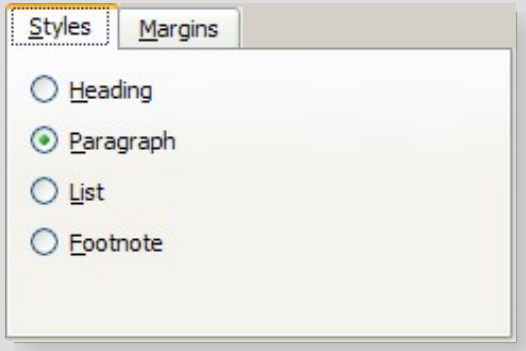
- Данная категория виджетов используется для объединения виджетов в смысловые группы.
- Использование виджетов группировки иногда изменяет поведение виджетов, входящих в группу (например, переключателей).
- Над виджетами, входящих в группу, можно выполнять операции как с одним целым (например, сдвигать, скрывать и т.д.).

Виджеты группировки

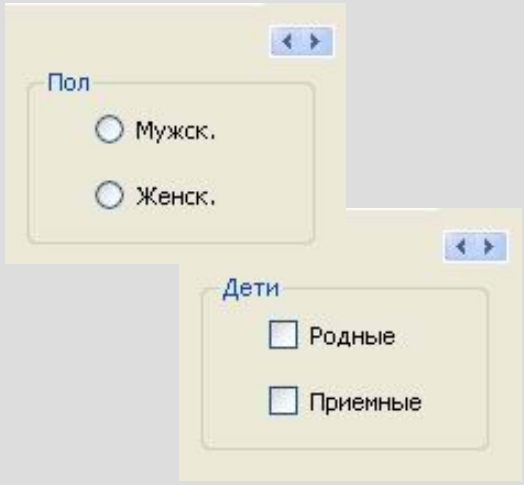
- Виджет «группа» (**GroupBox**) объединяет несколько виджетов под одним заголовком.



- Виджет «закладки» (**TabWidget**) включает в себя несколько групп виджетов. Группе соответствует одна закладка.



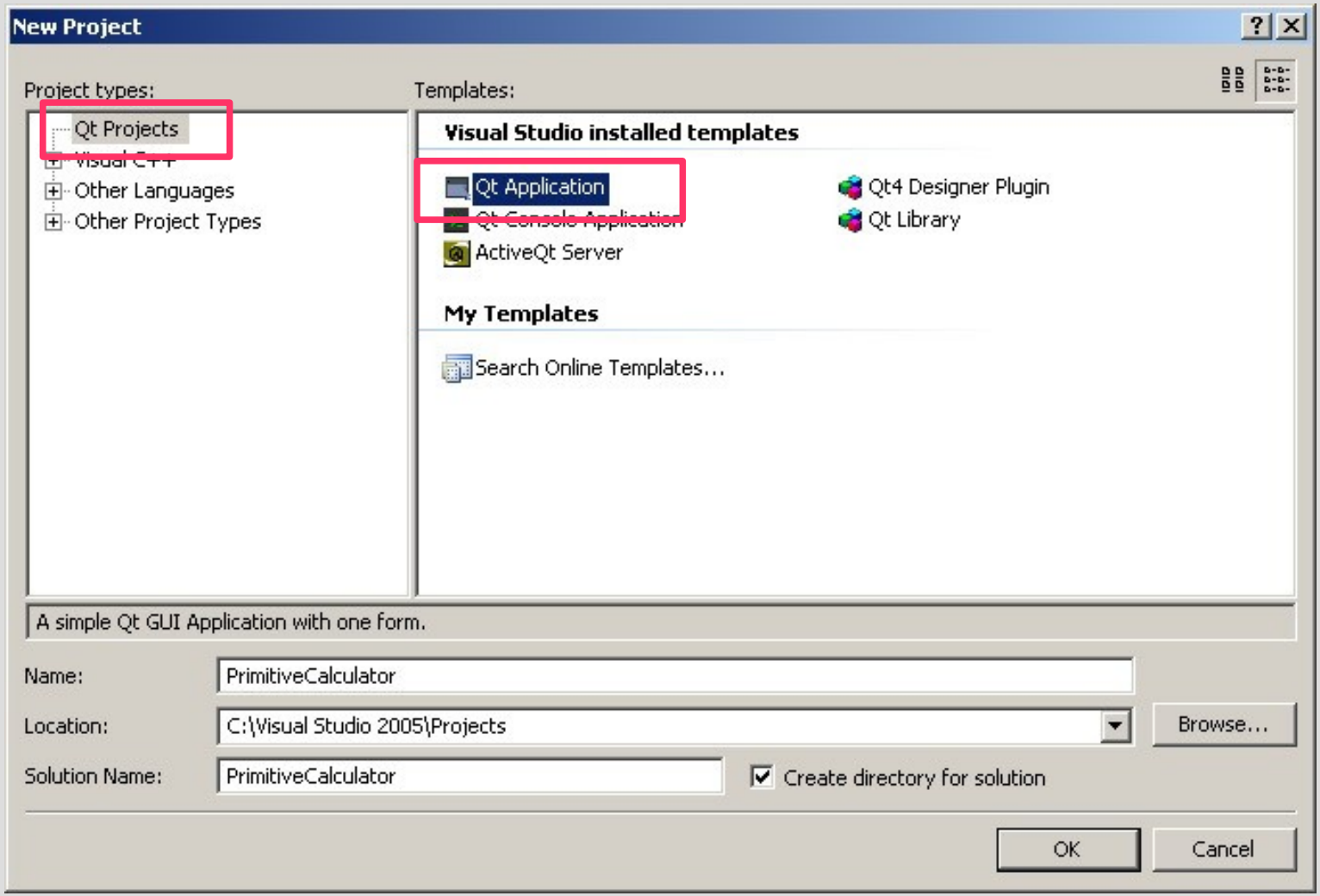
- Виджет «стек» (**StackedWidget**) включает в себя несколько групп виджетов, однако, на экране отображается только одна группа виджетов.



