Объектно-ориентированное программирование (ООП)

Язык C++

Три кита ООП

- Инкапсуляция (encapsulation)
- Полиморфизм (polymorphism)
- Наследование (inheritance)

Инкапсуляция

- Объединение данных и функций их обработки
- Скрытие информации, ненужной для использования данных

Полиморфизм

- в биологии - наличие в пределах одного вида резко отличных по облику особей

- в языках программирования - взаимозаменяемость объектов с одинаковым интерфейсом

  «Один интерфейс, множество реализаций».

Наследование

- Возможность создания иерархии классов
- Наследование потомками свойств предков
- Возможность изменения наследуемых свойств и добавления новых

Классы. Инкапсуляция. Полиморфизм.

- Основные понятия:
  - Описание класса
  - Конструкторы и деструкторы
  - Ссылки и указатели. Указатель **this**
  - Функции и операции
  - Перегрузка функций и операторов

Описание класса

Класс – это способ описания сущности, определяющий состояние и поведение, зависящее от этого состояния, а также правила для взаимодействия с данной сущностью.

С точки зрения программирования, **класс** является абстрактным типом данных, определяемым пользователем, который содержит набор данных (полей, атрибутов, членов класса) и функций для работы с ними (методов).

Описание класса

```
class myclass
{
private:                //ключ доступа
    int a;              //члены-данные, свойства
    float b;            //структура в языке C
public:
     void setvalue(int, float); //члены-функции,
     int geta();            //методы,
     float getb();
};
```

Описание класса

```
void myclass::setvalue(int sa, float sb)
{
  a=sa;
  b=sb;  //или this->b=sb;
}

int myclass::geta()
{
  return a;
}

float myclass::getb()
{
  return b;
}

void main()
{
  myclass mc;
        cout<<mc.geta()<<"\n"<<mc.getb()<<"\n";
  mc.setvalue(31, 3.5);
  cout<<mc.geta()<<"\n"<<mc.getb()<<"\n";
}
```

Конструкторы и деструкторы класса

```
#include <iostream>
using namespace std;

class myclass
{
private:
  int a;
  float b;
  int *m;
public:
    myclass();                    //конструктор по умолчанию
  myclass(int, float);
  myclass(int, float, int*);
  myclass(const myclass &); //конструктор копирования
  ~myclass();                   //деструктор
  void print();
};
```

Конструкторы и деструкторы класса

```
myclass::myclass()
{
        a=0;
        b=0.0;
  m = new int[5];
}

myclass::myclass(int n, float f)
{
  m = new int[5];
  this->a=n;
  this->b=f;
}

myclass::myclass(int n, float f, int *p)
{
  m = new int[5];
  a=n;
  b=f;
  for (int i=0; i<5; i++)
        m[i]=p[i];

}
```

```
myclass::myclass(const myclass & mc)
{
        if (mc.m)
        {   m= new int[5];
            for (int i=0; i<5; i++) m[i]=mc.m[i];
        }
        else m=0;
        a=mc.a; b=mc.b;
}
myclass::~myclass()
{
        delete [] m;
}
void myclass::print()
{
        cout<<"a="<<a<<"\nb="<<b<<"\nm=";
        for (int i=0; i<5; i++)
        cout<<" "<<m[i];
        cout<<"\n";
}
void main()
{
        int dig[]={1,2,3,4,5};
        myclass mc(12, 25.6, dig);
        mc.print();
}
```

Указатели и ссылки

Указатель – переменная, значением которой является адрес некоторой области памяти.

```
int *a, n;        *a=10; a=&n;
float *b;     …..
char *c;      …..
void *f;      …..
```

Указатели и ссылки на объект

```
myclass *pmc, mc1,mc2(45, 3.5);
```

При объявлении указателя на объект выделяется память только для указателя!

```
pmc->a=23; //ошибка-не выделена память под объект
*pmc=mc1; pmc->a=23; (*pmc).b=12.05;
pmc=&mc2;
```

Указатели и ссылки

Ссылка – понятие, родственное указателю. Является скрытым указателем. Во всех случаях ее можно использовать как еще одно имя переменной

Ссылку можно:

1. Передавать в функцию
2. Возвращать из функции
3. Использовать как независимую переменную

При использовании ссылки как независимой переменной, она должна быть проинициирована при объявлении

