

Глава 6. Объектная библиотека Qt

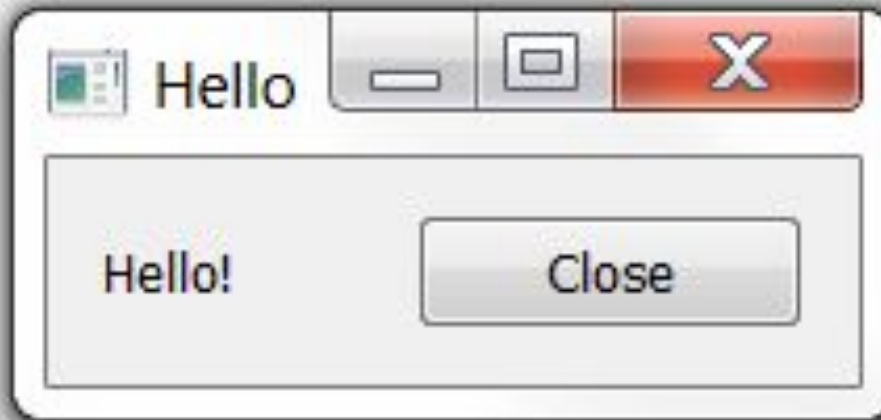
МГТУ им. Н.Э. Баумана
Факультет Информатика и системы управления
Кафедра Компьютерные системы и сети
Лектор: д.т.н., проф.
Иванова Галина Сергеевна

6.1 Простейшая программа с Qt интерфейсом

Пример 6.1 Приложение Hello

Объект QWidget –
виджет управления
ОКНОМ

Объект
QLabel



Окно
приложения

Объект
QPushButton

Каждому элементу оконного интерфейса соответствует *виджет* – объект интерфейсного класса библиотеки Qt.

Виджеты визуальных компонентов управляются контейнером – главным виджетом – виджетом управления окном приложения. В качестве такого виджета может использоваться объект класса QWidget.

Текст программы

```
#include <QApplication>
#include <QLabel>
#include <QPushButton>
#include <QHBoxLayout>

int main(int argc, char *argv[]) {
    QApplication app(argc, argv);
    QWidget win;
    win.setWindowTitle("Hello");
    QLabel *helloLabel=new QLabel("Hello!", &win);
    QPushButton *exitButton=
        new QPushButton("Close", &win);
    QHBoxLayout *layout = new QHBoxLayout(&win);
    layout->addWidget(helloLabel);
    layout->addWidget(exitButton);

    QObject::connect(exitButton, SIGNAL(clicked(bool)),
        &win, SLOT(close()));

    win.show();
    return app.exec();
}
```

Объект-приложение

Объект управления окном

Метка

Кнопка

Компоновщик

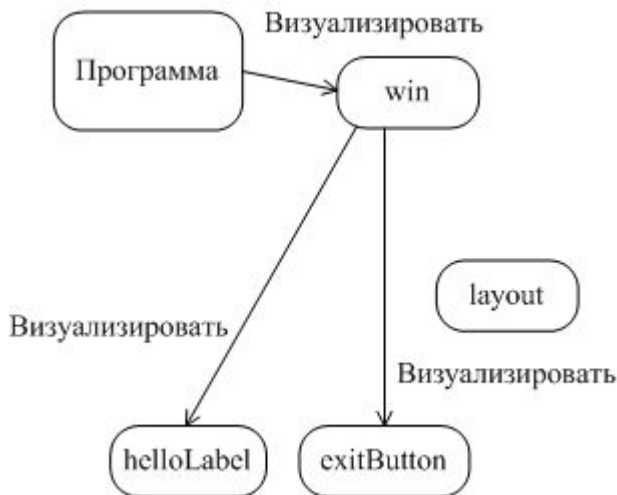
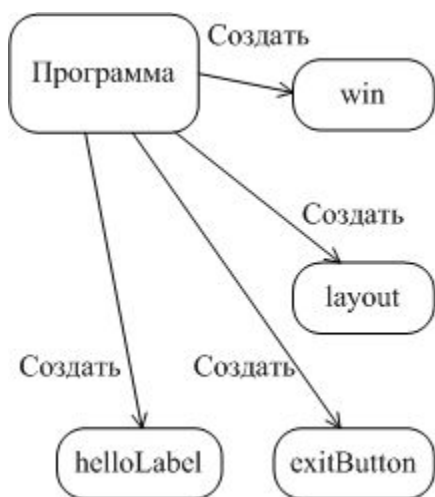
Связь сигнала со слотом

Визуализация окна

Запуск цикла обработки сообщений

Контейнерные свойства виджетов

Объект класса QWidget – win – контейнер, который отвечает за визуализацию компонентов и освобождение ими памяти.

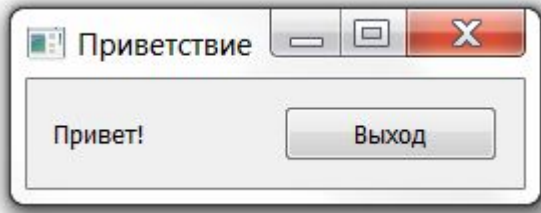


Компоновщик – контейнер, который отвечает за политику изменения размеров визуальных компонентов.



6.2 Создание класса окна

Вид окна интерфейса



Пример 6.2. Файл hello.h:

```
#ifndef hello_h
#define hello_h
#include <QWidget>
#include <QLabel>
#include <QPushButton>
class Win: public QWidget
{
    QLabel *helloLabel;
    QPushButton *exitButton;
public:
    Win(QWidget *parent = nullptr);
};
#endif
```

Диаграмма
компоновки приложения

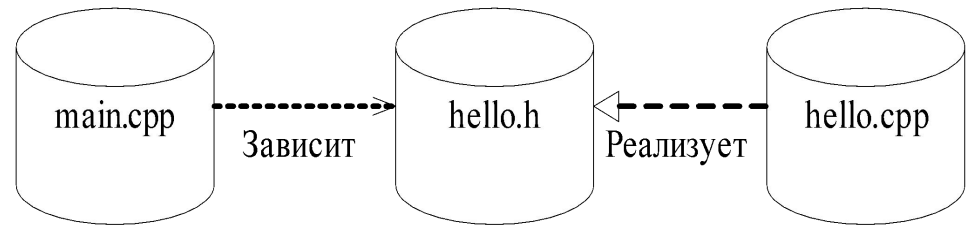
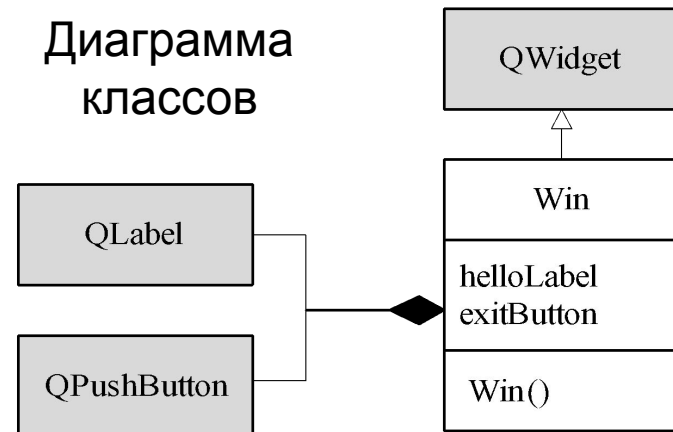


