// LISTING # 1 – PIERWSZE PODEJŚCIE

```
#include <iostream.h>
#include <objbase.h>

// interfejs A
interface IA
{
     virtual void __stdcall fa() = 0;
};

// interfejs B
interface IB
{
     virtual void __stdcall fb() = 0;
};

// Składnik
class CMyComp : public IA, public IB
{
     // Implementacja interfejsu A
     virtual void __stdcall fa();
     // Implementacja interfejsu B
     virtual void __stdcall fb();
};

void CMyComp::fa()
{
     cout << "Jestem fa" << endl;
}

void CMyComp::fb()
{
     cout << "Jestem fb" << endl;
}

void main()
{
     CMyComp *pComp = new CMyComp;
     // wskaźnik do interfejsu A
     IA *pIA = pComp;
     pIA->fa();
     // wskaźnik do interfejsu B
     IB *pIB = pComp;
     pIB->fb();
     delete pComp;
}
```

// LISTING #2 – DRUGIE PODEJŚCIE

```
//////////////////////////////
// Plik nagłówkowy interface.h
#include <objbase.h>

// interfejs A
interface IA : public IUnknown
{    virtual void __stdcall fa() = 0;
};

// interfejs B
interface IB : public IUnknown
{    virtual void __stdcall fb() = 0;
};

// deklaracje identyfikatorów IID
extern const IID IID_IA;
extern const IID IID_IB;

// deklaracja funkcji tworzącej egzemplarz składnika
IUnknown *CreateInstance();


//////////////////////////////
// Definicja identyfikatorów IID

// {033E7EC7-50FA-404c-9C59-4CBA8892F054}
static const GUID IID_IA =
     { 0x33e7ec7, 0x50fa, 0x404c, { 0x9c, 0x59, 0x4c, 0xba, 0x88,
       0x92, 0xf0, 0x54 } };
// {FD854FCC-6AE2-4876-BB60-DC276E5DFC68}
static const GUID IID_IB =
     { 0xfd854fcc, 0x6ae2, 0x4876, { 0xbb, 0x60, 0xdc, 0x27,
       0x6e, 0x5d, 0xfc, 0x68 } };
```

```
//////////////////////////////
// Aplikacja kliencka

#include "interface.h"
void main()
{    // inicjalizacja składnika
     IUnknown *pUnknown = CreateInstance();

     // chcemy skorzystać z interfejsu IA
     IA *pIA = NULL;
     HRESULT hr = pUnknown->QueryInterface(IID_IA, (void**)&pIA);
     if (SUCCEEDED(hr))
     {    pIA->fa();
          pIA->Release();
     }

     // chcemy skorzystać z interfejsu IB
     IB *pIB = NULL;
     hr = pUnknown->QueryInterface(IID_IB, (void**)&pIB);
     if (SUCCEEDED(hr))
     {    pIB->fb();
          pIB->Release();
     }
     pUnknown->Release();       // delete pUnknown;
}

//////////////////////////////
// Moduł składnika
#include "interface.h"
#include <iostream.h>

// Klasa składnika
class CMyComp : public IA, public IB
{    // Implementacja interfejsu IUnknown
     virtual HRESULT _stdcall QueryInterface(REFIID iid, void **ppv);
     virtual ULONG _stdcall AddRef();
     virtual ULONG _stdcall Release();

     // Implementacja interfejsu A
     virtual void __stdcall fa() {cout << "Jestem fa" << endl; }

     // Implementacja interfejsu B
     virtual void __stdcall fb() {cout << "Jestem fb" << endl; }
public:
     CMyComp() : m_nRef(0) { }
private:
     long m_nRef;
};

IUnknown *CreateInstance()
{
     IUnknown *p = (IA*)new CMyComp;
     p->AddRef();
     return p;
}

HRESULT _stdcall CMyComp::QueryInterface(REFIID iid, void **ppv)
{
     if (iid == IID_IUnknown)
          *ppv = (IA*)this;
     else if (iid == IID_IA)
          *ppv = (IA*)this;
     else if (iid == IID_IB)
          *ppv = (IB*)this;
     else
     {    ppv = NULL;
          return E_NOINTERFACE;
     }
     ((IUnknown*)(*ppv))->AddRef();
