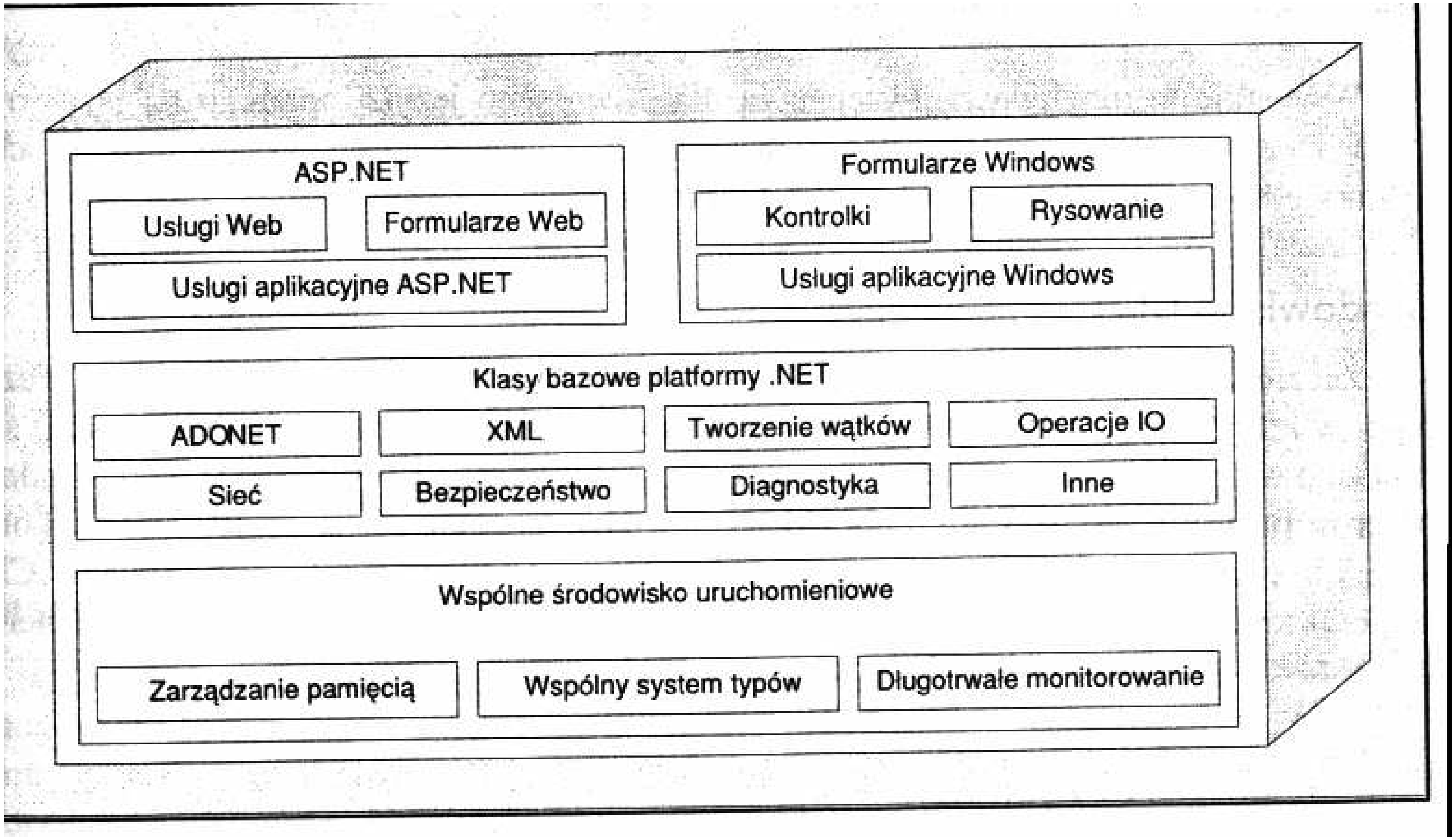
# .NET Framework

Główne składniki .NET Framework:



# .NET Framework

1. **CLR** (= Common Language Runtime) - wspólne środowisko uruchomieniowe; środowisko (maszyna wirtualna?) w którym wykonują się programy .NET; CLR dostarcza usług: zarządzania pamięcią (GC!), wątakami; dostarcza CTS

2. języki (programowania) .NET: Csharp, MC++ (Managed C++), VB.NET, ...