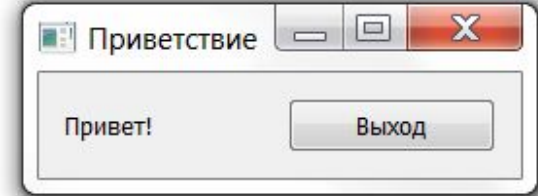
Диаграмма
классов



Конструктор класса окна и русификация интерфейса

Файл hello.cpp:

```
#include "hello.h"
#include <QTextCodec>
#include <QHBoxLayout>
Win::Win(QWidget *parent) : QWidget(parent)
{
    QTextCodec *codec =
        QTextCodec::codecForName("Windows-1251");
    setWindowTitle(codec->toUnicode("Приветствие"));
    helloLabel = new QLabel(codec->toUnicode("Привет!"), this);
    exitButton =
        new QPushButton(codec->toUnicode("Выход"), this);
    QHBoxLayout *layout = new QHBoxLayout(this);
    layout->addWidget(helloLabel);
    layout->addWidget(exitButton);
    connect(exitButton, SIGNAL(clicked(bool)),
            this, SLOT(close()));
}
```



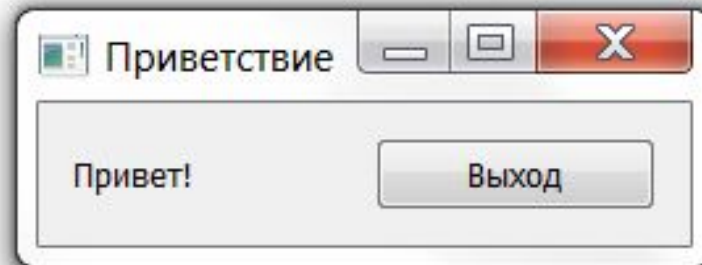
Перекодировщик

Основная программа

Файл main.cpp:

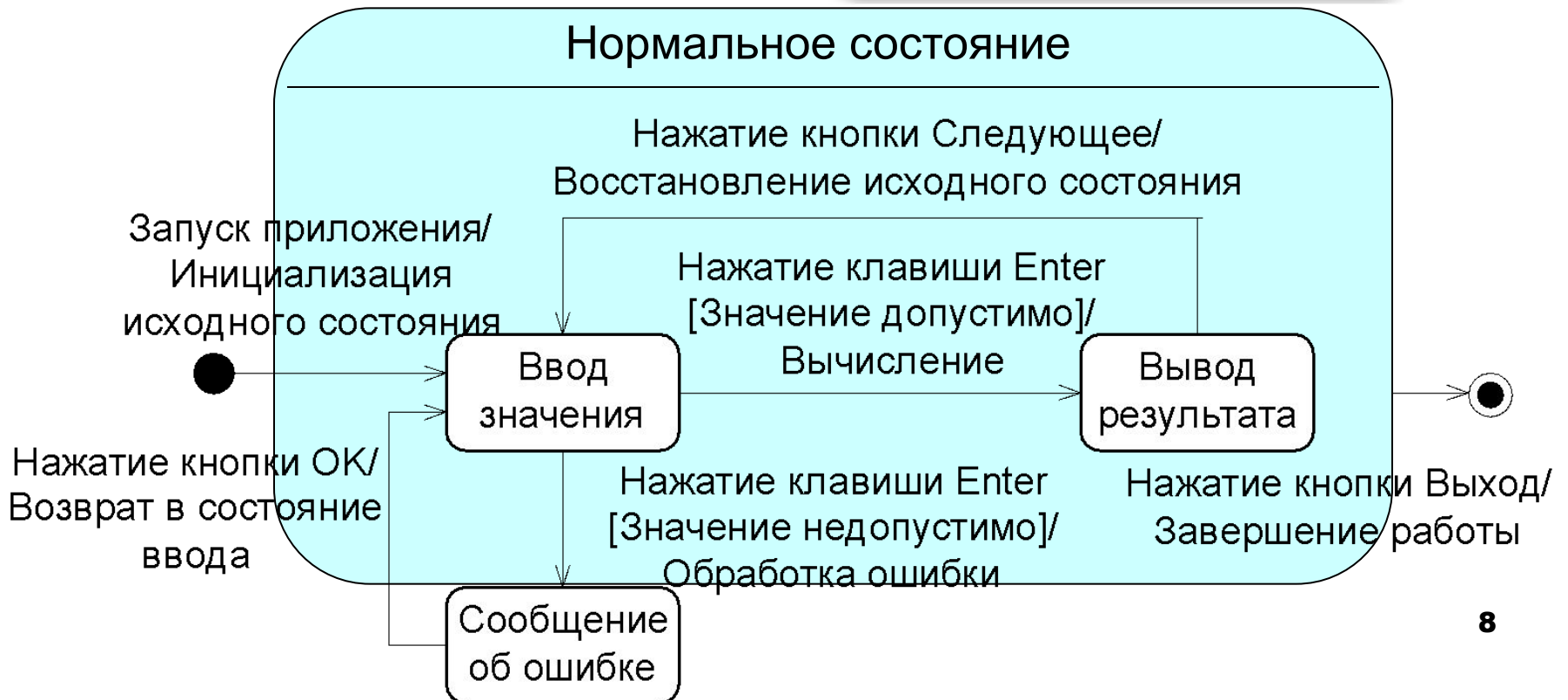
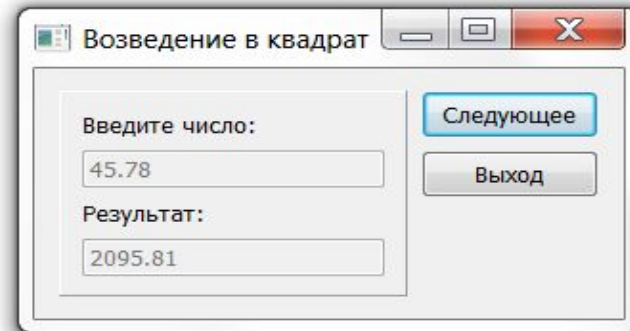
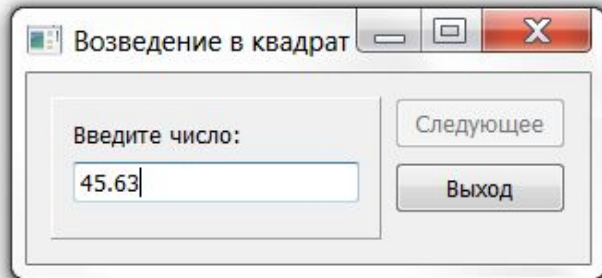
```
#include "hello.h"
#include <QApplication>

int main(int argc, char *argv[])
{
    QApplication app(argc, argv);
    Win win;
    win.show();
    return app.exec();
}
```



