

Лекция 2: COM технология

(компоненты на основе C++)

Разделение интерфейса и реализации

- библиотеки исходных файлов (или статические библиотеки) – дублирование кода в программах, перекомпиляция клиентских приложений при обновлении реализаций методов класса;

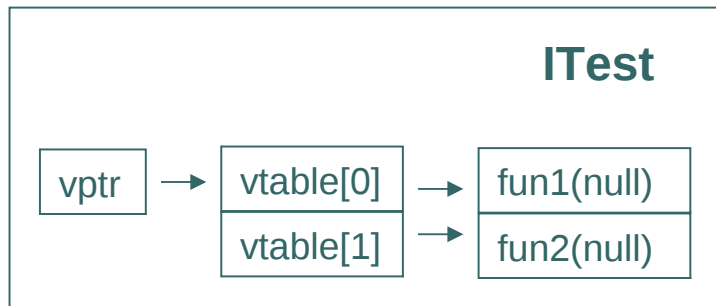
- библиотеки динамической компоновки – проблема *коррекции имен*, (несовместимость компиляторов):

Exports: 5
Function
??4A@@@QAEAAV0@ABV0@@Z
?my_f1@@@QAEHHH@Z
?my_f1@@@QAEXH@Z
?my_f1@@@QAEXXZ
?my_g1@@@AAEXXZ

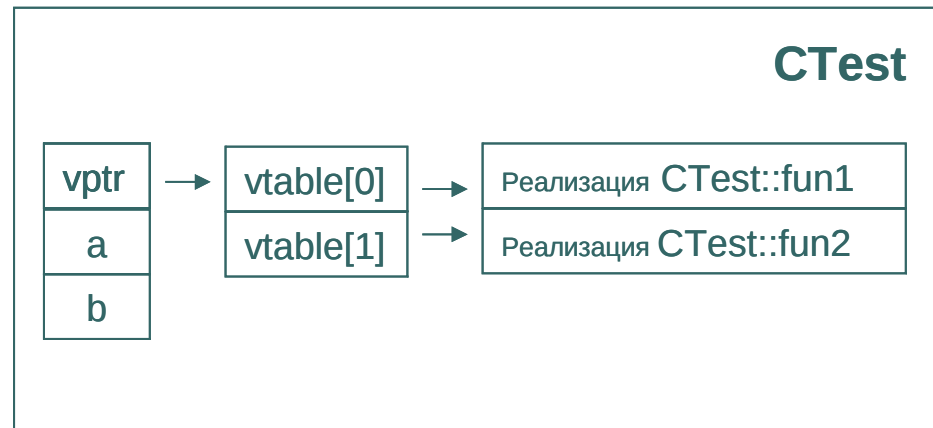
```
class __declspec(dllexport) A{
public:
    void my_f1(int);
    int my_f1(int,int);
    void my_f1();
private:
    void my_g1();
};
```

Разделение интерфейса и реализации (механизм вызова виртуальных функций)

```
class ITest{  
public:  
    virtual int fun1()=0;  
    virtual double fun2(int)=0;  
};
```



```
class CTest : public ITest{  
public:  
    double a,b;  
    int fun1(){ return 0; }  
    double fun2(int d){ return (double)d; }  
};
```



Глоссарий: позднее связывание, полиморфизм времени исполнения, виртуальная таблица (vtable), виртуальный указатель (vptr).



Разделение интерфейса и реализации (Pattern #1)

IFunction.h

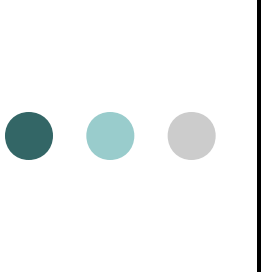
```
class IFunction{  
public:  
    virtual double getValue()=0;  
    virtual void setValue(double)=0;  
};
```

CFunction.h

```
#include "IFunction.h"  
  
class CFunction : public IFunction{  
public:  
    double getValue();  
    void setValue(double);  
private:  
    double value;  
};
```

CFunction.cpp

```
#include "CFunction.h"  
double CFunction::getValue(){  
    return value;  
}  
void CFunction::setValue(double v){  
    value=v;  
}  
extern "C" __declspec(dllexport) IFunction* CreateObject(IFunction* pif){  
    return new CFunction;  
}  
extern "C" __declspec(dllexport) void DeleteObject(IFunction* pif){  
    delete pif;  
}
```