     return S_OK;
}

ULONG _stdcall CMyComp::AddRef()
{
     return InterlockedIncrement(&m_nRef);
}

ULONG _stdcall CMyComp::Release()
{
     if (InterlockedDecrement(&m_nRef) == 0)
     {
         delete this;
          return 0;
     }
     return m_nRef;
}
```

// LISTING #3 – TRZECIE PODEJŚCIE

```
//////////////////////////////
// Plik nagłówkowy interface.h

#include <objbase.h>

// interfejs A
interface IA : public IUnknown
{
     virtual void __stdcall fa() = 0;
};

// interfejs B
interface IB : public IUnknown
{
     virtual void __stdcall fb() = 0;
};

// deklaracje identyfikatorów IID
extern const IID IID_IA;
extern const IID IID_IB;


//////////////////////////////
// Definicja identyfikatorów IID

// {59F1DE0D-8843-4a57-82CE-D6DBA3D28D28}
static const GUID CLSID_Component =
     { 0x59f1de0d, 0x8843, 0x4a57, { 0x82, 0xce, 0xd6, 0xdb,
       0xa3, 0xd2, 0x8d, 0x28 } };

// {033E7EC7-50FA-404c-9C59-4CBA8892F054}
static const GUID IID_IA =
     { 0x33e7ec7, 0x50fa, 0x404c, { 0x9c, 0x59, 0x4c, 0xba, 0x88,
       0x92, 0xf0, 0x54 } };

// {FD854FCC-6AE2-4876-BB60-DC276E5DFC68}
static const GUID IID_IB =
     { 0xfd854fcc, 0x6ae2, 0x4876, { 0xbb, 0x60, 0xdc, 0x27,
       0x6e, 0x5d, 0xfc, 0x68 } };
```

```
//////////////////////////////
// Aplikacja kliencka - korzystająca z serwera COM

#include "interface.h"
void main()
{
     // inicjalizacja podsystemu COM
     CoInitialize(NULL);

     // inicjalizacja składnika
     IUnknown *pUnknown = NULL;
     HRESULT hr = CoCreateInstance(CLSID_Component, NULL,
          CLSCTX_INPROC_SERVER, IID_IUnknown, (void**)&pUnknown);

     if (SUCCEEDED(hr))
     {    // chcemy skorzystać z interfejsu IA
          IA *pIA = NULL;
          hr = pUnknown->QueryInterface(IID_IA, (void**)&pIA);
          if (SUCCEEDED(hr))
          {
               pIA->fa();
               pIA->Release();
          }

          // dalej jak w listingu #2
          pUnknown->Release();
     }
}

//////////////////////////////
// Moduł składnika (serwer)

#include "interface.h"
#include <iostream.h>
#include "registry.h"      // narzędzia do rejestru systemowego...

////////////////////////////////////////////////////////
// Zmienne globalne

static HMODULE g_hModule = NULL;  // uchwyt modułu DLL
static long g_cComponents = 0 ;  // Licznik aktywnych składników
static long g_cServerLocks = 0 ; // Licznik dla LockServer

// Dane do rejestru systemowego
const char g_szProgID[] = "COMLecture.Approach3.1";
const char g_szFriendlyName[] = "Wykład z COM, podejście 3";
const char g_szVerIndProgID[] = "COMLecture.Approach3";
```

```
//////////////////////////////////////////////////////////
// Klasa składnika

class CMyComp : public IA, public IB
{
public:
      // Implementacja interfejsu IUnknown
      virtual HRESULT _stdcall QueryInterface(REFIID iid, void **ppv);
      virtual ULONG _stdcall AddRef();
      virtual ULONG _stdcall Release();

      // Implementacja interfejsu A
      virtual void __stdcall fa() {cout << "Jestem fa" << endl; }

      // Implementacja interfejsu B
      virtual void __stdcall fb() {cout << "Jestem fb" << endl; }
public:
      CMyComp() : m_nRef(1) { InterlockedIncrement(&g_cComponents); }
      ~CMyComp()            { InterlockedDecrement(&g_cComponents); }
private:
      long m_nRef;