6.3 Механизм сигналов и слотов

Пример 6.3 Возведение числа в квадрат. Объявление новых слотов.



Файл win.h. Описание класса окна

```
#ifndef win_h
```

```
#define win_h
```

```
#include <QWidget>
```

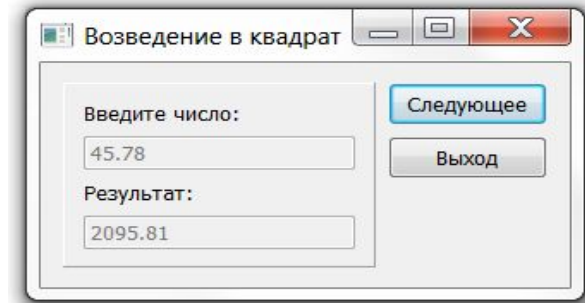
```
#include <QFrame>
```

```
#include <QLabel>
```

```
#include <QLineEdit>
```

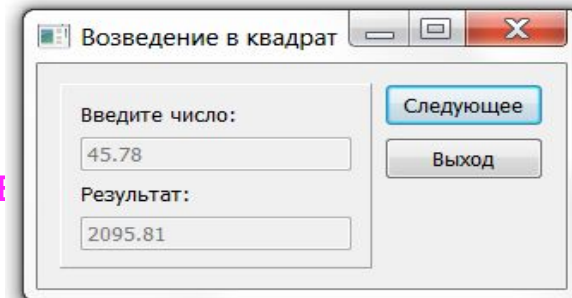
```
#include <QPushButton>
```

```
#include <QValidator>
```



Файл win.h. Описание класса окна

```
class Win:public QWidget           // класс окна
{
    Q_OBJECT // макрос Qt для реализации сигналов
protected:
    QTextCodec *codec;           // перекодировщик
    QFrame *frame;              // рамка
    QLabel *inputLabel;         // метка ввода
    QLineEdit *inputEdit;       // строчный редактор ввода
    QLabel *outputLabel;        // метка вывода
    QLineEdit *outputEdit;      // строчный редактор вывода
    QPushButton *nextButton;    // кнопка Следующее
    QPushButton *exitButton;    // кнопка Выход
public:
    Win(QWidget *parent = nullptr); // конструктор
public slots:
    void begin();               // инициализация интерфейса
    void calc();                // реализация вычислений
};
```



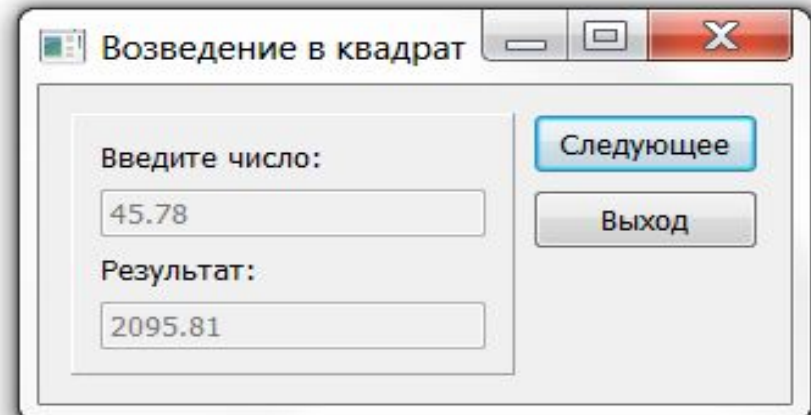
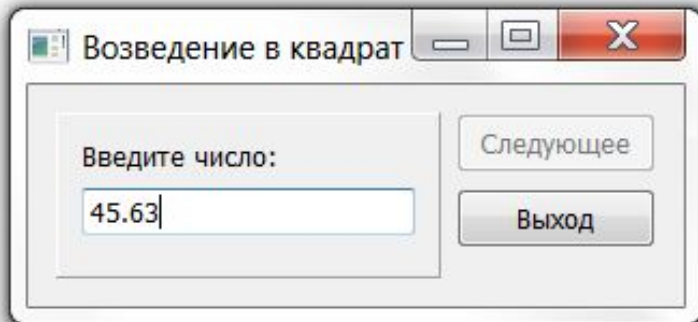
Файл win.h. Описание класса валидатора

```
class StrValidator:public QValidator // класс проверки ввода
{
public:
    StrValidator(QObject *parent):QValidator(parent){}
    virtual State validate(QString &str,int &pos)const;
    {
        return Acceptable; // метод всегда принимает
                            // вводимую строку
    }
};
#endif
```

Объект-валидатор при создании связывается с объектом строчного редактора. Метод `validate()` автоматически вызывается для проверки вводимой строки. Если метод возвращает `Acceptable`, то редактор генерирует сигналы `editingFinished()` – завершение редактирования и `returnPressed()` – нажатие клавиши Enter.