1. myclass mc(12, 25.6, dig), &s=mc;

Указатели. Передача в функцию

```
void swap(int *a, int *b)
{
    int d;
    d=*a;
    *a=*b;
    *b=d;
}
void main()
{
    int a=10, b=20;
    cout<<"a="<<a<<" b="<<b<<"\n";
    swap(&a,&b);
    cout<<"a="<<a<<" b="<<b<<"\n";
}
```

Ссылки. Передача в функцию

```
void swp(int &a, int &b)
{
    int d;
    d=a;
    a=b;
    b=d;
}
void main()
{
    int a=10, b=20;
    cout<<"a="<<a<<" b="<<b<<"\n";
    swp(a,b);
    cout<<"a="<<a<<" b="<<b<<"\n";
}
```

Указатель **this**

C++ содержит специальный указатель **this**. Он автоматически передается любой функции-члену при ее вызове и указывает на объект, генерирующий вызов.

Перегрузка функций

- Сигнатурой функции называют список типов ее параметров и возвращаемого значения.
- В C++ можно определять функции с одним и тем же именем, но разной сигнатурой. Эта возможность называется перегрузкой функции.
- Перегрузка функций является проявлением полиморфизма.

Операторы

```
class myclass
{
private:
        int a;
        float b;
        int *m;
public:
        myclass();
        myclass(int, float);
        myclass(int, float, int*);
        myclass(const myclass &);
        ~myclass();
        void print();
        myclass & operator=(const myclass &);
};
```

Оператор * можно рассматривать как функцию с именем **operator***

Вызов этой функции происходит без операции «.»:
x=y;
или, что менее удобно:
x.operator=(y);

Оператор присваивания

```
myclass & myclass::operator=(const myclass &mc)
{
    m= new int[5];
    for (int i=0; i<5; i++)
     m[i]=mc.m[i];
    a=mc.a; b=mc.b;
    print();
    return *this;
}
```

Задание **1.** Строки

Реализовать класс **MyString**

```
class
    MyString
{
private:
   char *data;
   ...
};
```

Класс должен содержать:

1.Конструктор по умолчанию

2. Конструктор с параметром char*

3. Конструктор копирования

4. Деструктор

5. Функции для работы со строками (length, concat, compare, insert, print)

6. Операторы для работы со строками (=, +, +=, [])

Шаблоны

Шаблоны функций

Шаблоны классов

Шаблон позволяет отделить алгоритмы от конкретных типов данных.

Шаблон может применяться к любым типам данных без переписывания кода.

Шаблоны функций

- Шаблон функции – параметризованная (родовая, generic) функция, которая помимо обычных параметров имеет еще один – некоторый тип.
- Шаблоны функций чаще всего используются при создании функций, выполняющих одни и те же действия, но с данными различных типов.

Шаблоны функций

```
template <typename T>
T abs(T a)
{
	return (a >= 0) ? a : -a;
}

int main()
{
	int a=3, b=-10;
	float f=-5.5, g=0.07;
	cout<<abs<int>(a)<<"\n"<<abs(b)<<"\n";
	cout<<abs(f)<<"\n"<<abs<float>(g)<<"\n";
	return 0;
}
```

При вызове функции asb<T>() указывать явно параметр <T> необязательно.

Шаблоны классов

- Шаблон класса – параметризованный класс (родовой, generic), которому тип инкапсулированных в нем данных передается в качестве параметра.
- Чаще всего шаблоны используются при создании контейнерных классов

Шаблоны классов. Односвязный список

```
class LIST
{
        class Node
        {
        public:
          int dat;
          Node * next;
          Node (int d=0)
          {
                dat=d; next=0;
          }
        };
        Node * head;
public:
        LIST (){head=0;}
        ~LIST ();
        void insert_beg (int);
        void insert_end (int);
        void del (int);
        int find(int);
        void display();
};
```

```
template <class T>
class LIST
{
        class Node
        {
        public:
          T dat;
          Node * next;
          Node (T d=0)
          {
                dat=d;next=0;
          }
        };
        Node * head;
public:
        LIST (){head=0;}
        ~LIST ();
        void insert_beg (T);
        void insert_end (T);
        void del (T);
        int find(T);
        void display();
};
```