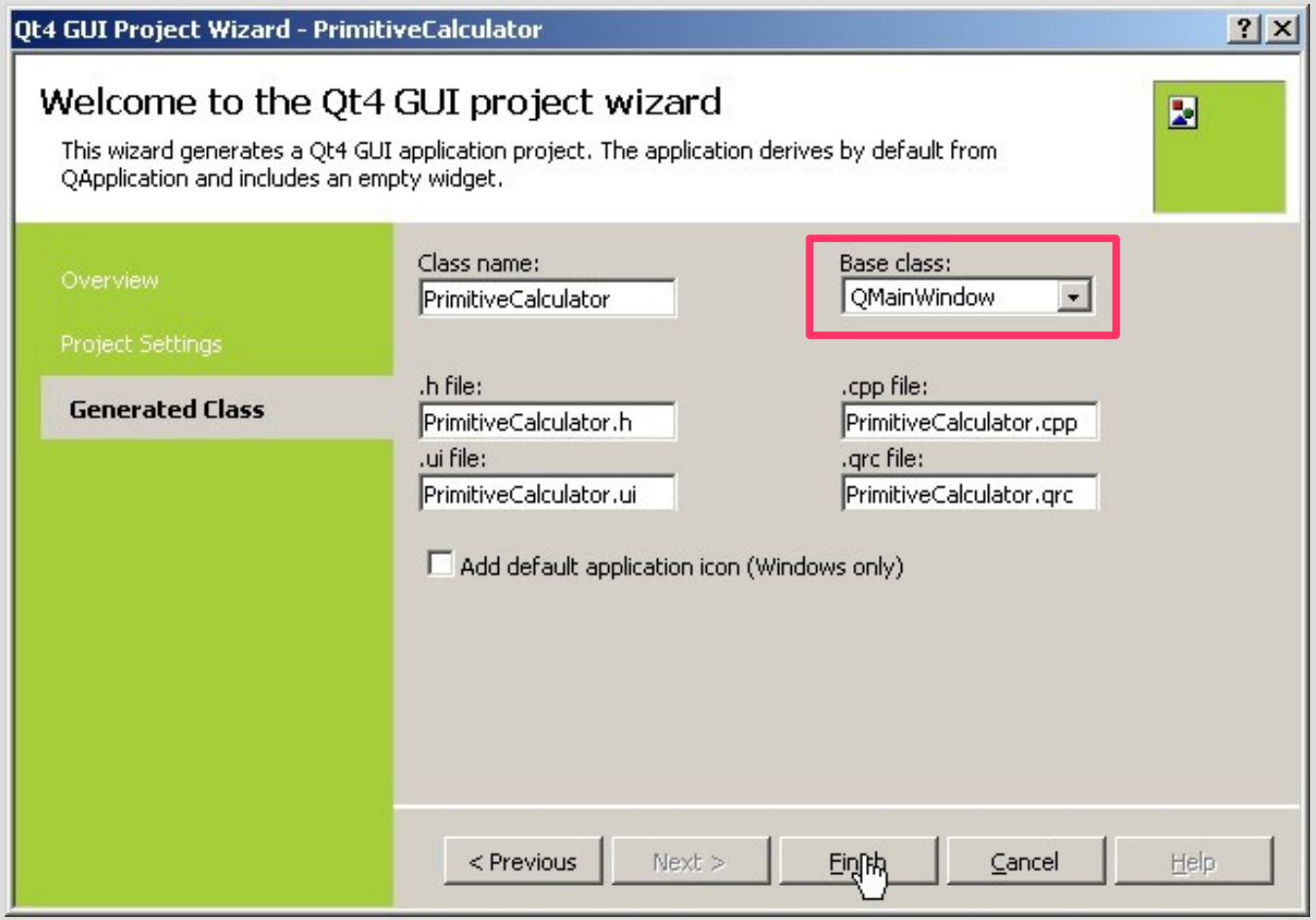
Раздел II. Процесс создания макета окна в редакторе форм

1) Создайте проект **`QT Application`**
2) Задайте в качестве главного окна программы класс **`QMainWindow`**
3) Расставьте виджеты на макете окна, перетаскивая их из панели инструментов на форму
4) Задайте основные свойств виджетов
5) Настройте последовательность обхода виджетов