3. MSIL (= MS Intermediate Language) - język pośredni do którego tłumaczy się kod programów w językach .NET

4. sposób kompilacji i wykonania programu .NET:
   (a) kompilujemy program źródłowy do MSIL (powstaja plik exe lub dll)
   (b) uruchamiamy program jak zwykły exe; CLR tłumaczy MSIL do kodu maszynowego i wykonuje ten kod (interpretacja kodu IL metodą kompilacji JIT (=Just-in-time))

5. inne pojęcia:
   (a) CLS (= Common Language Specification);
   CLR jest konkretną implementacją CLS (inne impl.: "Mono" na linuxie?)
   (b) CTS (= Common Type System);
   dzięki temu języki .NET mogą łatwo ze sobą współpracować

# .NET - cechy charakterystyczne

1. automatyczne zarządzanie pamięcia (GC, Garbage Collection);
   podobnie jak w Javie!; działą także w MC++!

2. ogromna biblioteka klas ("lasy bazowe platformy .NET"); tych klas używa się DOKŁADNIE tak samo z różnych języków programowania!

3. jest możliwa ścisła współpraca RÓŻNYCH języków programowania! (np klasa programu MC++ może dziedziczyć z klasy zdefiniowanej w Csharp, a nie tylko używać obiektów tej klasy...); jest to możliwe dzięki CTS

4. typy aplikacji które można tworzyć pod .NET: konsolowe, GUI, web app (ASP.NET), web services (XML/SOAP), ...

5. kod zarządzany (ang. managed) i niezarządzany "przez CLR";
   np w MC++ może współistnieć kod zarządzany i niezarządzany, kod niezarządzany jest znacznie szybszy, jednak jest bardziej niebezpieczny i nie działają w nim dobrodziejstwa CLR np GC ...
   MSIL jest zawsze zarządzany!

6. metadane i atrybuty; programy .NET (pliki wykonywalne) mogą zawierać dodatkowe informacje wymagane przez CLR lub np podczas współpracy różnych języków; atrybuty można definiować w kodzie źródłowym (Csharp wew "[" "]")

7. **złożenie** (ang. assembly) to podstawowa jednostka instalacyjna; przechowywane w pliku dll lub exe; aplikacja może się składać z wielu złożeń być może napisanych w innych językach; złożenia obsługują nr wersji (mogę używać równocześnie 2 wersji tego samego złozenia)

8. w .NET: "komponent" <=> złożenie

9. klasy .NET mogą być udostępniane programom spoza .NET jako obiekty COM

10. .NET a język C++: 2 wersje języka C++...
    **MC++** jest "rozszerzeniem" C++, **C++/CLI** jest uważany za odrębny język (nowe słowa kluczowe?)

11. .NET a język Java: **Jsharp**; trzeba go doinstalować do .NET Framework 1.0; zasadniczo to tylko syntaktyka Javy; używa się std klas .NET; UWAGA: ma mniejsze możliwości niż Csharp!