};

HRESULT _stdcall CMyComp::QueryInterface(REFIID iid, void **ppv)
{     if (iid == IID_IUnknown)
            *ppv = (IA*)this;
      else if (iid == IID_IA)
            *ppv = (IA*)this;
      else if (iid == IID_IB)
            *ppv = (IB*)this;
      else
      {     ppv = NULL;
            return E_NOINTERFACE;
      }
      ((IUnknown*)(*ppv))->AddRef();
      return S_OK;
}

ULONG _stdcall CMyComp::AddRef()
{     return InterlockedIncrement(&m_nRef);
}
ULONG _stdcall CMyComp::Release()
{     if (InterlockedDecrement(&m_nRef) == 0)
      {     delete this;
            return 0;
      }
      return m_nRef;
}
```

```
//////////////////////////////////////////////////////////
// Class Factory

class CFactory : public IClassFactory
{
public:
      // Implementacja interfejsu IUnknown
      virtual HRESULT _stdcall QueryInterface(REFIID iid, void **ppv);
      virtual ULONG _stdcall AddRef();
      virtual ULONG _stdcall Release();

      // Implementacja interfejsu IClassFactory
      virtual HRESULT __stdcall CreateInstance(
                  IUnknown* pUnknownOuter, const IID& iid, void** ppv);
      virtual HRESULT __stdcall LockServer(BOOL bLock);

      CFactory() : m_nRef(1) {}
private:
      long m_nRef ;
};

//////////////////////////////////////////////////////////
// Class Factory – interfejs IUnknown

HRESULT __stdcall CFactory::QueryInterface(const IID& iid, void** ppv)
{
      if ((iid == IID_IUnknown) || (iid == IID_IClassFactory))
            *ppv = (IClassFactory*)this;
      else
      {     ppv = NULL;
            return E_NOINTERFACE;
      }
      ((IUnknown*)(*ppv))->AddRef();
      return S_OK;
}

ULONG __stdcall CFactory::AddRef()
{     return InterlockedIncrement(&m_nRef) ;
}

ULONG __stdcall CFactory::Release()
{     if (InterlockedDecrement(&m_nRef) == 0)
      {
            delete this ;
            return 0 ;
      }
      return m_nRef ;
}
```

```
////////////////////////////////////////////////////////
// Class Factory – interfejs IClassFactory

HRESULT __stdcall CFactory::CreateInstance(IUnknown* pUnknownOuter,
                                        const IID& iid, void** ppv)
{
     if (pUnknownOuter != NULL)
          return CLASS_E_NOAGGREGATION;

     // Utworzenie składnika
     CMyComp* pMyComp = new CMyComp;
     if (!pMyComp)
          return E_OUTOFMEMORY;

     // Utworzenie żądanego interfejsu
     HRESULT hr = pMyComp->QueryInterface(iid, ppv);

     pMyComp->Release();
     return hr;
}

HRESULT __stdcall CFactory::LockServer(BOOL bLock)
{
     if (bLock)
          InterlockedIncrement(&g_cServerLocks) ;
     else
       InterlockedDecrement(&g_cServerLocks) ;
     return S_OK ;
}

////////////////////////////////////////////////////////
//
// Inicjalizacja DLL

BOOL APIENTRY DllMain(HMODULE hModule, DWORD ul_reason_for_call,
LPVOID lpReserved)
{
     if (ul_reason_for_call == DLL_PROCESS_ATTACH)
          g_hModule = hModule;
     return TRUE ;
}
```

```
////////////////////////////////////////////////////////
// Funkcje eksportowane!

STDAPI DllCanUnloadNow()
{    if ((g_cComponents == 0) && (g_cServerLocks == 0))
          return S_OK;
     else
          return S_FALSE;
}

STDAPI DllGetClassObject(const CLSID& clsid, const IID& iid, void** ppv)
{
     // Czy możemy utworzyć taki składnik?