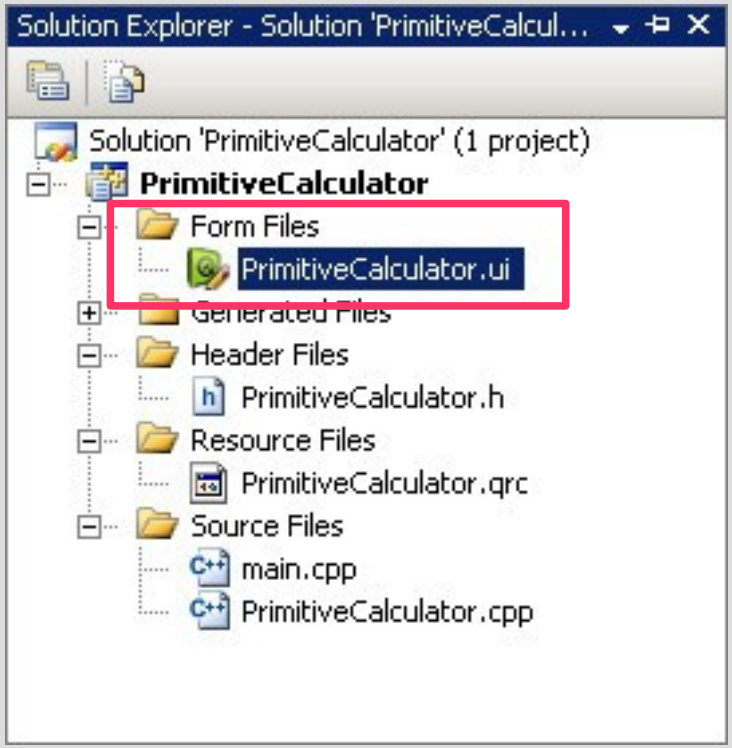
Создание проекта QT Application



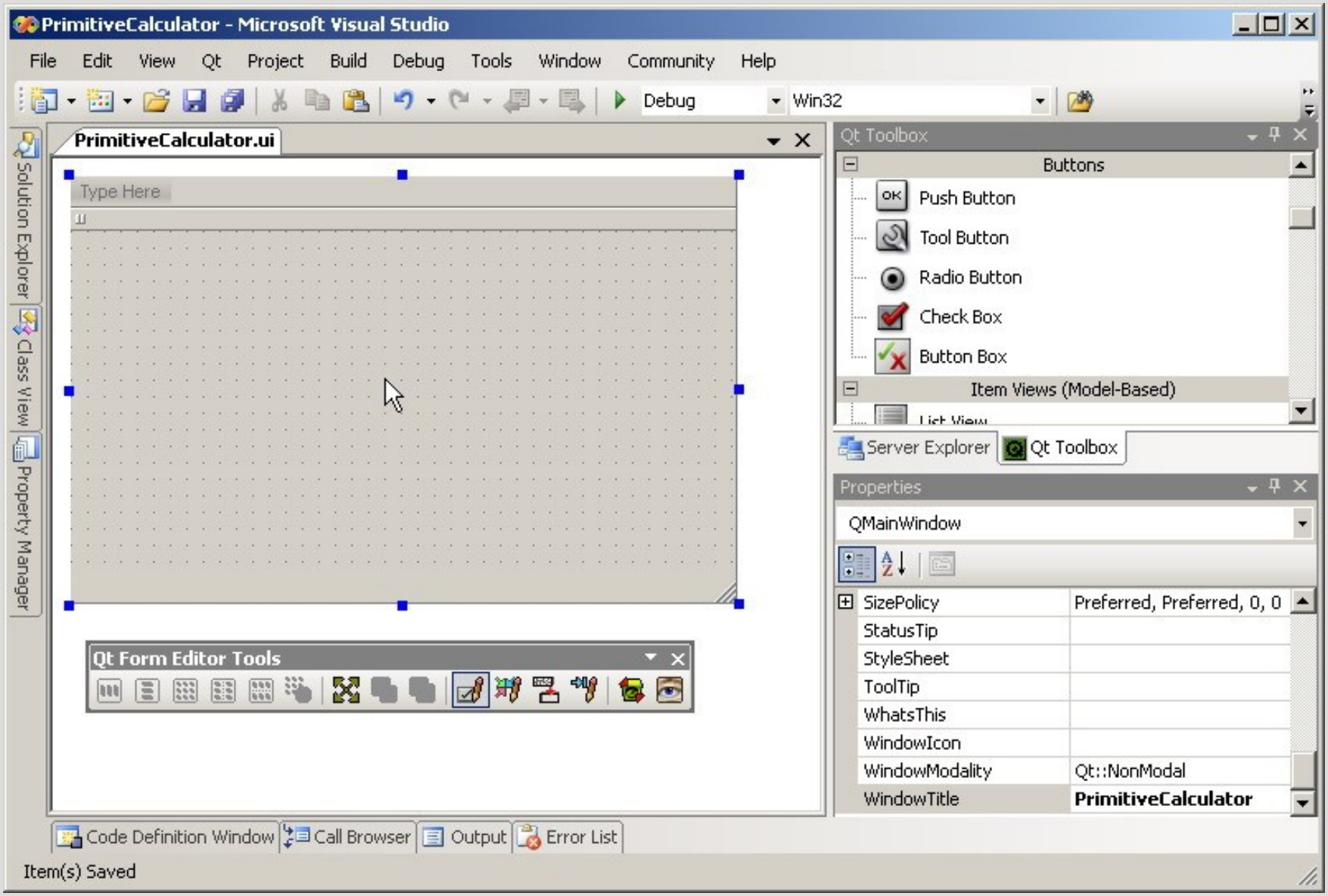
Задание в качестве главного окна программы QMainWindow