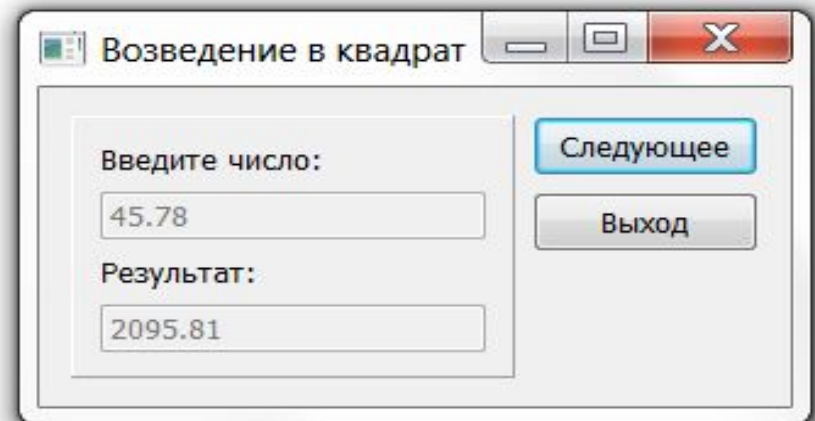
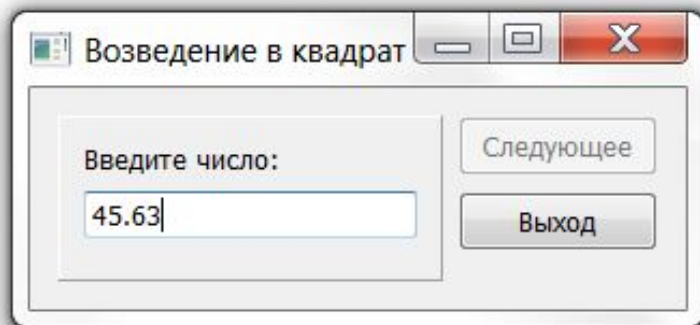
Файл win.cpp. Создание и настройка виджетов

```
#include "win.h"
#include <QTextCodec>
#include <QVBoxLayout>
Win::Win(QWidget *parent) : QWidget(parent)
{
    codec = QTextCodec::codecForName("Windows-1251");
    setWindowTitle(codec->toUnicode("Возведение в квадрат"));
    frame = new QFrame(this);
    frame -> setFrameShadow(QFrame::Raised);
    frame -> setFrameShape(QFrame::Panel);
    inputLabel =
        new QLabel(codec->toUnicode("Введите число:"), this);
    inputEdit = new QLineEdit("", this);
    StrValidator *v=new StrValidator(inputEdit);
    inputEdit -> setValidator(v);
```



Файл win.cpp. Создание и настройка виджетов

```
outputLabel =  
    new QLabel(codec->toUnicode("Результат:"), this);  
outputEdit = new QLineEdit("", this);  
nextButton =  
    new QPushButton(codec->toUnicode("Следующее"), this);  
exitButton =  
    new QPushButton(codec->toUnicode("Выход"), this);
```

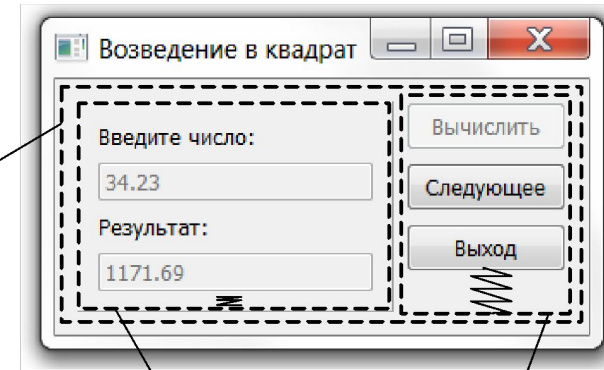


Файл win.cpp. Компоновка виджетов

```
QVBoxLayout *vLayout1 = new QVBoxLayout(frame) ;  
vLayout1->addWidget(inputLabel) ;  
vLayout1->addWidget(inputEdit) ;  
vLayout1->addWidget(outputLabel) ;  
vLayout1->addWidget(outputEdit) ;  
vLayout1->addStretch() ;
```

```
QVBoxLayout *vLayout2 = new QVBoxLayout() ;  
vLayout2->addWidget(nextButton) ;  
vLayout2->addWidget(exitButton) ;  
vLayout2->addStretch() ;
```

```
QHBoxLayout *hLayout = new QHBoxLayout(this) ;  
hLayout->addWidget(frame) ;  
hLayout->addLayout(vLayout2) ;
```



QVBoxLayout

QVBoxLayout

Файл win.cpp. Инициализация интерфейса и СВЯЗЫВАНИЕ СИГНАЛОВ И СЛОТОВ

```
begin() ;    // инициализация интерфейса
```

```
// связь нажатия кнопки Выход и закрытия главного окна
```

```
connect(exitButton, SIGNAL(clicked(bool)) ,  
        this, SLOT(close())) ;
```

```
// связь нажатия кнопки Следующее и инициализации интерфейса
```

```
connect(nextButton, SIGNAL(clicked(bool)) ,  
        this, SLOT(begin())) ;
```

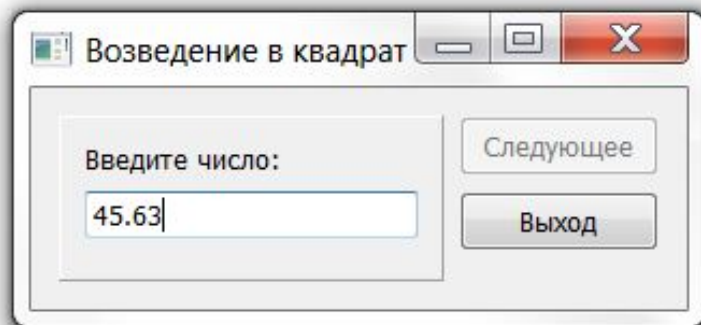
```
// связь нажатия клавиши Enter и вычислений
```

```
connect(inputEdit, SIGNAL(returnPressed()) ,  
        this, SLOT(calc())) ;
```

```
}
```

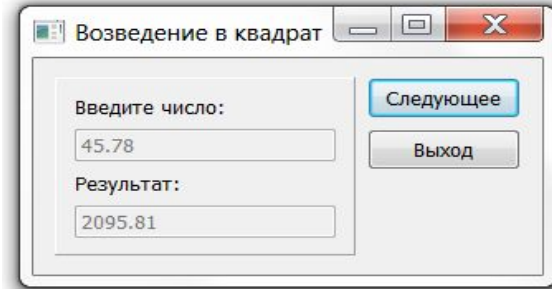
Файл win.cpp. Метод начальной настройки (инициализации) интерфейса

```
void Win::begin()  
{  
    inputEdit->clear(); // очистка строки ввода  
    nextButton->setEnabled(false); // деактивация кнопки Следующее  
    nextButton->setDefault(false); // отмена активации кнопки  
                                // Следующее при нажатии Enter  
    inputEdit->setEnabled(true); // активация строки ввода  
    outputLabel->setVisible(false); // сокрытие метки результата  
    outputEdit->setVisible(false); // сокрытие строки результата  
    inputEdit->setFocus(); // установка фокуса на строку ввода  
}
```