Разделение интерфейса и реализации (Pattern #1, продолжение)

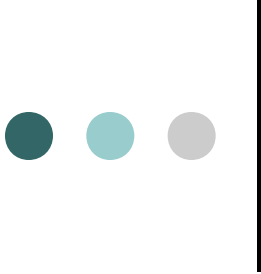
main3.cpp

```
#include <cstdio>
#include "IFunction.h"
extern "C" __declspec(dllimport) IFunction* CreateObject();
extern "C" __declspec(dllimport) void DeleteObject(IFunction*);

int main(){
    IFunction* pif=CreateObject();

    pif->setValue(3.145192);
    printf("%g\n", pif->getValue());

    DeleteObject(pif);
    return 0;
}
```



Разделение интерфейса и реализации

(Pattern #1, окончание)

main4.cpp

```
#include <cstdio>
#include "IFunction.h"

int main(){
    IFunction* pif;
    HINSTANCE hInst;
    IFunction* (*create_object)();
    void (*delete_object)(IFunction*);

    hInst=LoadLibrary("CFunction.dll");
    create_object=(IFunction* (*)())GetProcAddress(hInst,"CreateObject");
    delete_object=(void (*)(IFunction*))GetProcAddress(hInst,"DeleteObject");

    pif=create_object();

    pif->setValue(28.70);
    printf("%g\n", pif->getValue());

    delete_object(pif);
    FreeLibrary(hInst);
    return 0;
}
```



Разделение интерфейса и реализации (модификация компонентов, Pattern #2)

IFunction.h

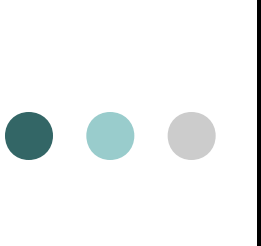
```
class IFunction{  
public:  
    virtual double getValue()=0;  
    virtual void setValue(double)=0;  
};
```

ISerialization.h

```
class ISerialization{  
public:  
    virtual void Delete()=0;  
    virtual void Load()=0;  
    virtual void Store()=0;  
};
```

CFunction.h

```
#include "IFunction.h"  
#include "ISerialization.h"  
  
class CFunction : public IFunction, public ISerialization{  
public:  
    double getValue();  
    void setValue(double);  
    void Delete();  
    void Load();  
    void Store();  
private:  
    double value;  
};
```

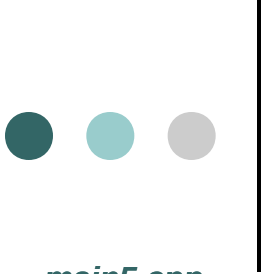


Разделение интерфейса и реализации

(модификация компонентов, Pattern #2, продолжение)

CFunction.cpp

```
#include "CFunction.h"
#include <iostream>
using namespace std;
double CFunction::getValue(){
    return value;
}
void CFunction::setValue(double v){
    value=v;
}
void CFunction::Delete(){
    delete this;
}
void CFunction::Store(){
    cout<<"We stored: "<<value<<endl;
}
void CFunction::Load(){
    cin>>value;
}
extern "C" __declspec(dllexport) IFunction* CreateObject(IFunction* pif){
    return new CFunction;
}
extern "C" __declspec(dllexport) void DeleteObject(IFunction* pif){
    delete pif;
}
```



Разделение интерфейса и реализации (модификация компонентов, Pattern #2)

main5.cpp

```
#include <cstdio>
#include "IFunction.h"
#include "ISerialization.h"
extern "C" __declspec(dllimport) IFunction* CreateObject();

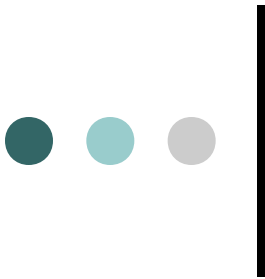
int main(){
    IFunction* pif=CreateObject();
    ISerialization* pis=dynamic_cast<ISerialization*>(pif);

    pif->setValue(3.145192);

    pis->Store();
    pis->Load();

    printf("Get %g\n", pif->getValue());

    pis->Delete();
    return 0;
}
```