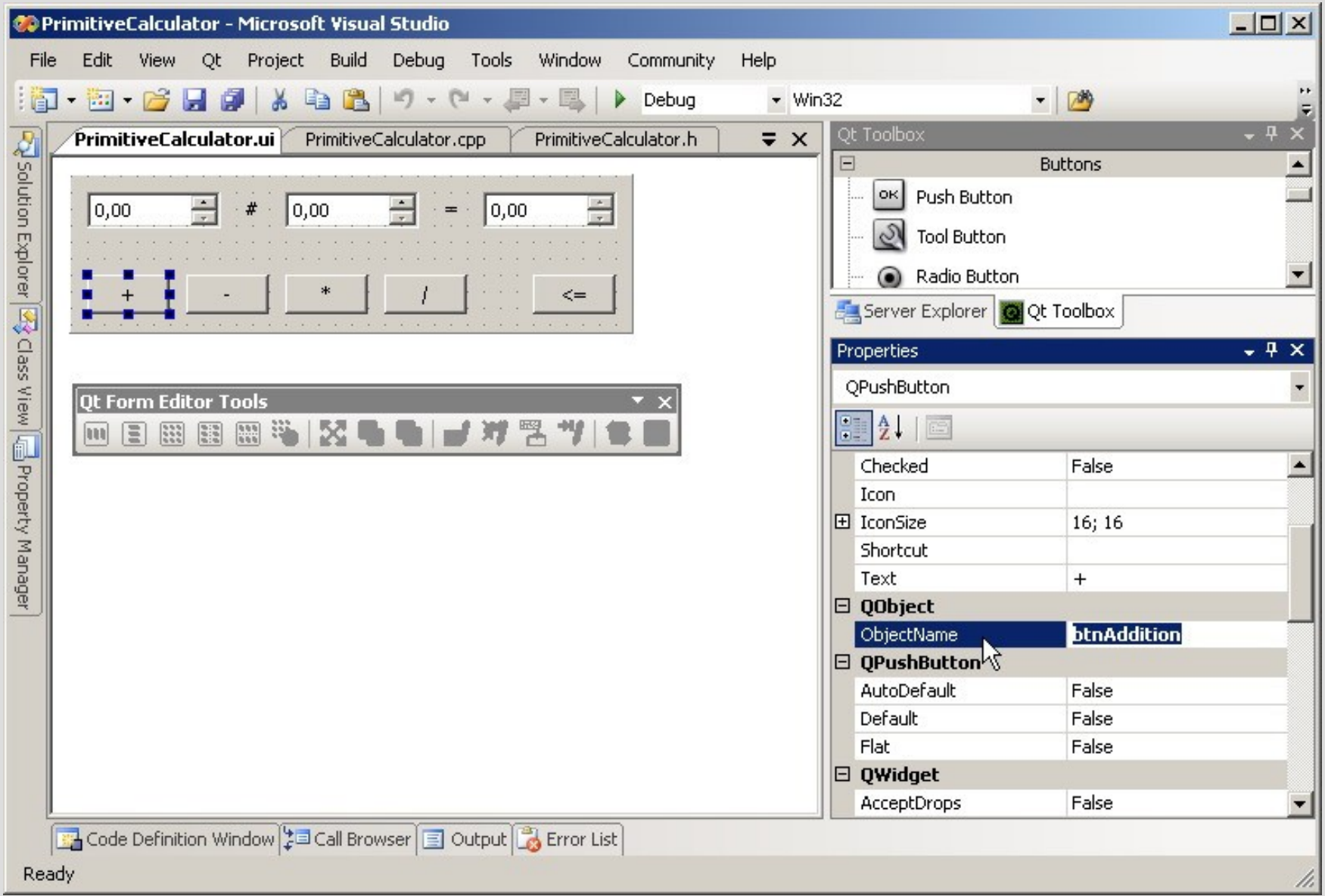
Выбор макета окна в проекте



Расстановка виджетов на макете окна



Расстановка виджетов на макете окна, задание их основных свойств



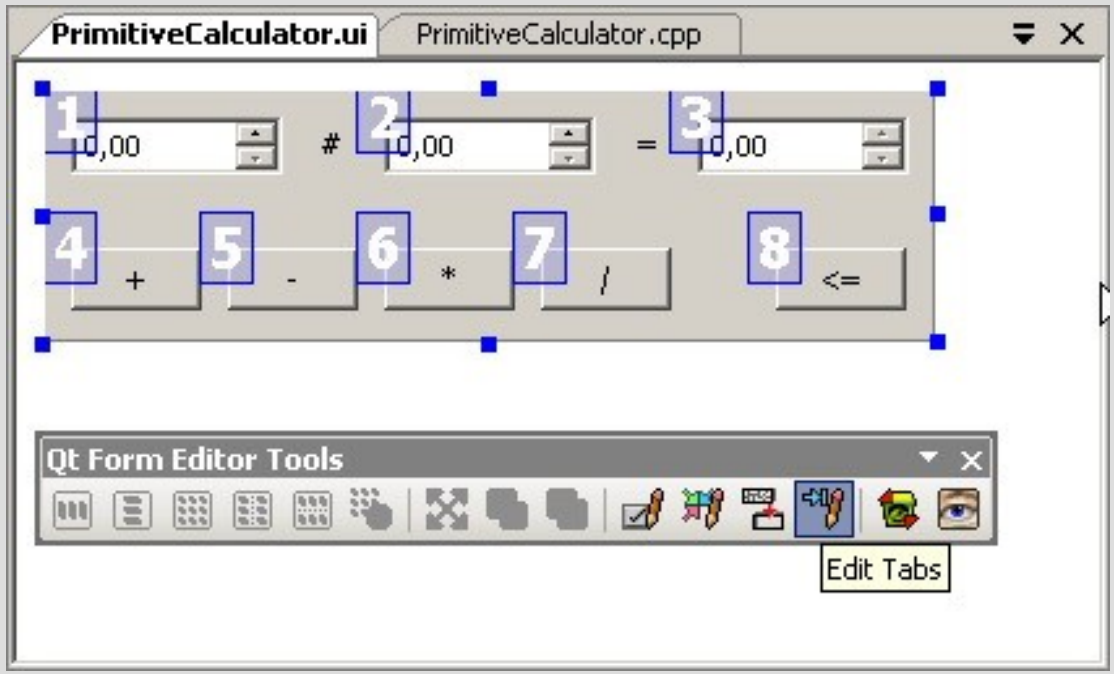
Настройка последовательности обхода виджетов. Понятие фокуса

- Пользователь в каждый момент времени может работать только с одним виджетом - виджетом, который имеет фокус.
- Если виджет в фокусе, то все коды нажатых клавиш направляются этому виджету, в результате чего изменяется его состояние.
- При отображении окна один из виджетов автоматически получает фокус.

Настройка последовательности обхода виджетов. Понятие фокуса

- Перенаправить фокус с одного виджета на другой можно с помощью мыши или клавиш **`Tab`** (прямой обход) и **`Shift+Tab`** (обратный обход).
- Последовательность, в которой фокус переходит от одного виджета к другому (с помощью клавиш), называется последовательностью обхода.
- Последовательность обхода задается при создании макета окна.

Настройка последовательности обхода виджетов в редакторе форм



Проблема различных разрешений экрана

- Разрешение экрана, в котором создается макет окна, может отличаться от рабочего разрешение экрана
- Если разрешение меньше, то видны не все виджеты
- Если разрешение больше, то большая часть экрана не используется

Решение проблемы различных разрешений экрана

- Необходимо растягивать/сжимать виджеты в соответствии с текущим разрешением экрана.
- Различные виджеты должны сжиматься и растягиваться по-разному с учетом их текущего содержимого.
- Необходимо растягивать/сжимать пространство между виджетами в соответствии с текущим разрешением экрана.

Решение проблемы различных разрешений экрана

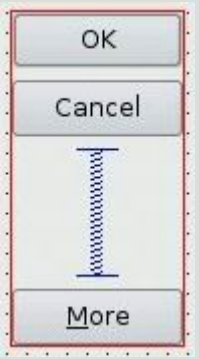
- В QT Library проблема различных разрешений экрана решается с помощью менеджеров компоновки

Менеджеры компоновки

- Обеспечивают разумные размеры по умолчанию для каждого типа виджетов, учитывая "идеальные" размеры каждого из них, которые, в свою очередь, зависят от выбранного размера шрифта, стиля отображения и объема содержимого.
- Учитывают минимальные и максимальные размеры, и автоматически корректируют расположение виджетов, в ответ на изменение шрифта, содержимого или размеров окна.

Менеджеры компоновки

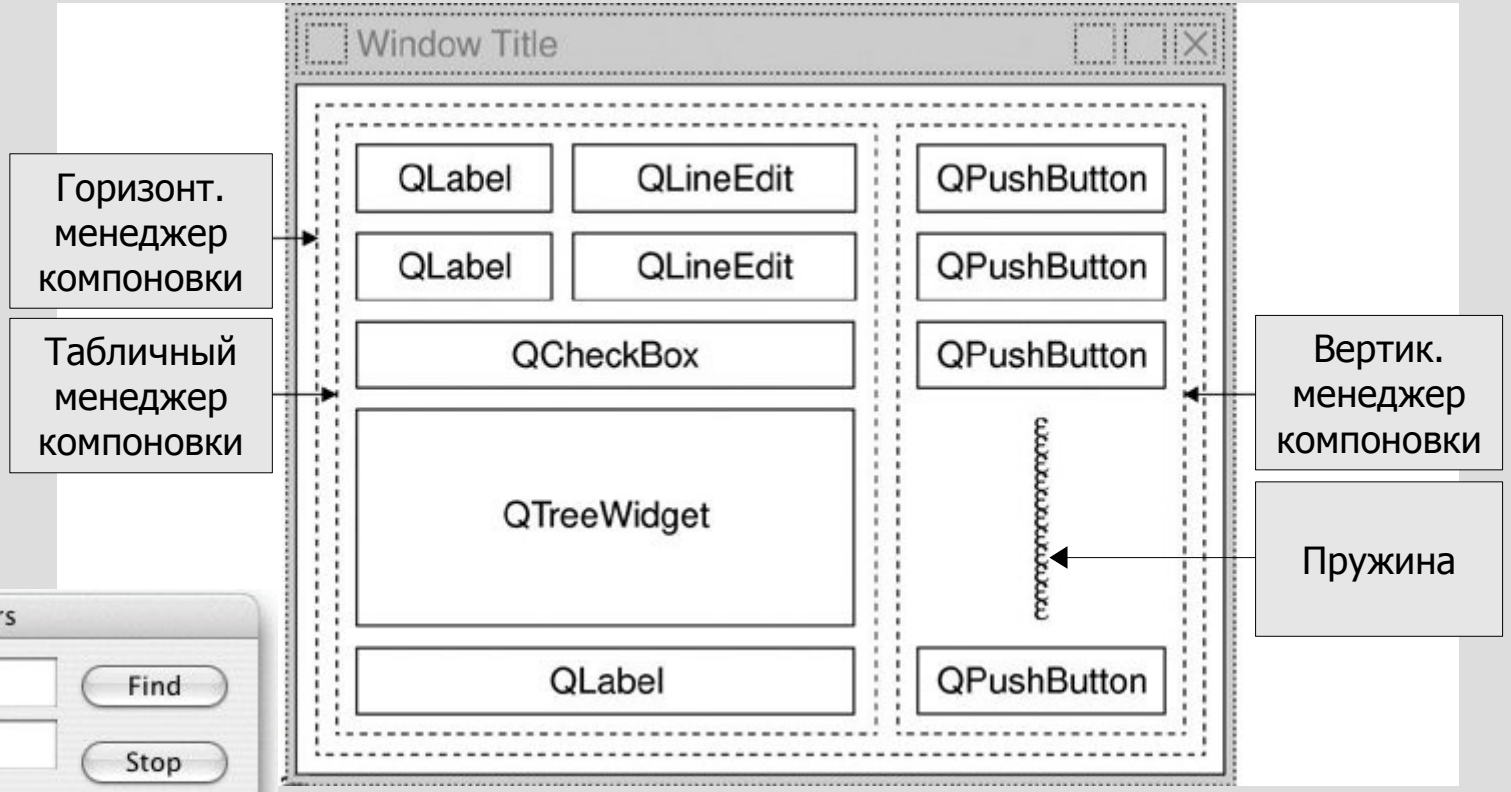
- Горизонтальный менеджер компоновки (**`HBoxLayout`**)
- Вертикальны менеджер компоновки (**`VBoxLayout`**)
- Табличный менеджер компоновки (**`GridLayout`**)
- «Пружина» (**`SpacerItem`**)