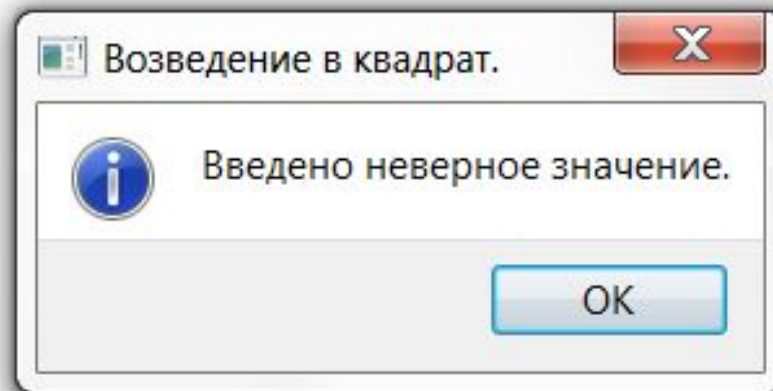
Файл win.cpp. Вычисление результата

```
void Win::calc()
{
    bool Ok=true;
    float r,a;
    QString str=inputEdit->text(); // копирование введенной строки
    a=str.toDouble(&Ok);           // преобразование строки в число
    if (Ok)                         // если преобразование успешно, то
    {
        r=a*a;                      // возводим число в квадрат
        str.setNum(r);              // преобразуем число в строку
        outputEdit->setText(str);   // заносим результат в окно результата
        inputEdit->setEnabled(false); // деактивируем окно ввода
        outputEdit->setEnabled(false); // деактивируем окно вывода
        outputLabel->setVisible(true); // показываем метку результата
        outputEdit->setVisible(true); // показываем окно результата
        nextButton->setDefault(true); // назначаем Следующее кнопкой
        // активируемой по нажатию клавиши Enter
        nextButton->setFocus();      // устанавливаем фокус и активируем
        nextButton->setEnabled(false); // кнопку Следующее
    }
}
```



Файл win.cpp. Метод вычислений: выдача сообщения об ошибке ввода

```
else
    if (!str.isEmpty())
    {
        QMessageBox msgBox(QMessageBox::Information,
            codec->toUnicode("Возведение в квадрат."),
            codec->toUnicode("Введено неверное значение."),
            QMessageBox::Ok);
        msgBox.exec();
    }
}
```



Файл win.cpp. Метод проверки строки

```
QValidator::State
```

```
    StrValidator::validate(QString &str,int &pos) const
```

```
{
```

```
    return Acceptable; // метод всегда принимает  
                        // вводимую строку
```

```
}
```

Файл main.cpp

```
#include "win.h"
#include <QApplication>

int main(int argc, char *argv[])
{
    QApplication app(argc, argv);
    Win win;
    win.show();
    return app.exec();
}
```

6.4 Обработка событий. Рисование. Таймеры

Пример 6.4 Создание движущихся изображений

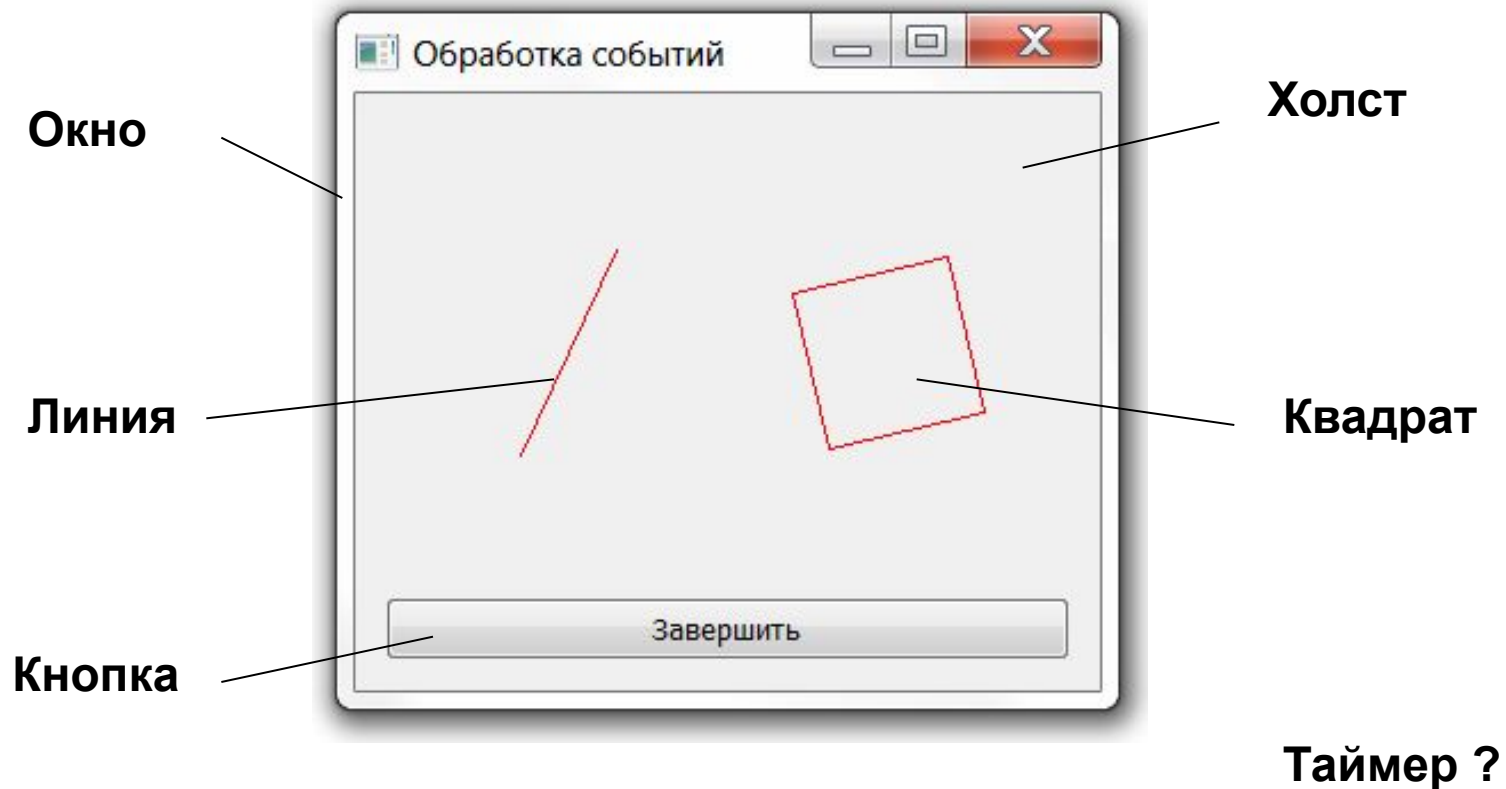


Диаграмма объектов приложения

Кроме объекта Приложение программа включает 6 объектов:

- окно;
- кнопка Выход;
- холст для рисования;
- таймер;
- линия;
- квадрат.