Шаблоны классов. Шаблоны функций

```
void  LIST::insert_beg (int data)
{
      Node * nel=new Node(data);
      nel->next=head;
      head=nel;
}
```

```
template <class T>
void  LIST <T>::insert_beg (T data)
{
      Node * nel=new Node(data);
      nel->next=head;
      head=nel;
}
```

Шаблоны классов. Использование

```
void main()
{
    LIST <char> lst;
    char i;
    do
    {
      cin>>i;
      if (i!=48)
            lst.insert_beg(i);
    } while (i!=48);
    lst.display();
}
```

Задание **2.** Шаблоны классов

- Реализовать шаблон класса List (методы, объявленные в классе).
- Реализовать конструктор копирования и оператор присваивания для класса List.

Наследование

Наследование – механизм, поддерживающий

- построение иерархии классов
- полиморфизм

```
class имя_произв_кл: ключ_доступа имя_баз_кл
{
    ….
};
```

Наследование. Ключевые понятия

- Ключи доступа
- Простое наследование. Конструкторы и деструкторы.
- Раннее и позднее связывание
- Виртуальные методы. Абстрактные классы
- Множественное наследование

Ключи доступа

Ключ доступа	Спецификатор в баз. классе	Доступ в произв. классе
private	private protected public	нет private private
protected	private protected public	нет protected protected
public	private protected public	нет protected public

Конструкторы и деструкторы

- Конструкторы не наследуются. В производном классе (ПК) должен быть собственный конструктор.

Порядок вызова конструкторов:

- Если в конструкторе ПК нет явного вызова конструктора базового класса (БК), то вызывается конструктор БК по умолчанию.
- Для иерархии, состоящей из нескольких уровней, конструкторы БК вызываются, *начиная с самого верхнего уровня,* а затем выполняется конструктор класса.

Конструкторы и деструкторы

- Деструкторы не наследуются. Если в производном классе (ПК) деструктор не определен, то он формируется по умолчанию и вызывает деструкторы всех БК.

Порядок вызова деструкторов:

- Деструкторы БК вызываются из деструктора ПК автоматически.
- Для иерархии, состоящей из нескольких уровней, деструкторы *вызываются в порядке, строго обратном* вызову конструкторов.

Виртуальные методы

- Указателю на БК можно присвоить значение адреса объекта любого ПК.

```
class Base_Class {void f();….};
class Derived_Class : public Base_Class {void f();…};
Base_Class *bc;
bc= new  Derived_Class; // указатель ссылается на объект ПК.
bc->f(); //вызывается метод Base_Class – механизм раннего связывания
```

Виртуальные методы

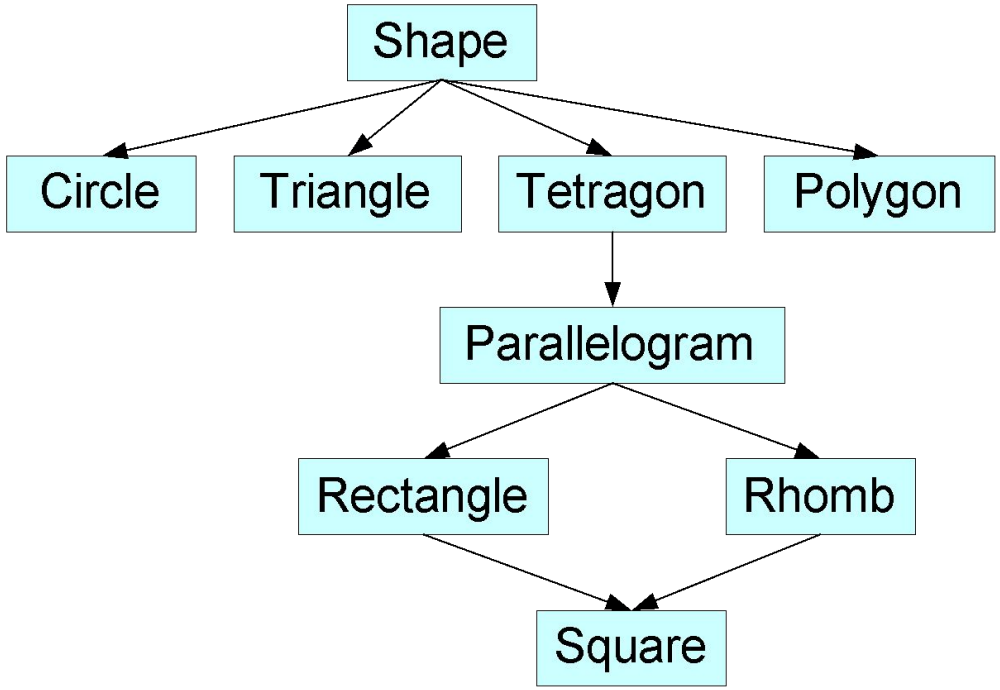
- Наряду с ранним связыванием, в C++ реализован механизм позднего связывания. Этот механизм реализован с помощью виртуальных методов.

```
class Base_Class {virtual void f();….};
class Derived_Class : public Base_Class {virtual void f()=0;…};
Base_Class *bc;
bc= new  Derived_Class; // указатель ссылается на объект ПК.
bc->f(); //вызывается метод  Derived_Class
```