Пример использования менеджеров компоновки

Пример «растягивающего» окна: «растягиваются» поля ввода, таблица и пространство между кнопками «Close» и «Help»

Пример использования менеджеров компоновки



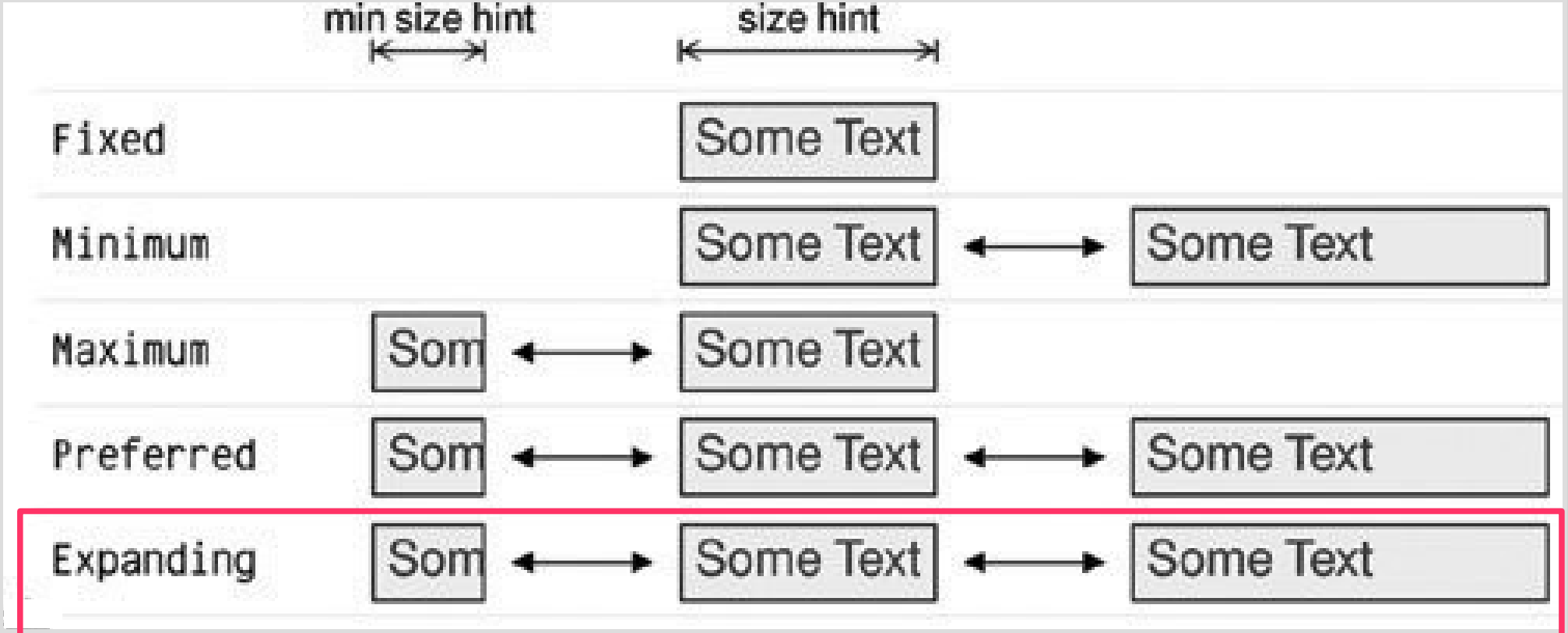
Политика изменения размеров виджетов

- Как будет растягиваться виджет определяется его свойством **`SizePolicy`**. Оно может принимать следующие значения.
- **`Fixed`** - виджет имеет фиксированные размеры (равные "идеальным"), т.е. он не может растягиваться или сжиматься.
- **`Minimum`** - "идеальный" размер виджета, это минимально возможный его размер. Виджет не может сжиматься меньше этого размера, но может растягиваться и занимать все доступное пространство, если это потребуется.

Политика изменения размеров виджетов

- **Maximum** - "идеальный" размер виджета, это максимально возможный его размер, т.е. виджет может сжиматься до минимально возможного размера, но не может растягиваться больше "идеального".
- **Preferred** - "идеальный" размер виджета, это предпочтительный его размер, но в случае необходимости виджет может как растягиваться, так и сжиматься.
- **Expanding** - виджет может и растягиваться, и сжиматься, но он предпочитает растягиваться.

Различные политики изменения размеров для метки



Значение свойства **`SizePolicy`** по умолчанию

Раздел III. Программная модель GUI. Класс главного окна

- При создании макета окна в программе автоматически создается класс главного окна, производный от класса **`QMainWindow`**.
- Класс главного окна содержит в себе:
  - объекты, соответствующие виджетам, которые расположены на макете окна;
  - данные, время жизни которых совпадает со временем жизни программы;
  - функции, которые выполняет программа в ответ на действия пользователя.