     if (clsid != CLSID_Component)
          return CLASS_E_CLASSNOTAVAILABLE ;

     // Utwórz fabrykę klas
     CFactory* pFactory = new CFactory;

     if (pFactory == NULL)
          return E_OUTOFMEMORY;

     // Zwróć żądany interfejs
     HRESULT hr = pFactory->QueryInterface(iid, ppv);
     pFactory->Release();
     return hr;
}

STDAPI DllRegisterServer()
{    return RegisterServer(g_hModule, CLSID_Component,
                    g_szFriendlyName, g_szVerIndProgID, g_szProgID);
}
STDAPI DllUnregisterServer()
{    return UnregisterServer(CLSID_Component, g_szVerIndProgID,
                                                        g_szProgID);
}

//////////////////////////////
// Plik COMPONENT.DEF

LIBRARY         Component.dll
DESCRIPTION     'Przykład do wykładu COM, (C) Jarosław Francik 2001'

EXPORTS
                DllGetClassObject   @2   PRIVATE
                DllCanUnloadNow     @3   PRIVATE
                DllRegisterServer   @4   PRIVATE
                DllUnregisterServer @5   PRIVATE
```

// LISTING #4 – CZWARTE PODEJŚCIE (SERWER EXE)

```
////////////////////////////////
// Specyfikacja mytypes.IDL

import "unknwn.idl";

// Interface IA
[    object,
     uuid(B5DB3493-9925-4633-B7DB-30DA5E75D347),
     helpstring("IA Interface"),
     pointer_default(unique)
]
interface IA : IUnknown
{    HRESULT fa();
};

// Interface IB
[    object,
     uuid(10AE43A2-2D3F-48a9-8ED3-D3F1F21CC898),
     helpstring("IB Interface"),
     pointer_default(unique)
]
interface IB : IUnknown
{    HRESULT fb();
};

// Biblioteka typów + clsid składników
[    uuid(3221CCFF-68B2-442b-AFDC-FA9C767F1FC4),
     version(1.0),
     helpstring("Biblioteka typów Approach3a")
]
library MyTypeLib
{    importlib("stdole32.tlb") ;
     [
          uuid(E843265D-B374-4d56-A980-0EC5E9188B47),
          helpstring("Klasa komponentu Approach3a")
     ]
     coclass Component
     {
          [default] interface IA;
          interface IB;
     };
} ;
```

```
////////////////////////////////
// Aplikacja kliencka - korzystająca z serwera COM

#include "..\\mytypes.h"
#include "..\\mytypes_i.c"
void main()
{
     // inicjalizacja podsystemu COM
     CoInitialize(NULL);

     // inicjalizacja składnika
     IUnknown *pUnknown = NULL;
     HRESULT hr = CoCreateInstance(CLSID_Component, NULL,
            CLSCTX_LOCAL_SERVER, IID_IUnknown, (void**)&pUnknown);

////////////////////////////////
// Moduł proxy

// zawiera wyłącznie pliki utworzone przez midl:
// dlldata.c, abc_i.c, abc_p.c,
// linkowane z rpcndr.lib, rpcns4.lib i rpcrt4.lib
// Oto plik PROXY.DEF:

LIBRARY         proxy.dll
DESCRIPTION     'Przykład do wykładu COM, (C) Jarosław Francik 2001'

EXPORTS
                DllGetClassObject   @1   PRIVATE
                DllCanUnloadNow     @2   PRIVATE
                GetProxyDllInfo     @2   PRIVATE
                DllRegisterServer   @4   PRIVATE
                DllUnregisterServer @5   PRIVATE

////////////////////////////////
// Moduł składnika (serwer)

#include "..\\mytypes.h"
#include "..\\mytypes_i.c"

#include <iostream.h>
#include <memory.h>
#include "registry.h"      // narzędzia do rejestru systemowego...