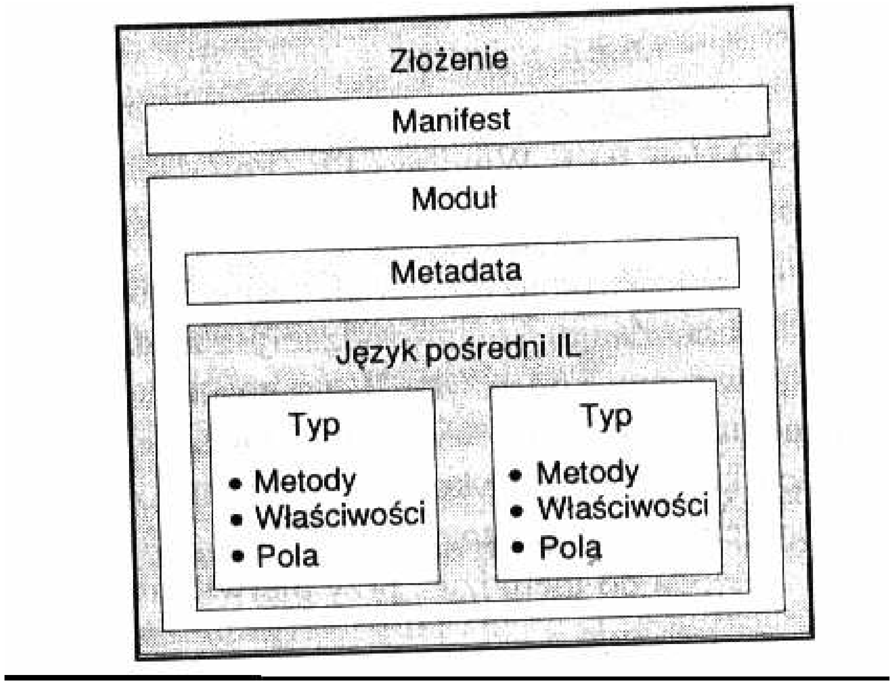
12. opis języka Csharp: specyfikacja ECMA 334; jest tam też opis nieformalny "nadający się do czytania":
    http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-334.pdf

13. książki o .NET:
    (a) "Wprowadzenie do .NET", J. Conrad, B. Francis, ... (wielu autorów)

# .NET - złożenia

Aplikacja .NET składa się ze złożeń;

1. złożenie zawiera: 1 manifest oraz moduły
2. każdy moduł zawiera metadane oraz pewną liczbę typów (czyli klas!)
3. klasy są skompilowane do IL (IL zawarty jest w dll/exe)
4. zawartość złożenia można zobaczyć przy pomocy disassemblera il "ildasm"



# .NET - MC++

## Przykład programu w MC++ korzystającego z GC ...

```
#using <mscorlib.dll>
using namespace System;
#include <stdio.h>
/* jak kompilowac programy MC++ ?
     cl /clr q2.cpp
*/

// klasa zarzadzana (patrz słowo __gc)
__gc class CCC {
  public:
    CCC(){ printf("CCC konstr\n"); }
    virtual ~CCC(){ printf("CCC destrukt\n"); }
    int i,j;
};
// klasa nie zarzadzana
class DDD {
  public:
    DDD(){ printf("DDD konstr\n"); }
    virtual ~DDD(){ printf("DDD destrukt\n"); }
    int i,j;
};
```

```
main()
{
  Console::WriteLine("A ku ku");
  printf("A ku ku 2\n");

  CCC *q1;
  q1= new CCC();
  //delete q1; // to nie jest konieczne
    // destruktor JEST wywolywany i tak!!!

  DDD *q2;
  q2= new DDD();
  delete q2; // jest konieczne
}
```

# .NET - Csharp

Język Csharp jest b. podobny do Javy/C++
Przykład programu w Csharp ...

```
/* plik lib.cs; kompilowac tak:
        csc /t:library lib.cs
   tworzy zlozenie jednomodulowe w pliku dll
*/
using System;
public class CCC
{
  public void pisz(int i)
  {
    Console.WriteLine("CCC: i="+i);
  }
  public String zwrocAkuku()
  {
    return "A ku ku !!!";
  }
};
```

```
/* plik prog.cs; kompilowac tak:
        csc prog.cs /r:lib.dll
   tworzy program, który bedzie uzywal zlozenia lib.dll
*/
using System;
class DDD
{
  public static void Main()
  {
    Console.WriteLine("A ku ku !!!");
    (new CCC()).pisz(321);
  }

};
```

```
/* plik prog_cpp.cpp; kompilowac tak:
        cl /clr prog_cpp.cpp
   tworzy program, który bedzie uzywal zlozenia lib.dll
*/
#using <mscorlib.dll>
using namespace System;
#using <lib.dll>
  // zamiast tego mozna uzyc opcji /FUlib.dll
#include <stdio.h>

void main()
{
  Console::WriteLine("A ku ku !!!");
  printf("A ku ku 2 !!!\n");
  (new CCC())->pisz(321);
}
```