Виртуальным называется метод, ссылка на который разрешается на этапе выполнения программы.

Задание **3.**

- Реализовать иерархию классов геометрических объектов



Задание **3.**

Класс Shape должен содержать такие свойства и методы:

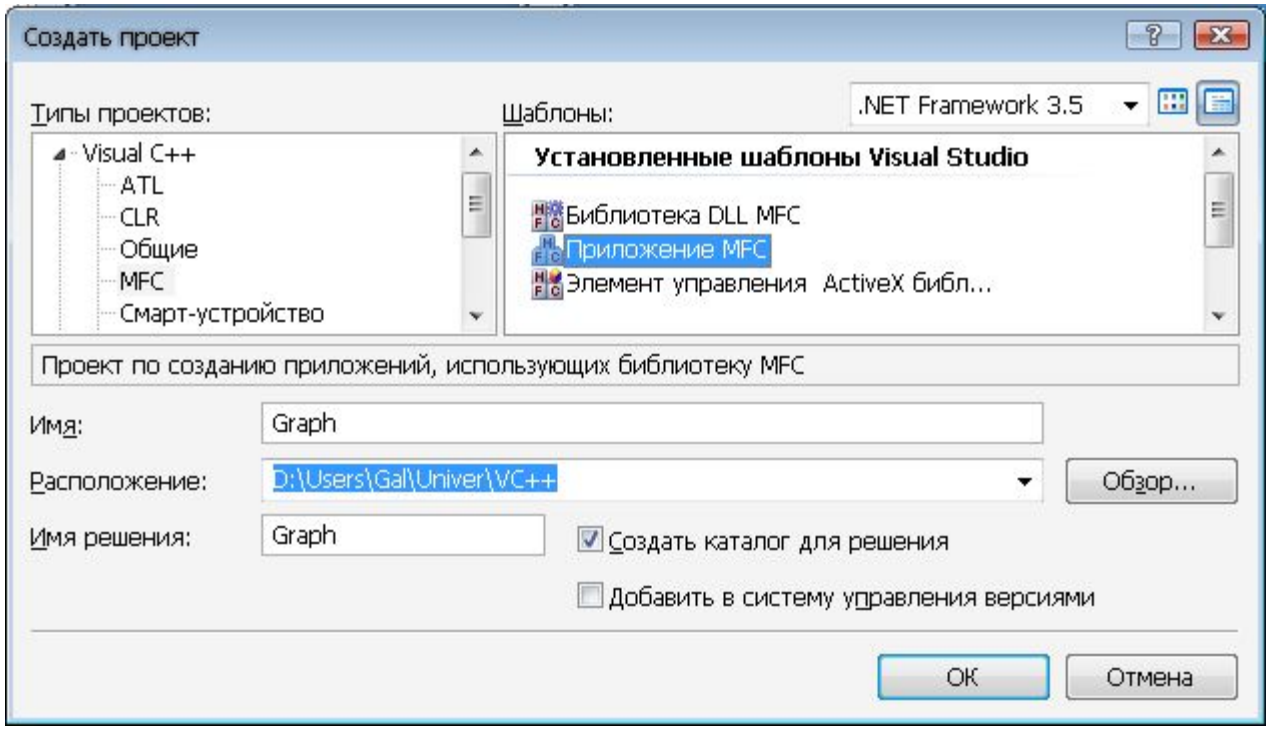
- Периметр и площадь фигуры;
- Параллельный перенос фигуры;
- Поворот фигуры;
- Печать информации о фигуре;
- Определение класса фигуры;

Методы в классе Shape виртуальные. Они должны определяться в конкретных классах.