Пример класса главного окна

```
class PrimitiveCalc : public QMainWindow
{    Q_OBJECT
public:
    // Конструктор и деструктор
    PrimitiveCalc(QWidget *parent=0,Qt::WFlags flags=0);
    ~PrimitiveCalc();
private:
    // Виджеты, расположенные на макете окна
    Ui::PrimitiveCalcClass ui;

    // Данные
    ......
private slots:
    // Слоты, т.е. функции, реагирующие на
    // действия пользователя
    ......
};
```

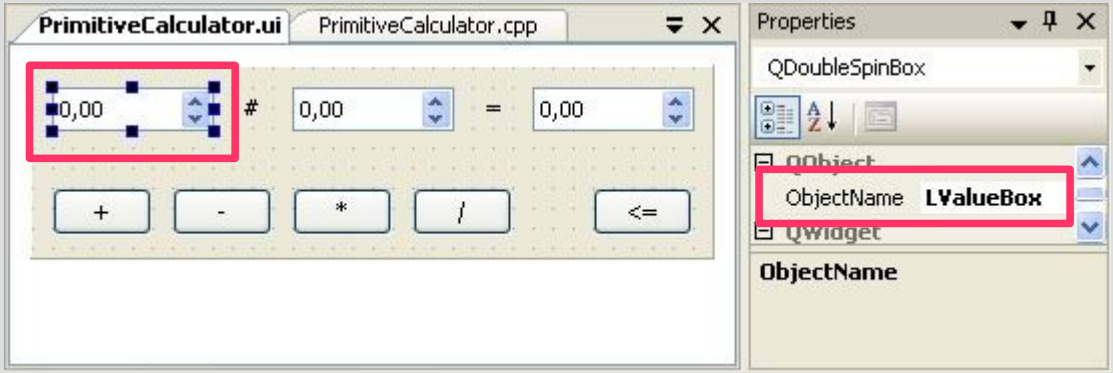
Объекты, соответствующие виджетам

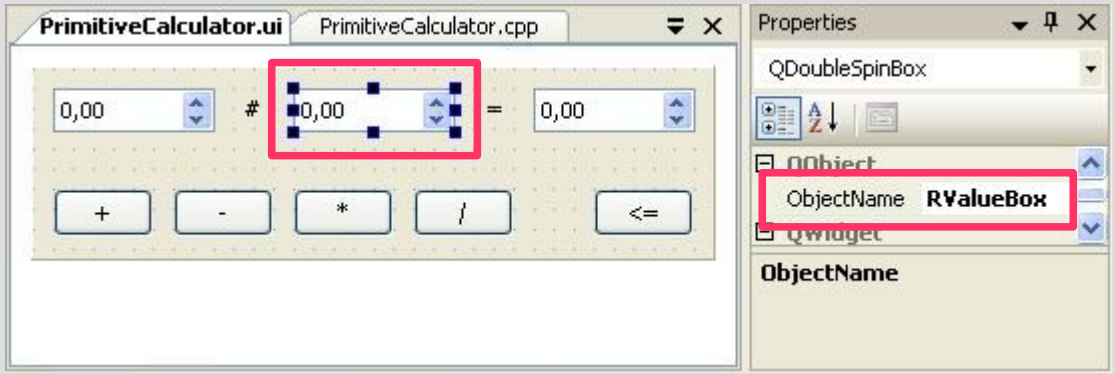
- Каждому виджету, расположенному на макете окна, внутри класса главного окна автоматически создается объект.
- Доступ к такому объекту внутри класса главного окна осуществляется с помощью следующей конструкции:

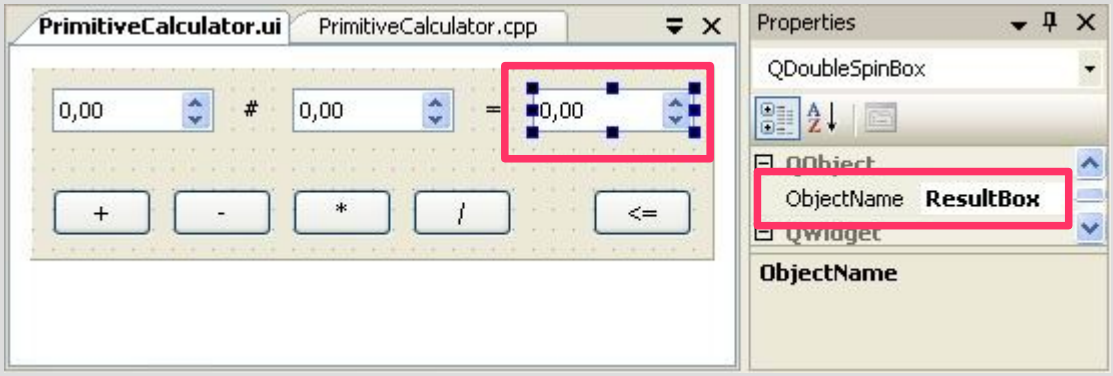
```
ui.<имя объекта>
```

где **<имя объекта>** задается при построении макета окна

Задание виджету имени объекта







Доступ к виджетам из программы

- Конструкция **`ui.<имя объекта>`** возвращает указатель на объект, т.е. экземпляр класса, соответствующего некоторому виджету
- Пример:
  - пусть для одного из счетчиков, расположенных на макете окна, определено имя **`LValueBox`**
  - Любой счетчик описывается классом **`QDoubleSpinBox`**
  - Тогда конструкция **`ui.LValueBox`** будет иметь тип **`QDoubleSpinBox *`**

Пример доступа к виджетам из программы

```
// Сложение двух чисел
void PrimitiveCalc::on_addition()
{
   double LValue, RValue;    // операнды

   // Считываем операнды из двух счетчиков
   LValue = ui.LValueBox->value();
   RValue = ui.RValueBox->value();

   // Вычисляем результат и отображаем его
   // в третьем счетчике
   ui.ResultBox->setValue(LValue + RValue);
}
```

Классы, соответствующие виджетам

- Каждому виджету соответствует определенный класс со своим набором свойств и методов.
- В QT Library имя класса виджета задается по следующему принципу: к английскому названию виджета добавляется большая буква **Q**, например, **QDoubleSpinBox**.
- Для использования класса необходимо подключить заголовочный файл, соответствующий имени класса, например,

```
#include <QDoubleSpinBox>
```

Виджеты и соответствующие им классы

Виджеты	Классы
Счетчики	**QSpinBox, QDoubleSpinBox**
Однострочное поле ввода	**QLineEdit**
Редактор многострочного текста	**QTextEdit**
Виджеты ввода даты и времени	**QDateEdit, QTimeEdit, QDateTimeEdit**
Командная кнопка	**QPushButton**
Флажок	**CheckBox**
Радио-кнопка	**QRadioButton**
Список	**QListWidget**
Таблица	**QTableWidget**

Пример описания класса QDoubleSpinBox

Properties – основные свойства

```
maximum : double                    singleStep : double
minimum : double                    value : double
```

Public Functions – методы чтения/записи свойств

```
QSpinBox ( QWidget * parent = 0 )
QString cleanText () const
void setRange ( double minimum, double maximum )
void setSingleStep ( double val )
double singleStep () const
double value () const
```

Public Slots – особый метод- слот

```
void setValue ( double val )
```

Задание

Используя метод

```
void setRange (double minimum,
               double maximum)
```

задайте трем счетчикам **LValueBox**, **RValueBox** и **ResultBox** в программе «Калькулятор» диапазон возможных значений, равный [-100.0, 100.0].

Действие выполните в конструкторе класса окна **PrimitiveCalc**.