# .NET - Csharp - typy

## Typy danych w Csharp (i w innych językach .NET):

```
object    object o = null; // "wszystko" jest typu object (to jest klasa!)
string    string s = "hello"; // string znaków Unicode!

sbyte // liczby całkowite
short int long byte ushort uint ulong

float     float val = 1.23F; // liczby "rzeczywiste"
double    double val1 = 1.23;
decimal   decimal val = 1.23M;

bool      bool val1 = true;
char      char val = 'h'; // znak Unicode
```

## (Proste) typy danych .NET też SĄ obiektami !!!

```
long l1; // kod MC++
System::Int64 l2; // long to tak naprawdę System::Int64
l1=123456; l2=654321;
System::Console::WriteLine(l1);
  // System - namespace, Console - klasa, WriteLine() - metoda statyczna
System::Console::WriteLine(l2);
printf("l1.GetHashCode()=%i\n", l1.GetHashCode()); // to działa !!!
printf("l2.GetHashCode()=%i\n", l2.GetHashCode());
```

# .NET - Csharp - typy C.D.

## Typy referencyjne i typy wartości

1. zmienna typu ref to tylko ref do danej na stercie
2. typy wartości są przechowywane na stosie (jak zm. automat. języka C)
3. typy ref to: **class**, interface, delegate, array
4. typy wartości to: wszystkie typy proste, enum, **struct**
5. boxing/unboxing - konwersja na (object) powoduje że obiekt przenosi się na sterte; konwersja do typu wartości przywraca pierwotną sytuację

```
using System;
struct Struct1 {
   public int Value;
}
class Class1 {
   public int Value = 0;
}
class Test {
   static void Mai n() {
      Struct1 val1  = new Struct1();
      Struct1 val2  = val1;
      val2.Value =  123;
```

```
        Class1 ref1 = new Class1();
        Class1 ref2 = ref1;
        ref2.Value = 123;
        Console.WriteLine("Values: {0}, {1}", val1.Value, val2.Value);
        Console.WriteLine("Refs: {0}, {1}", ref1.Value, ref2.Value);
        //System.Console.WriteLine("Refs: {0}, {1}", ref1.Value, ref2.Value);
           // System - namespace, Console - klasa, WriteLine - metoda statyczna
    }
}
/* wydruk:
Values: 0, 123
  Refs: 123, 123
*/
```

# .NET - Csharp - dziedziczenie, metody plimorficzne, interfejsy samoistne

```
using System;
public enum Color {
   Red, Blue, Green
}
public struct Point {
   public int x, y;
}
public interface IBase { // def interfejsu samoistnego!!!
   void F();
}
public interface IDerived: IBase { // dziedziczenie interfejsu
   void G();
}
public class A {
   protected virtual void H() { // metoda polimorficzna
                           // słowo virtual jest konieczne!
      Console.WriteLine("A.H");
   }
}
```

```
public class B: A, IDerived { // dziedziczenie z 1(!) klasy
                      // oraz implementowanie interfejsu
   public void F() {
      Console.WriteLine("B.F, implementation of IDerived.F");
   }
   public void G() {
      Console.WriteLine("B.G, implementation of IDerived.G");
   }
   override protected void H() { // "podmiana" metody polimorficznej
                // uwaga na słowo override !!! (virutal/ override)
      Console.WriteLine("B.H, override of A.H");
   }
}
```

# .NET - Csharp - tablice

```
class Test {
   static void Main() {
      int[] a1;          // single-dimensional array of int
      int[,] a2;         // 2-dimensional array of int
      int[,,] a3;        // 3-dimensional array of int
      int[][] j2;        // "jagged" (pol. szarpana) array: array of (array of int)
      int[][][] j3;      // array of (array of (array of int))
   }
}
class Test { // dodatkowo inicjalizacja tablic
   static void Main() {
      int[] a1 = new int[] {1, 2, 3};
      int[,] a2 = new int[,] {{1, 2, 3}, {4, 5, 6}};
      int[,,] a3 = new int[10, 20, 30];
      int[][] j2  =  new int[3][]; // [][] i [,] NIE są równoważne!
      j2[0]  = new  int[] {1, 2, 3};
      j2[1]  = new  int[] {1, 2, 3, 4, 5, 6};
      j2[2]  = new  int[] {1, 2, 3, 4, 5, 6, 7, 8, 9};
      a2[0,0]=123;
      j2[0][0]=123;
      System.Console.WriteLine(j2.Length); // tab. jest obiektem, zawiera pole Length
   }
}
```