Создание пользовательских интерфейсов средствами MFC

- Пакет **Microsoft Foundation Classes** (MFC) — библиотека на языке C++, разработанная Microsoft и призванная облегчить разработку GUI-приложений (**Graphical User Interface**) для Microsoft Windows путем использования богатого набора библиотечных классов.

Создание проекта. Шаг 1



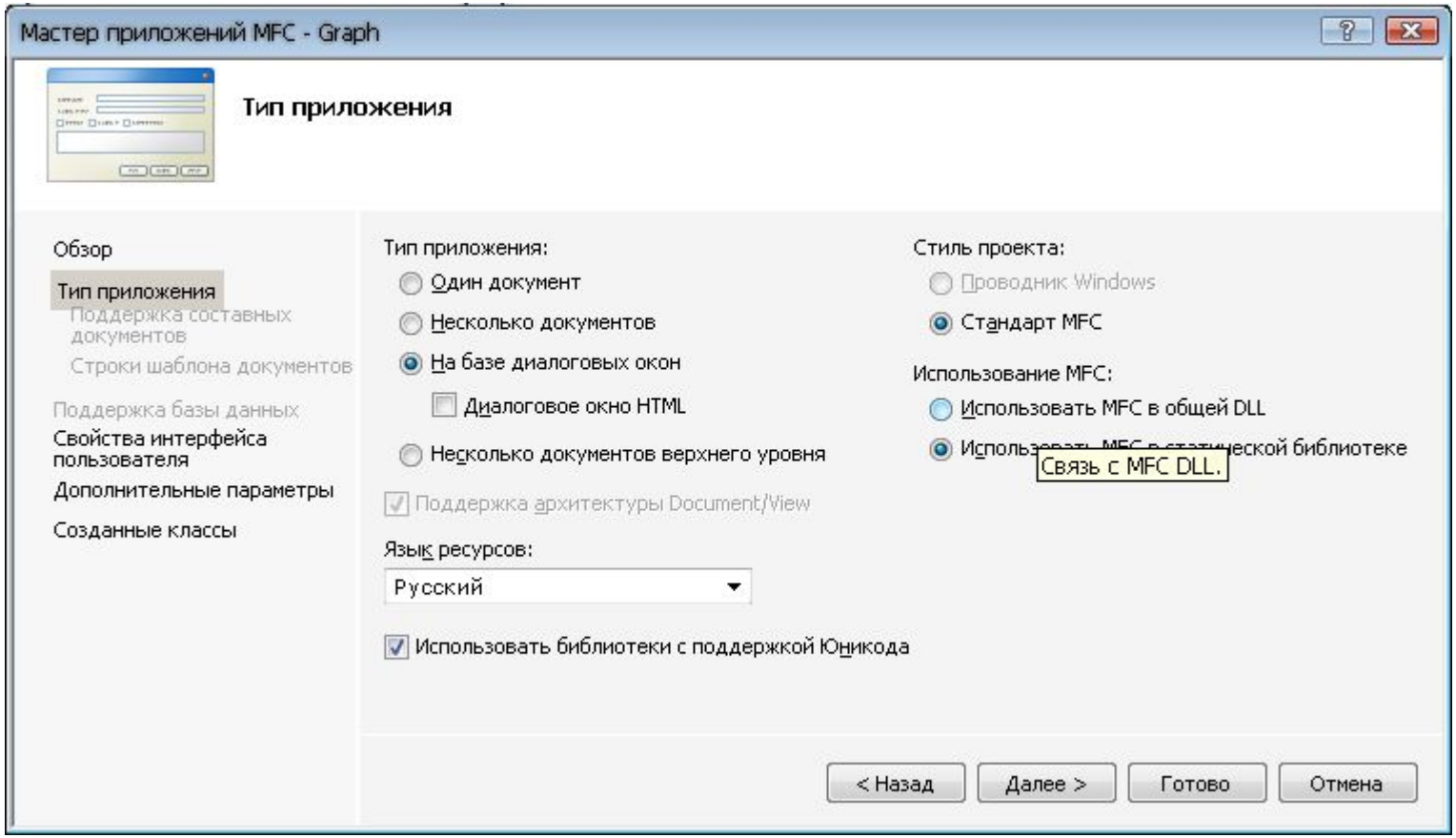
Создание проекта

В простейшем случае программа, написанная с помощью библиотеки MFC, содержит два класса, порождаемые от классов иерархии библиотеки: класс, предназначенный для создания приложения, и класс, предназначенный для создания окна.

```
class CTestGraphApp : public CWinApp
{
…
};
```

```
class CTestGraphDlg : public CDialog
{
};
```