Диаграмма объектов

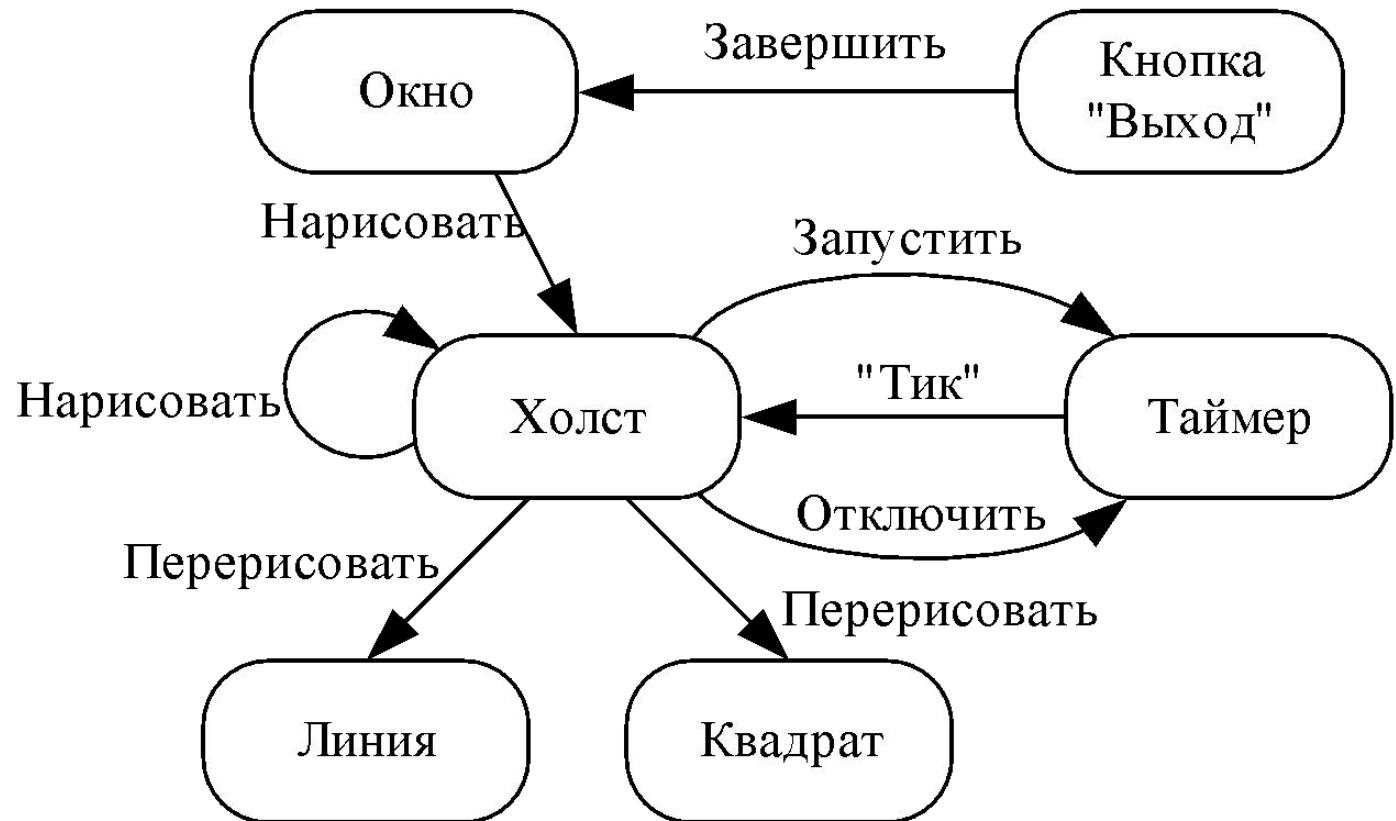
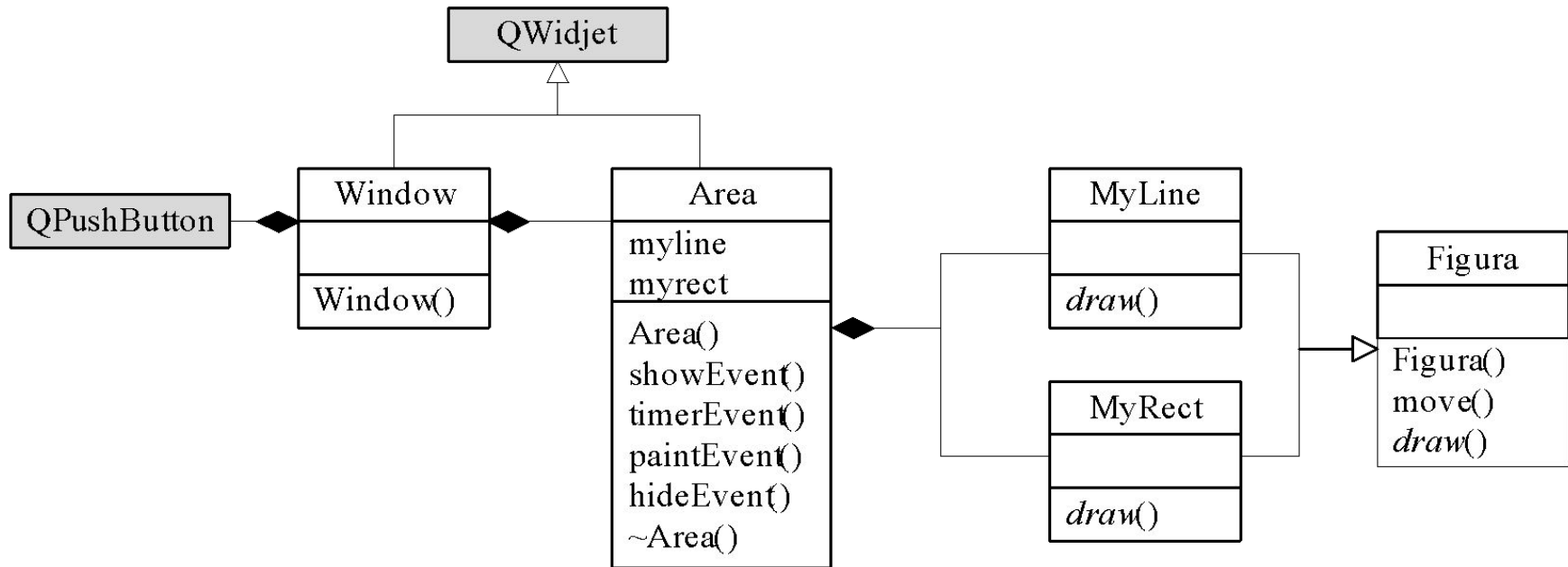


Диаграмма классов приложения



События:

- `showEvent()` – при визуализации области рисования – включаем таймер;
- `timerEvent()` – при получении сообщения от таймера – инициируем перерисовку окна;
- `paintEvent()` – при получении запроса на перерисовку области рисования – перерисовываем фигуры;
- `hideEvent()` – при сокрытии области рисования – выключаем таймер.