Разделение интерфейса и реализации

(динамическое преобразование типа на стороне компонента, Pattern #3)

intfc.h

```
class IBase{
public:
    virtual void Delete()=0;
    virtual void* DynamicCast(char*)=0;
};

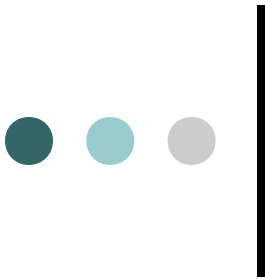
class IFunction : public IBase{
public:
    virtual double getValue()=0;
    virtual void setValue(double)=0;
};

class ISerialization : public IBase{
public:
    virtual void Load()=0;
    virtual void Store()=0;
};
```

CFunction.h

```
#include "intfc.h"

class CFunction : public IFunction, public ISerialization{
public:
    double getValue();
    void setValue(double);
    void Delete();
    void Load();
    void Store();
    void* DynamicCast(char*);
private:
    double value;
};
```



Разделение интерфейса и реализации

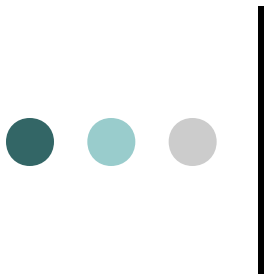
(динамическое преобразование типа на стороне компонента, Pattern #3, продолжение)

CFunction.cpp

```
#include "CFunction.h"
#include <iostream>
using namespace std;
double CFunction::getValue(){
    return value;
}
void CFunction::setValue(double v){
    value=v;
}
void CFunction::Delete(){
    delete this;
}
void CFunction::Store(){
    cout<<"We stored: "<<value<<endl;
}
void CFunction::Load(){
    cin>>value;
}
//p.1
```

```
void* CFunction::DynamicCast(char* intf_type){
    if (strcmp(intfc_type, "IFunction") == 0)
        return static_cast<IFunction*>(this);
    else if (strcmp(intfc_type, "ISerialization") == 0)
        return static_cast<ISerialization*>(this);
    else
        return 0;
}

extern "C" __declspec(dllexport) IFunction*
CreateObject(IFunction* pif){
    return new CFunction;
}
//p.2
```



Разделение интерфейса и реализации

(динамическое преобразование типа на стороне компонента, Pattern #3, окончание)