Создание проекта. Шаг 2

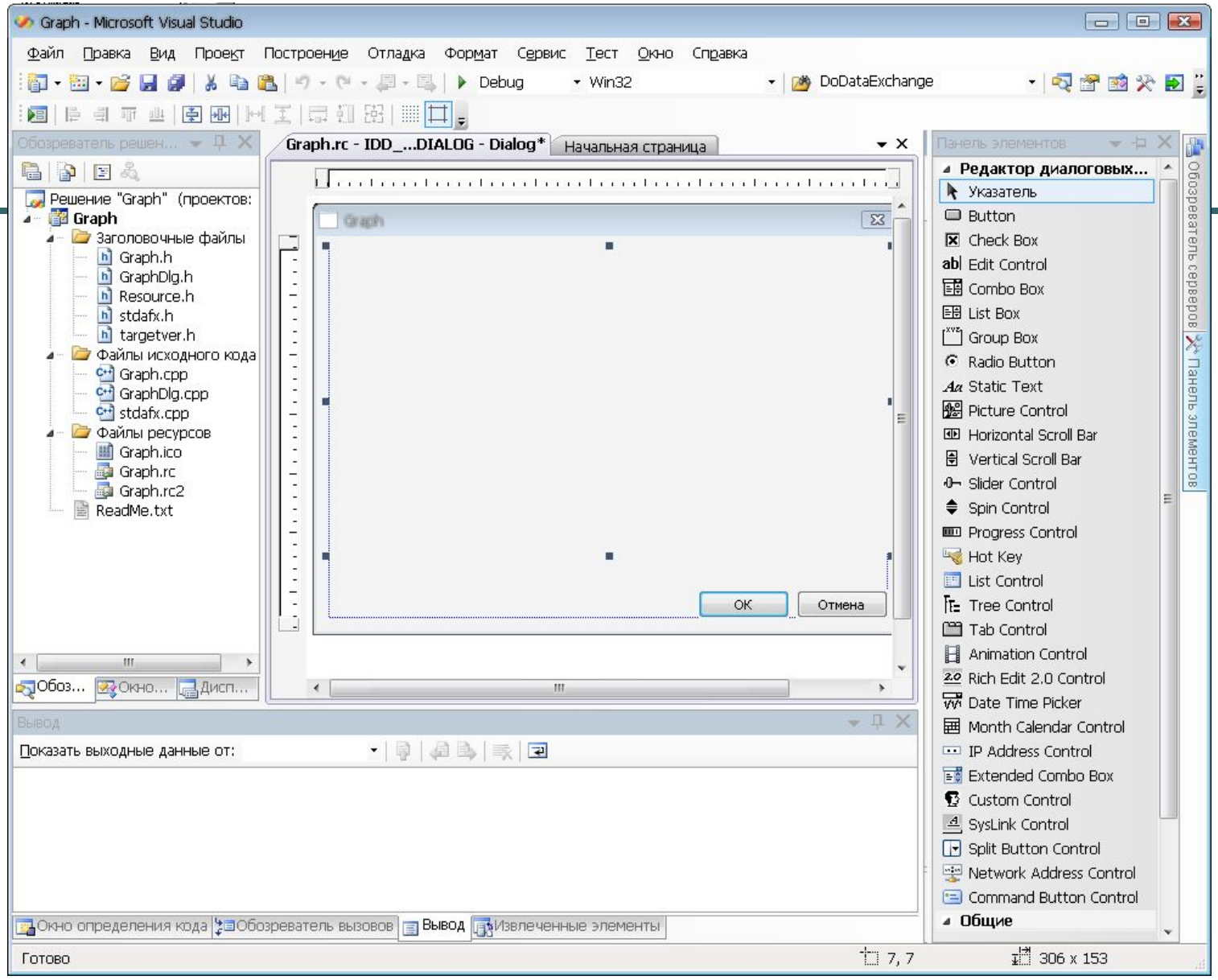


Создание проекта. Шаг 3

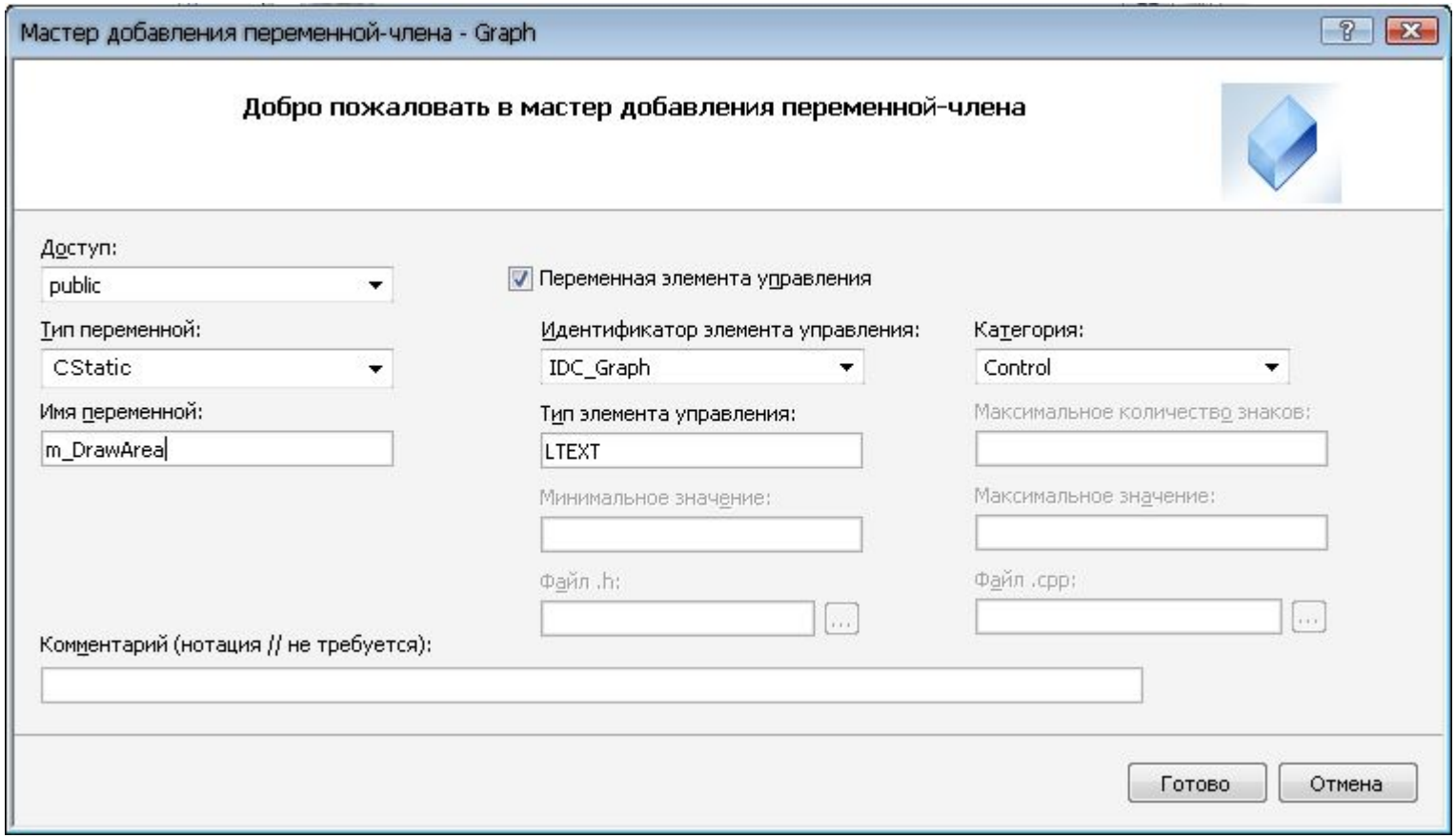
- Помещаем на диалог элемент, в котором будет рисоваться график (н-р Static Text)
- В окне свойств задаем ему уникальный ID IDC_GRAPH
- Добавляем в класс IDC_GRAPH переменную типа CStatic m_DrawArea;
- Связываем переменную m_DrawArea и элемент IDC_GRAPH:

DDX_Control(pDX, IDC_GRAPH, m_DrawArea);

в методе DoDataExchange


Graph - Microsoft Visual Studio
Файл Правка Вид Проект Построение Отладка Формат Сервис Тест Окно Справка
Debug
Win32
DoDataExchange
Обозреватель решен...
Решение "Graph" (проектов:
Graph
Заголовочные файлы
Graph.h
GraphDlg.h
Resource.h
stdafx.h
targetver.h
Файлы исходного кода
Graph.cpp
GraphDlg.cpp
stdafx.cpp
Файлы ресурсов
Graph.ico
Graph.rc
Graph.rc2
ReadMe.txt
Обоз...
Окно...
Дисп...
Graph.rc - IDD_...DIALOG - Dialog*
Начальная страница
Graph
OK
Отмена
Панель элементов
Редактор диалоговых...
Указатель
Button
Check Box
Edit Control
Combo Box
List Box
Group Box
Radio Button
Static Text
Picture Control
Horizontal Scroll Bar
Vertical Scroll Bar
Slider Control
Spin Control
Progress Control
Hot Key
List Control
Tree Control
Tab Control
Animation Control
Rich Edit 2.0 Control
Date Time Picker
Month Calendar Control
IP Address Control
Extended Combo Box
Custom Control
SysLink Control
Split Button Control
Network Address Control
Command Button Control
Общие
Обозреватель серверов
Панель элементов
Вывод
Показать выходные данные от:
Окно определения кода
Обозреватель вызовов
Вывод
Извлеченные элементы
Готово
7, 7
306 x 153

Создание проекта. Шаг 4



Создание проекта. Шаг 5

- Добавляем на диалоговое окно кнопку, при нажатии на которую будет присходить отрисовка графика

- Двойным щечком по кнопке создаем соответствующий метод

Контекст устройств

Графический ввод-вывод в Windows унифицирован для работы с различными физическими устройствами. Для этого предусмотрен специальный объект, называемый контекстом устройства (Device context). Рисование на некотором абстрактном DC. Если DC связать с окном на экране, то рисование будет в происходить в окне; если связать его с принтером – то на принтере; если с файлом – то, соответственно, в файл.

Класс **CClientDC** – разновидность контекстов устройств; позволяет выводить графику в рабочей области окна.

Для рисования в некоторой функции (н-р, обработчике события нажатия кнопки), нужно получить контекст устройства. Это делается так: CClientDC dc(this);

Отрисовка графика

```
void CGraphDlg::OnBnClickedDraw()
{
  // TODO: добавьте свой код обработчика уведомлений
  //Создаем контекст, в котором будем рисовать
  CClientDC dc(&m_DrawArea);

  //Узнаем размеры прямоугольника
  CRect rc;  //Графический объект
  m_DrawArea.GetClientRect(&rc);
  int w = rc.Width();
  int h = rc.Height();

  int x_start = 10;
  int y_start = h-10;
```

```
//Отрисовка …
	CPen pnPenBlack(PS_SOLID,1,RGB(0,0,0)); //Графический
//объект. Устанавливаем гр. объект в контекст устройства
	CPen * pOldPen = dc.SelectObject(&pnPenBlack);
	dc.FillSolidRect(rc,RGB(255,255,255));
	dc.MoveTo(x_start - 5,y_start);
	dc.LineTo(x_start + w-15, y_start);
	dc.MoveTo(x_start,y_start+5);
	dc.LineTo(x_start, y_start-h+15);
	CPen pnPenRed(PS_SOLID,1,RGB(255,0,0));
	dc.SelectObject(&pnPenRed);
	dc.MoveTo(x_start, y_start);
	for(int i = 3; i < w-x_start-2; i+=3)
	{
		dc.LineTo(x_start + i, y_start - int(h/3*(1 - sin((float)i))));
	}
	dc.SelectObject(pOldPen);
}
```