Диаграмма последовательностей действий

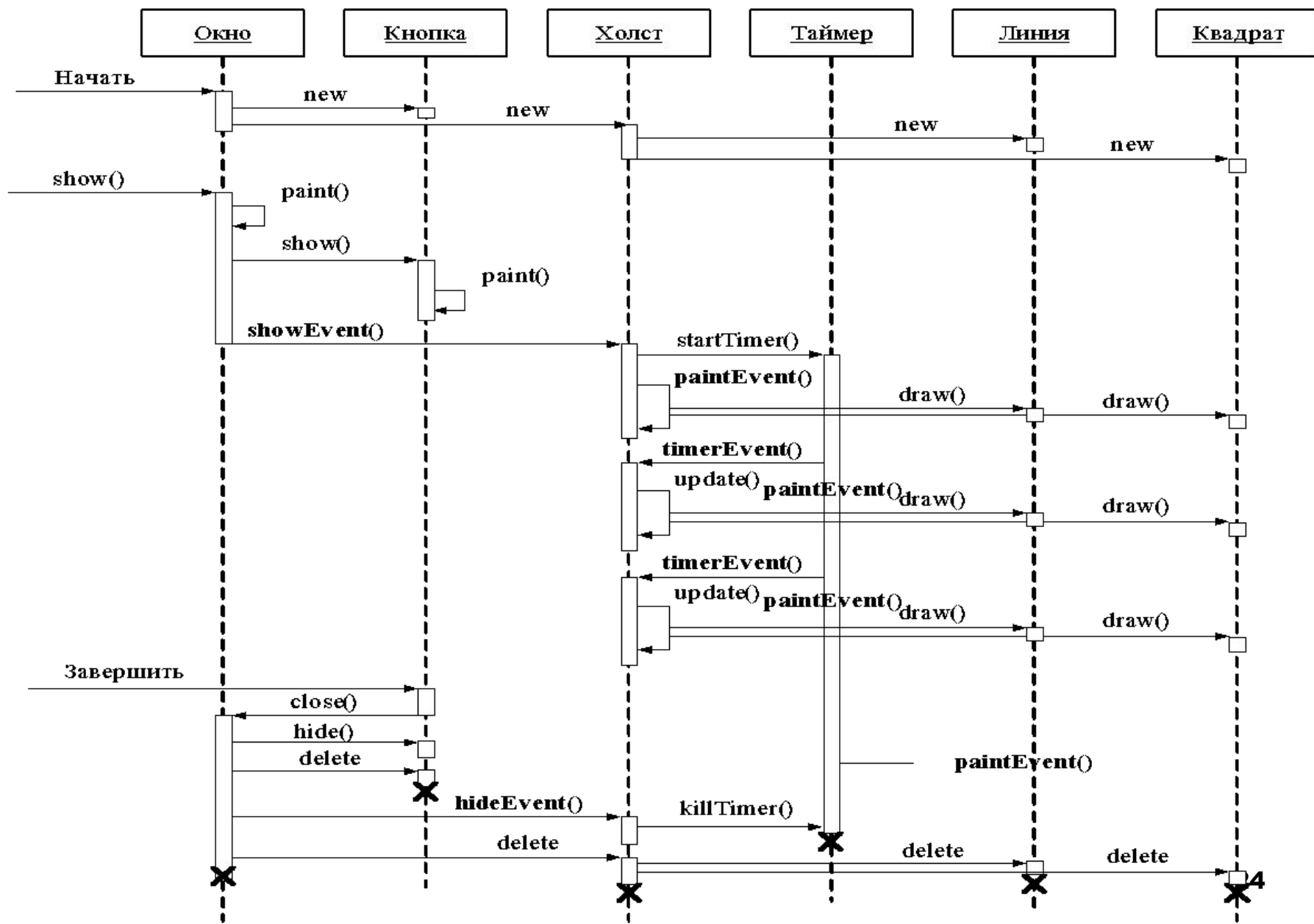
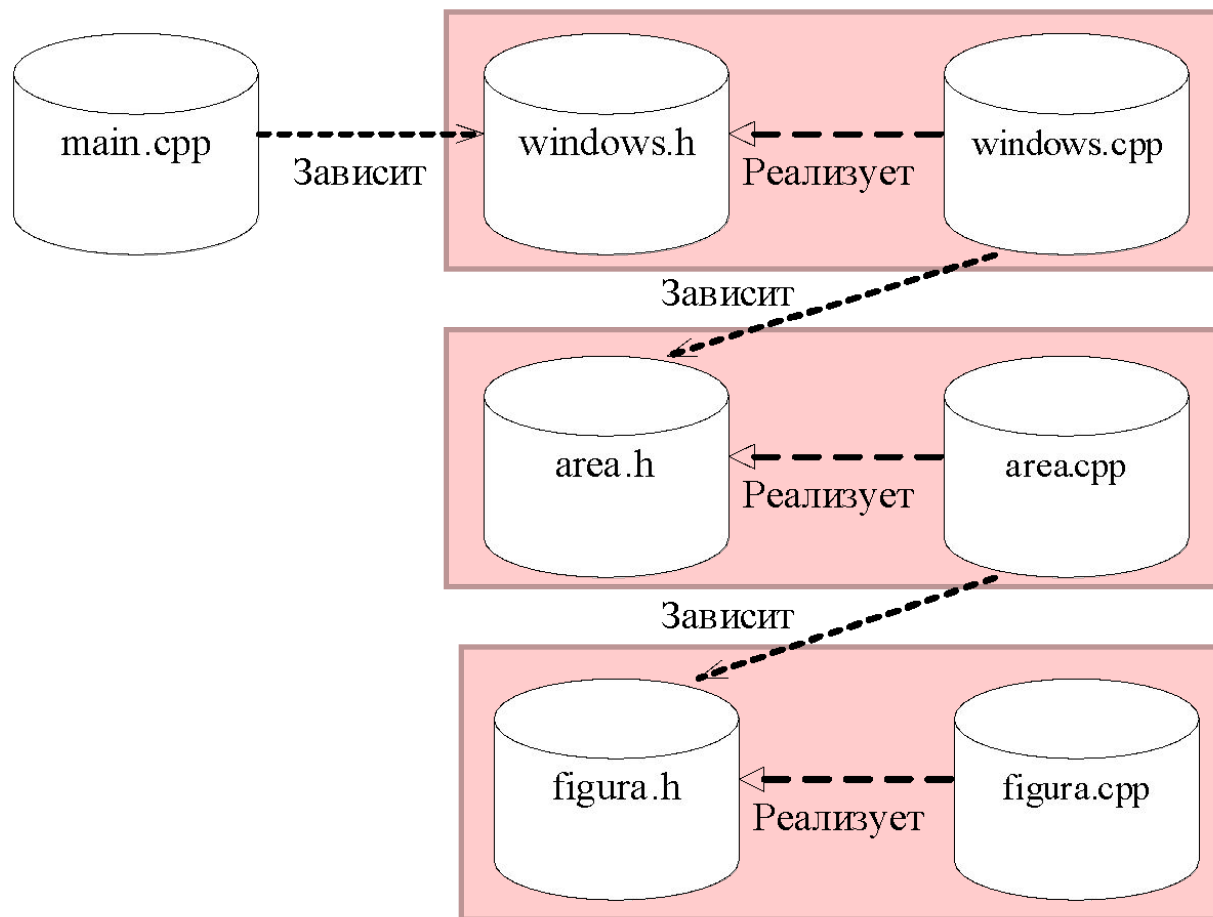