Пример доступа к виджетам из программы

```
// Конструктор класса окна

void PrimitiveCalc::PrimitiveCalc()
{
    ....

    // Задаем диапазоны счетчиков
    ui.LValueBox->setRange(-100.0, 100.0);
    ui.RValueBox->setRange(-100.0, 100.0);
    ui.ResultBox->setRange(-100.0, 100.0);
}
```

GUI-программа — это реактивная программа

- Последовательность действий в активной программе полностью определяется ее кодом (набором процедур).
- Последовательность действий в реактивной программ определяется пользователем, т.к. программа выполняет процедуры, реагируя на воздействия пользователя.
- Реактивная программа представляет собой набор независимых процедур, взаимодействующих между собой посредством общих данных.

Понятие сигнала

- Когда пользователь воздействует на виджет, то виджет изменяет свое состояние и испускает сигнал, извещая программу об этом событии.
- Каждый виджет обладает своим предопределенным набором сигналов. Например, нажатие на командную кнопку приводит к испусканию сигнала `clicked`.

Понятие слота

- Если программе требуется отреагировать на некоторое событие, то в ней должен быть создан обработчик события, также называемый слотом.
- Слот — это обычная функция, которая вызывается в ответ на испускание сигнала.
- Для того чтобы слот был вызван в ответ на сигнал, он должен быть соединен с сигналом.

Свойства сигналов и слотов

- Сообщения, посылаемые посредством сигналов, могут иметь множество аргументов любого типа.
- Сигнал можно соединять с различным количеством слотов. Высылаемый сигнал, в этом случае, поступит ко всем подсоединенным слотам (в произвольном порядке).
- Слот может принимать сообщения от многих сигналов, принадлежащих разным объектам.

Свойства сигналов и слотов

- Соединение (отсоединение) слотов и сигналов можно производить в любой точке приложения.
- Соединение сигналов и слотов обычно выполняется в конструкторе класса окна.
- Возможно отсоединение слотов и сигналов, но в простых программах этот механизм не используется.

Объявление слотов

- Слот обычно объявляется как закрытый метод класса окна в специальном разделе `slots`.
- Слот не имеет возвращаемого значения.
- Слот может иметь аргументы, если они определены в сигнале.

Объявление слотов

- Слот не обязан использовать все аргументы сигнала. Однако, если он их использует, то состав, тип и порядок аргументов слота и сигнала должны совпадать.
- Разрешается использовать в слоте только первые n аргументов сигнала.

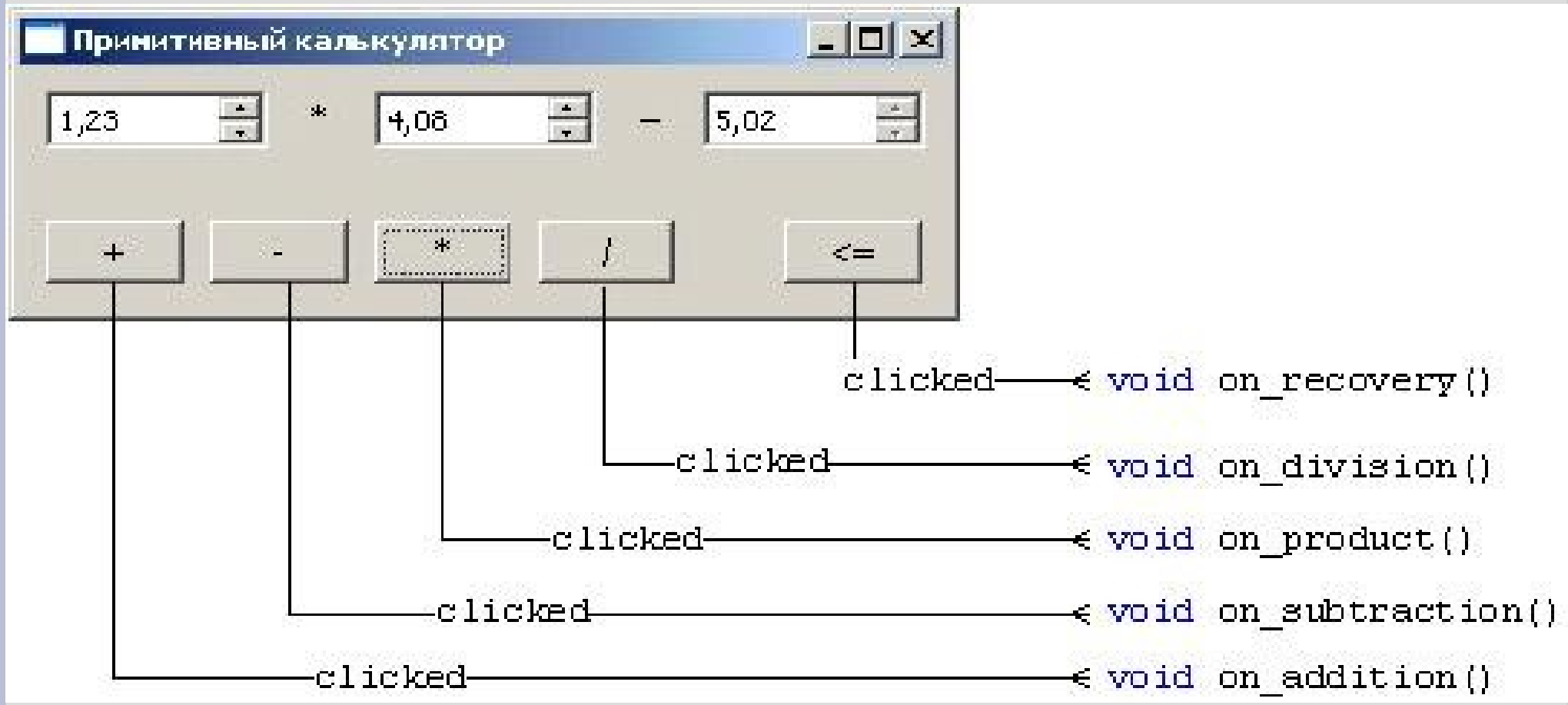
Пример объявление слотов

- В программе «Калькулятор» все действия выполняются по нажатию пяти кнопок, соответственно должно быть создано пять различных обработчиков событий (слотов).
- У кнопки имеется сигнал **clicked**, который испускается, когда кнопка нажимается, а затем отпускается:

```
void clicked (bool checked = false)
```

- Так как используется непереключательная кнопка, то аргумент сигнала не используется.

Схема соединения слотов и сигналов



Пример объявление слотов

```
// Конструктор класса окна

class PrimitiveCalculator : public QMainWindow
{
  ...
private slots:                  //СЛОТЫ:
    void on_addition();         // Сложение двух чисел
    void on_subtraction();      // Вычитание двух чисел
    void on_product();          // Произвед. двух чисел
    void on_division();         // Деление двух чисел
    void on_recovery();         // Помещение результата
                                // в операнд
 };
```

Соединение слотов и сигналов

- Для соединения сигналов и слотов используется метод **connect**:

```
connect(<указатель на источник сигнала>, SIGNAL(<прототип сигнала>),
        <указатель на получатель сигнала>,
        SLOT(<прототип слота>));
```

Соединение слотов и сигналов

- **<указатель на источник сигнала>** - это указатель на объект виджета, испускающего сигал
- **<указатель на получатель сигнала>** - обычно это указатель на окно, к которому принадлежит виджет. В этом случае указывается специальное ключевое слово **this**.