////////////////////////////////////////////////////////
// Zmienne globalne

// usunieto zmienną g_hModule
// ponadto jak w podejściu Trzecim
```

```
//////////////////////////////////////////////////////
//
// Klasa składnika

class CMyComp : public IA, public IB
{
public:
     // Implementacja interfejsu IUnknown jak w Podejściu Trzecim

     // Implementacja interfejsu A
     virtual HRESULT __stdcall fa()
        { MessageBox(0, "Jestem fa", "Jestem fa", 0); return S_OK; }

     // Implementacja interfejsu B
     virtual HRESULT __stdcall fb()
        { MessageBox(0, "Jestem fb", "Jestem fb", 0); return S_OK; }

public:
     CMyComp() : m_nRef(1) { InterlockedIncrement(&g_cComponents); }
     ~CMyComp()      { InterlockedDecrement(&g_cComponents);
                       if (g_cComponents == 0 && g_cServerLocks == 0)
                             PostQuitMessage(0); }
private:
     long m_nRef;
};

// Implementacja klasy CMyComp – jak w Podejściu Trzecim

//////////////////////////////////////////////////////
//
// Class Factory

// Deklaracja i implementacja fabryki klas jak w Podejściu Trzecim;
// jedyna różnica dotyczy poniższej funkcji:

HRESULT __stdcall CFactory::LockServer(BOOL bLock)
{
     if (bLock)
          InterlockedIncrement(&g_cServerLocks);
     else
          InterlockedDecrement(&g_cServerLocks);

     if (g_cComponents == 0 && g_cServerLocks == 0)
          PostQuitMessage(0);

     return S_OK ;
}
```

```
//////////////////////////////////////////////////////
//
// Część aplikacyjna + obsługa COM

LRESULT CALLBACK WndProc(HWND, UINT, WPARAM, LPARAM);

int APIENTRY WinMain(HINSTANCE hInstance,
                     HINSTANCE hPrevInstance,
                     LPSTR     lpCmdLine,
                     int       nCmdShow)
{    ////////////////////////////////////////////////////////
     // Inicjalizacja COM

     // flaga określająca tryb pracy
     bool bEmbedding = false;

     // Inicjalizacja COM
     HRESULT hr = CoInitialize(NULL) ;
     if (FAILED(hr))
          return 0 ;

     // Odczyt linii wywołania
     char szTokens[] = "-/";
     char* szToken = strtok(lpCmdLine, szTokens) ;
     while (szToken != NULL)
     {
          if (_stricmp(szToken, "UnregServer") == 0)
          {
               UnregisterServer(FALSE, CLSID_Component,
                                     g_szVerIndProgID, g_szProgID);
               CoUninitialize();
               return 0;
          }
          else if (_stricmp(szToken, "RegServer") == 0)
          {
               RegisterServer(FALSE, hInstance, CLSID_Component,
                   g_szFriendlyName, g_szVerIndProgID, g_szProgID);
               CoUninitialize();
               return 0;
          }
          else if (_stricmp(szToken, "Embedding") == 0)
          {
               bEmbedding = true;
               break ;
          }
          szToken = strtok(NULL, szTokens);
     }
```

```
// Rejestracja fabryki klas
DWORD dRegister;
CFactory *pFactory = new CFactory;
hr = ::CoRegisterClassObject(CLSID_Component, pFactory,
          CLSCTX_LOCAL_SERVER, REGCLS_MULTIPLEUSE, &dRegister);
if (FAILED(hr))
{
      pFactory->Release() ;
      return FALSE ;
}

////////////////////////////////////////////////////////////
// Standardowa część aplikacji

if (!bEmbedding)
{
      // Rejestracja klasy okna
      WNDCLASS wc;
      memset(&wc, 0, sizeof(wc));
      wc.lpfnWndProc = (WNDPROC)WndProc;
      wc.hInstance = hInstance;
      wc.hCursor = LoadCursor(NULL, IDC_ARROW);
      wc.hbrBackground = (HBRUSH)GetStockObject(WHITE_BRUSH);
      wc.lpszClassName = "my_class1";
      RegisterClass(&wc);

      // Utworzenie okna
      HWND hWnd;
      hWnd = CreateWindow("my_class1", "Tytuł",
               WS_OVERLAPPEDWINDOW, CW_USEDEFAULT, CW_USEDEFAULT,
                                    CW_USEDEFAULT, CW_USEDEFAULT,
                                    NULL, NULL, hInstance, NULL);
      if (!hWnd)
           return FALSE;
      ShowWindow(hWnd, nCmdShow);
      UpdateWindow(hWnd);

      // Blokuje serwer na czas używania interfejsu użytkownika
      InterlockedIncrement(&g_cServerLocks);
}
// Pętla komunikatów
MSG msg;
while (GetMessage(&msg, NULL, 0, 0))
{
      TranslateMessage(&msg);
      DispatchMessage(&msg);
}

      ////////////////////////////////////////////////////////////
      // Końcowa deinicjalizacja systemu COM!