Диаграмма компоновки приложения

Модуль Окно
содержит описание
класса окна
приложения.

**Модуль Область
рисования** содержит
описание класса Холст.

Модуль Фигура
содержит описание
классов Фигура, Линия
и квадрат.



Файл figura.h. Класс Фигура:

```
#ifndef figura_h
#define figura_h
#include <QPainter>
class Figura
{
protected:
    int x,y,halflen,dx,dy,r;
    virtual void draw(QPainter *Painter)=0;
public:
    Figura(int X,int Y,int Halflen):
        x(X),y(Y),halflen(Halflen) {}
    void move(float Alpha,QPainter *Painter);
    virtual ~Figura(){} // необходим для полиморфного объекта
};
```

Файл figura.h. Классы Линия и Квадрат:

```
class MyLine:public Figura
{
protected:
    void draw(QPainter *Painter);
public:
    MyLine(int x,int y,int halflen):Figura(x,y,halflen){}
};
class MyRect:public Figura
{
protected:
    void draw(QPainter *Painter);
public:
    MyRect(int x,int y,int halflen):Figura(x,y,halflen){}
};
#endif
```

Файл figura.cpp

```
#include "figura.h"
#include <math.h>
void Figura::move(float Alpha, QPainter *Painter)
{
    dx=static_cast<int>(halfflen*cos(Alpha));
    dy=static_cast<int>(halfflen*sin(Alpha));
    draw(Painter); // необходим сложный полиморфизм!
}
void MyLine::draw(QPainter *Painter)
{
    Painter->drawLine(x+dx,y+dy,x-dx,y-dy);
}
void MyRect::draw(QPainter *Painter)
{
    Painter->drawLine(x+dx,y+dy,x+dy,y-dx);
    Painter->drawLine(x+dy,y-dx,x-dx,y-dy);
    Painter->drawLine(x-dx,y-dy,x-dy,y+dx);
    Painter->drawLine(x-dy,y+dx,x+dx,y+dy);
}
```

Файл area.h

```
#ifndef AREA_H
#define AREA_H
#include "figura.h"
#include <QWidget>
#include <QTimer>
class Area : public QWidget
{
    int myTimer; // идентификатор таймера
    float alpha; // угол поворота
public:
    Area(QWidget *parent = nullptr);
    ~Area();
    MyLine *myline; // указатели на объекты фигур
    MyRect *myrect;
protected:
    // обработчики событий
    void paintEvent(QPaintEvent *event);
    void timerEvent(QTimerEvent *event);
    void showEvent(QShowEvent *event);
    void hideEvent(QHideEvent *event);
};
#endif
```

Файл area.cpp. Обработка событий визуализации и перерисовки

```
#include "area.h"
#include <QTimerEvent>
Area::Area(QWidget *parent) : QWidget(parent)
{
    setFixedSize(QSize(300,200)); // фиксируем размер Холста
    myline=new MyLine(80,100,50); // создаем объект Линия
    myrect=new MyRect(220,100,50); // создаем объект Квадрат
    alpha=0;
}
void Area::showEvent(QShowEvent *)
{
    myTimer=startTimer(50); // включаем таймер
}
void Area::paintEvent(QPaintEvent *)
{
    QPainter painter(this); // создаем контент рисования на Холсте
    painter.setPen(Qt::red); // задаем красное Перо
    myline->move(alpha, &painter); // рисуем Линию
    myrect->move(alpha*(-0.5), &painter); // рисуем Квадрат
}
```

Файл area.cpp. Обработка событий таймера и сокрытия Холста

```
void Area::timerEvent (QTimerEvent *event)
{
    if (event->timerId() == myTimer) // если наш таймер
    {
        alpha += 0.2;
        update (); // ОБНОВИТЬ ВНЕШНИЙ ВИД
    }
    else
        QWidget::timerEvent (event); // иначе передать далее
}
void Area::hideEvent (QHideEvent *)
{
    killTimer (myTimer); // ВЫКЛЮЧИТЬ таймер
}
Area::~Area ()
{
    delete myline;
    delete myrect;
}
```

Файл window.h

```
#ifndef window_h
#define window_h
#include "area.h"
#include <QWidget>
#include <QTextCodec>
#include <QPushButton>

class Window : public QWidget
{
protected:
    QTextCodec *codec;
    Area * area;           // область отображения рисунка
    QPushButton * btn;
public:
    Window();
};
#endif
```


Файл window.cpp

```
#include "window.h"
#include <QVBoxLayout>

Window::Window()
{
    codec = QTextCodec::codecForName("Windows-1251");
    this->setWindowTitle(codec->toUnicode("Обработка событий"));
    area = new Area( this );
    btn = new QPushButton(codec->toUnicode("Завершить"), this );
    QVBoxLayout *layout = new QVBoxLayout(this);
    layout->addWidget(area);
    layout->addWidget(btn);
    connect(btn, SIGNAL(clicked(bool)), this, SLOT(close()));
};
```

Файл main.cpp

```
#include "window.h"
#include <QApplication>

int main(int argc, char *argv[])
{
    QApplication appl(argc, argv);
    Window win;
    win.show();
    return appl.exec();
}
```