Таблица соединения слотов и сигналов

Источник сигнала	Сигнал	Приемник сигнала	Слот
btnAddition	clicked()	this	on_addition()
btnSubtraction	clicked()	this	on_subtraction()
btnProduct	clicked()	this	on_product()
btnDivision	clicked()	this	on_division()
btnRecovery	clicked()	this	on_recovery()

Пример соединения слотов и сигналов

```
// Конструктор класса окна
PrimitiveCalc::PrimitiveCalc(QWidget *parent,
                             Qt::WFlags flags)
 : QMainWindow(parent, flags)
{
    // Команда необходима для автоматичсекого
    // создания объектов, соотвествующих виджетам
    ui.setupUi(this);

    // Связываем слоты с сигналами
    connect(ui.btnAddition,SIGNAL(clicked()),this,
            SLOT(on_addition()));
    connect(ui.btnSubtraction,SIGNAL(clicked()),
            this, SLOT(on_subtraction()));
.....
}
```

Задание

У виджета «таблица» (**QTableWidget**) имеется сигнал, который испускается, когда фокус переходит от одной ячейки таблицы к другой

```
void currentCellChanged (
int currentRow, int currentColumn,
int previousRow, int previousColumn )
```

где **previousRow** и **previousColumn** — индексы той ячейки, которая имела фокус

currentRow, **currentColumn** - индексы той ячейки, которая получила фокус

Задание

Создайте в классе окна **MyWindow** слот **on_selection_changed()**, который бы выставлял флажок “**Справа**”, если фокус перешел к ячейке справа и сбрасывал его в потивном случае.

Таблица имеет название **table**.

Флажок “**Справа**” имеет название **checkRight**.

Для выставления/сброса флажка воспользуйтесь методом

void setChecked (bool)

Пример объявление слота

```
// Объявление класса
class MyWindow : public QMainWindow
{
 ...
private slots:                    //СЛОТ
    void on_selection_changed(int currentRow, int
      currentColumn, int previousRow, int
      previousColumn);
 };
```

Пример соединения слота

```
// Конструктор класса окна
MyWindow::MyWindow()
{
 ....
    connect(ui.table,
      SIGNAL(currentCellChanged(int, int, int, int)),
      this, SLOT(on_selection_changed(int, int, int,
      int)));
 }
```

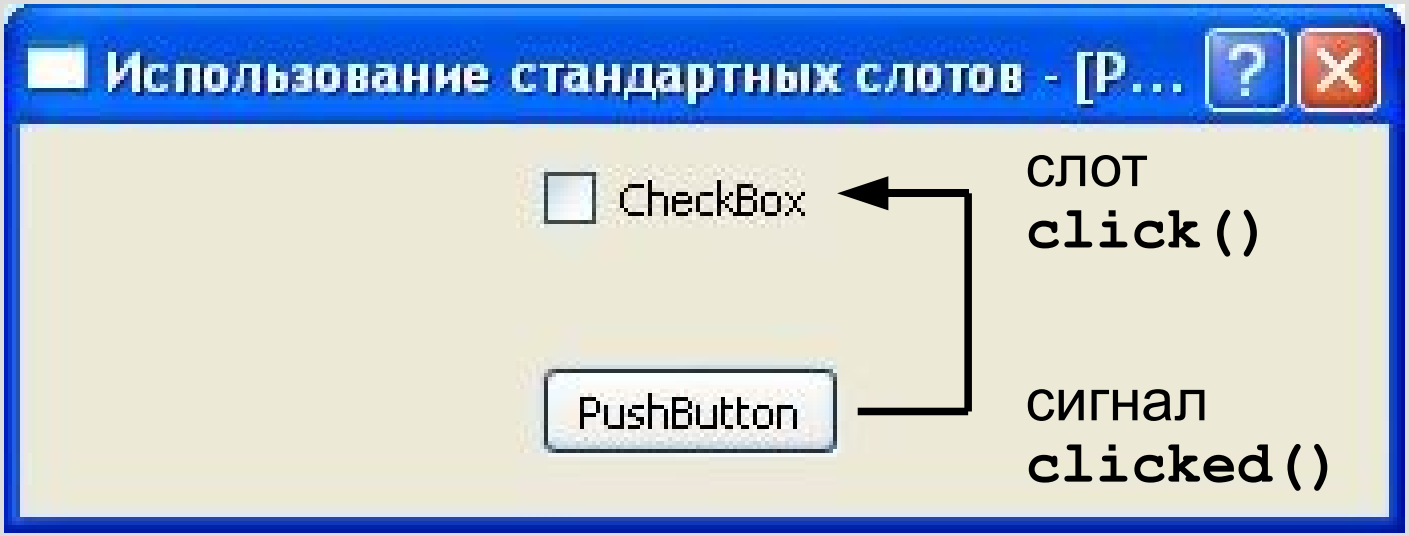
Пример реализации слота

```
// Слот
void MyWindow::on_selection_changed(int currentRow,
                int currentColumn, int previousRow,
                int previousColumn)
{
    bool isRight;     // выбрана ячейка справа

    isRight = (currentRow == previousRow) &&
              (currentColumn – previousColumn == 1);

    ui.checkRight->setChecked(isRight);
 }
```

Стандартные слоты

- Виджеты могут иметь готовые слоты.
- В этом случае создавать собственные слоты нет необходимости.
- Просто требуется соединить сигнал одного виджета со слотом другого.

Пример использования стандартных слотов



Пример использования стандартных слотов

```
// Конструктор класса окна
StandartSlots::StandartSlots(QWidget *parent,
                             Qt::WFlags flags)
 : QMainWindow(parent, flags)
{
    .....

    // Связываем слот одного виджета со слотом
    // другого
     connect(ui.pushButton, SIGNAL(clicked()),
             ui.checkBox, SLOT(click()));
}
```

Объявление и испускание сигнала

- Сигнал обычно объявляется как открытый метод класса окна в специальном разделе **`signals`**.
- Сигнал не имеет возвращаемого значения (возвращаемое значение типа **`void`**).
- Сигнал может иметь аргументы.
- Сигнал не имеет тела (кода).
- Испускание сигнала аналогично вызову функции, но предваряется ключевым словом **`emit`**

Объявление и испускание сигнала

```
class MyWindow : public QMainWindow
{
...
public signals:
    void create_successful(bool);
private:
    int *mass;
};
// Конструктор класса
MyWindow::MyWindow(QWidget *parent,Qt::WFlags flags)
 : QMainWindow(parent, flags)
{
    mass = (int *)malloc(1000*sizeof(int));
    // Вызов сигнала
    emit  create_successful(mass != NULL);
}
```