main6.cpp

```
#include <cstdio>
#include "intfc.h"
extern "C" __declspec(dllimport) IFunction* CreateObject();

int main(){
    IFunction* pif=CreateObject();
    ISerialization* pis=(ISerialization*)pif->DynamicCast("ISerialization");

    pif->setValue(3.145192);

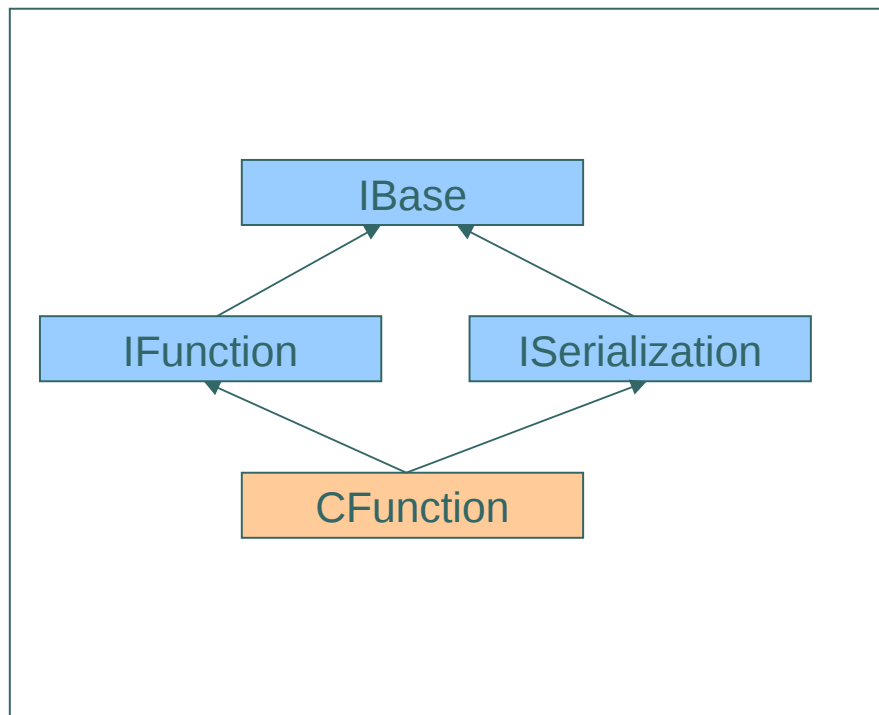
    pis->Store();
    pis->Load();

    printf("Get %g\n", pif->getValue());

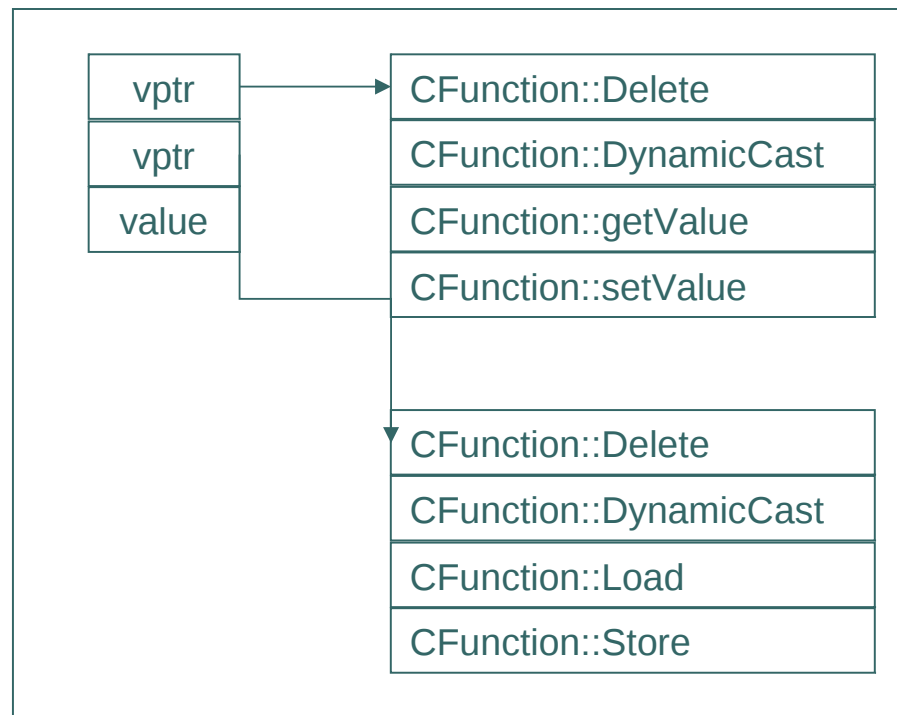
    pis->Delete();
    // printf("Get %g\n", pif->getValue()); обращение к «убитому» указателю
    return 0;
}
```

Иерархия типов и двоичное представление компонента CFunction

Иерархия типов



Двоичное представление





Освобождение памяти

```
class IBase{  
public:  
    virtual void* DynamicCast(char*)=0;  
    virtual void  AddRef()=0;  
    virtual void  FreePointer()=0;  
};
```

```
#include "intfc.h"  
  
class CFunction : public IFunction, public ISerialization{  
public:  
    CFunction():ref_counter(0){}  
    double getValue();  
    void setValue(double);  
    void Load();  
    void Store();  
    void* DynamicCast(char*);  
    void AddRef();  
    void FreePointer();  
private:  
    double value;  
    int ref_counter;  
};
```

Освобождение памяти (продолжение)

```
void* CFunction::DynamicCast(char* intfct_type){
    void* pib=0;
    if (strcmp(intfct_type, "IFunction") == 0)
        pib=static_cast<IFunction*>(this);
    else if (strcmp(intfct_type, "ISerialization") == 0)
        pib=static_cast<ISerialization*>(this);
    else
        return 0;
    ((IBase*)pib)->AddRef();
    return pib;
}
void CFunction::AddRef(){
    ref_counter++;
}
void CFunction::FreePointer(){
    if(--ref_counter==0)
        delete this;
}
extern "C" __declspec(dllexport) IFunction* CreateObject(void){
    IFunction* pif=new CFunction;
    if(pif)
        pif->AddRef();
    return pif;
}
```



ЗАКЛЮЧЕНИЕ

Представленные образцы решают задачу создания двоичных компонентов (ООП на двоичном уровне) средствами языка C++.

Полноценная COM – модель включает (см. *Лекцию 4*):

- поддержку компонент различными языки программирования;
- разрешение конфликта имен интерфейсов;
- использование компонент вне процесса пользователя (out-of-process).