      CoRevokeClassObject(dRegister);
      CoUninitialize();

      return msg.wParam;
}

////////////////////////////////////////
// Funkcja okienkowa

LRESULT CALLBACK WndProc(HWND hWnd, UINT message,
                         WPARAM wParam, LPARAM lParam)
{
      switch (message)
      {
      case WM_DESTROY:
            // nie pozwala zamknąć aplikacji,
            // dopóki klienci korzystają z serwera
            if (g_cComponents == 0 && g_cServerLocks == 0)
                 PostQuitMessage(0);
            break ;

      case WM_CLOSE:
            // znosi blokadę na czas używania interfejsu użytkownika
            ::InterlockedDecrement(&g_cServerLocks);
            return (DefWindowProc(hWnd, message, wParam, lParam)) ;

      default:
            return DefWindowProc(hWnd, message, wParam, lParam);
   }
   return 0;
}
```

// LISTING #5 – PIĄTE PODEJŚCIE (DISPINTERFEJSY)

```
////////////////////////////////
// Specyfikacja mytypes.IDL

import "unknwn.idl";

// Interface IA
[    object,
     uuid(B5DB3493-9925-4633-B7DB-30DA5E75D347),
     helpstring("IA Interface"),
     pointer_default(unique),
     dual,
     oleautomation
]
interface IA : IDispatch
{    HRESULT fa();
};

// reszta bez zmian

////////////////////////////////
// Aplikacja kliencka - korzystająca z interfejsu vtable
// BEZ ŻADNYCH ZMIAN

////////////////////////////////
// Moduł proxy – bez zmian
```

```
////////////////////////////////
// Moduł składnika (serwer)

// rozszerzenie klasy CMyComp
// obejmuje implementację interfejsu IDispatch

class CMyComp : public IA, public IB
{
public:
     // Implementacja interfejsu IDispatch
    virtual HRESULT __stdcall GetTypeInfoCount(
                                          UINT __RPC_FAR *pctinfo);

    virtual HRESULT __stdcall GetTypeInfo(
        UINT iTInfo, LCID lcid,
        ITypeInfo __RPC_FAR *__RPC_FAR *ppTInfo);

    virtual HRESULT __stdcall GetIDsOfNames(
        REFIID riid,
        LPOLESTR __RPC_FAR *rgszNames,
        UINT cNames,
        LCID lcid,
        DISPID __RPC_FAR *rgDispId);

    virtual HRESULT __stdcall Invoke(
        DISPID dispIdMember,
        REFIID riid,
        LCID lcid,
        WORD wFlags,
        DISPPARAMS __RPC_FAR *pDispParams,
        VARIANT __RPC_FAR *pVarResult,
        EXCEPINFO __RPC_FAR *pExcepInfo,
        UINT __RPC_FAR *puArgErr);

protected:
     ITypeInfo *m_pITypeInfo;     // rzeczy pomocne
     HRESULT InitTypeInfo();
public:

     // uzupełnienie konstruktora:
     CMyComp() : m_nRef(1)
                { InterlockedIncrement(&g_cComponents);
                  InitTypeInfo(); }

     // RESZTA BEZ ZMIAN
};
```

```
// Implementacja interfejsu IDispatch opiera się na
// wykorzystaniu funkcjonalności obiektu informacji o typach

HRESULT __stdcall CMyComp::GetTypeInfoCount(UINT __RPC_FAR *pctinfo)
{    *pctinfo = 1;
     return S_OK;
}

HRESULT __stdcall CMyComp::GetTypeInfo(UINT iTInfo, LCID lcid,
    ITypeInfo __RPC_FAR *__RPC_FAR *ppTInfo)
{
     *ppTInfo = NULL;
     if (iTInfo != 0)
          return DISP_E_BADINDEX;
     m_pITypeInfo->AddRef();
     *ppTInfo = m_pITypeInfo;
     return S_OK;
}

HRESULT __stdcall CMyComp::GetIDsOfNames(REFIID riid,
    LPOLESTR __RPC_FAR *rgszNames,
    UINT cNames,
    LCID lcid,
    DISPID __RPC_FAR *rgDispId)
{
     if (riid != IID_NULL)
          return DISP_E_UNKNOWNINTERFACE;
     HRESULT hr;
     hr = m_pITypeInfo->GetIDsOfNames(rgszNames, cNames, rgDispId);
     return hr;
}

HRESULT __stdcall CMyComp::Invoke(
    DISPID dispIdMember,
    REFIID riid,
    LCID lcid,
    WORD wFlags,
    DISPPARAMS __RPC_FAR *pDispParams,
    VARIANT __RPC_FAR *pVarResult,
    EXCEPINFO __RPC_FAR *pExcepInfo,
    UINT __RPC_FAR *puArgErr)
{
     if (riid != IID_NULL)
          return DISP_E_UNKNOWNINTERFACE;
     HRESULT hr = m_pITypeInfo->Invoke((IA*)this, dispIdMember,
             wFlags, pDispParams, pVarResult, pExcepInfo, puArgErr);
     return hr;
}
```

```
// Funkcja tworząca obiekt informacji o typach

HRESULT CMyComp::InitTypeInfo()
{
     HRESULT hr;

     // spróbuj załadować bibliotekę typów z rejestru systemowego
     ITypeLib *pTypeLib = NULL;
     hr = LoadRegTypeLib(LIBID_MyTypeLib, 1, 0, 0x00, &pTypeLib);
     if (FAILED(hr))
     {
         // załaduj z pliku

         // ustal nazwę pliku exe i podziel ją na czynniki pierwsze
         char szModule[512] ;
         DWORD dwResult =
                   ::GetModuleFileName(g_hInstance, szModule, 512);
         char szDrive[_MAX_DRIVE];
         char szDir[_MAX_DIR];
         _splitpath(szModule, szDrive, szDir, NULL, NULL);
         // Określ nazwę pliku TLB
         char szTypeLibFullName[_MAX_PATH];
         sprintf(szTypeLibFullName, "%s%s%s", szDrive, szDir,
                                                    "mytypes.tlb");
         // convert to wide char
         wchar_t wszTypeLibFullName[_MAX_PATH];
         mbstowcs(wszTypeLibFullName, szTypeLibFullName, _MAX_PATH);

         hr = LoadTypeLib(wszTypeLibFullName, &pTypeLib);
         if (FAILED(hr))
              return hr;
         // na wszelki wypadek zarejestruj
         hr = RegisterTypeLib(pTypeLib, wszTypeLibFullName, NULL);
         if (FAILED(hr))
              return hr;
     }
     // pobierz TypeInfo dla IA
     m_pITypeInfo = NULL;
     hr = pTypeLib->GetTypeInfoOfGuid(IID_IA, &m_pITypeInfo);
     pTypeLib->Release();
     if (FAILED(hr))
          return hr;
     return S_OK;
}

// Podnadto uzupełniono CMyComp::QueryInterface tak, by zwracała
// w razie potrzeby wskaźnik na IDispatch
```

```
//////////////////////////////////////
// Aternatywna Aplikacja kliencka
// Aplikacja korzysta z dispinterfejsu

#include <objbase.h>
#include <assert.h>

void main()
{
     // inicjalizacja podsystemu COM
     CoInitialize(NULL);

     wchar_t progid[] = L"COMLecture.Approach5";
     CLSID clsid;
     CLSIDFromProgID(progid, &clsid);

     IDispatch *pIDispatch = NULL;
     HRESULT hr = CoCreateInstance(clsid, NULL, CLSCTX_LOCAL_SERVER,
                              IID_IDispatch, (void**)&pIDispatch);
     assert(SUCCEEDED(hr));

     DISPID dispid;
     OLECHAR *name = L"fa";
     hr = pIDispatch->GetIDsOfNames(IID_NULL, &name, 1,
                                   GetUserDefaultLCID(), &dispid);
     assert(SUCCEEDED(hr));

     VARIANTARG varg;
     VariantInit(&varg);
     varg.vt = VT_EMPTY;

     DISPPARAMS param;
     param.cArgs = 0;
     param.rgvarg = &varg;
     param.cNamedArgs = 0;
     param.cArgs = NULL;
     hr = pIDispatch->Invoke(dispid, IID_NULL, GetUserDefaultLCID(),
                          DISPATCH_METHOD, &param, NULL, NULL, NULL);
     assert(SUCCEEDED(hr));
}
```

```
//////////////////////////////////////
// Klient w Visual Basicu:

     Dim x As Object
     Set x = CreateObject(COMLecture.Approach5”)
     x.fa


//////////////////////////////////////////
// Klient w MFC (po utworzeniu klasy “wrapper”

     IA ia;
     ia.CreateDispatch("COMLecture.Approach5");
     ia.fa();
```