# .NET - Csharp - parametry metod (ref i out)

Generalnie parametry przekazuje się przez wartość ...

```
class Test {
   static void Swap(ref int a, ref int b) {
      int t = a;
      a = b;
      b = t;
   }
   static void Main() {
      int x = 1;
      int y = 2;
      Console.WriteLine("pre: x = {0}, y = {1}", x, y);
      Swap(ref x, ref y);
        // ref jest dwukierunkowe (wejście ORAZ wyjście)
      Console.WriteLine("post: x = {0}, y = {1}", x, y);
   }
}
```

```
class Test {
   static void Divide(int a, int b, out int result, out int remainder) {
      result = a / b;
      remainder = a % b;
   }
   static void Main() {
      for (int i = 1; i < 10; i++)
         for (int j = 1; j < 10; j++) {
            int ans, r;
            Divide(i, j, out ans, out r);
              // out - tylko wyjście
            Console.WriteLine("{0} / {1} = {2}r{3}", i, j, ans, r);
         }
   }
}
```

# .NET - Csharp - właściwości i indeksatory

Właściwości (ang. properties) są podobne do pól obiektu, ale odczyt/ zapis właściwości powoduje uruchomienie pewnego kodu ... Indeksator powoduje że obiekt zachowuje się jak tablica.

```
using System;
class MyClass {
   public const int MyConst = 12;
   public int MyField = 34;
   public static int MyStaticField = 34;
   public void MyMethod(){
      Console.WriteLine("MyClass.MyMethod");
   }
   public int MyProperty { // def właściwości o nazwie MyProperty
     get { return MyField; }
     set { MyField = value; } // uwaga na słowo kluczowe value
   }
```

```
  public int this[int index] { // def indeksatora
     get { return 0; }
     set { Console.WriteLine("this[{0}] = {1}", index, value); }
  }
  public static MyClass operator+(MyClass a, MyClass b) {
                   // przeciążanie operatorów jest możliwe!
     return new MyClass(a.MyField + b.MyField);
  }
  public MyClass() {
     Console.WriteLine("Instance constructor");
  }
  public MyClass(int value) {
     MyField = value;
     Console.WriteLine("Instance constructor");
  }
  ~MyClass() {
     Console.WriteLine("Finalizer"); // to nie destruktor!!!
  }
  internal class MyNestedClass
  {}
}
```

```
class Test { // ... i sposób użycia
   static void Main() {
      // Instance constructor usage
      MyClass a = new MyClass();
      MyClass b = new MyClass(123);
      // Constant usage
      Console.WriteLine("MyConst = {0}", MyClass.MyConst);
      // Field usage
      a.MyField++;
      Console.WriteLine("a.MyField = {0}", a.MyField);
      // Method usage
      a.MyMethod();
      // Property usage
      a.MyProperty++;
      Console.WriteLine("a.MyProperty = {0}", a.MyProperty);
      // Indexer usage
      a[3] = a[1] = a[2];
      Console.WriteLine("a[3] = {0}", a[3]);
      // Overloaded operator usage
      MyClass c = a + b;
      // Nested type usage
      MyClass.MyNestedClass d = new MyClass.MyNestedClass();
  }
}
```

# .NET - Csharp - zdarzenia (events)

Język Csharp posiada wsparcie dla wzorca projektowego Obserwator...

```
/*  zdarzenia i delegacja w C# ...
      klasa Generator - generuje zdarzenia
      klasa Obserewator - obseruje zdarzenia
*/

public delegate void MojEventHandler(string s);
  // typ "delegatowy"; reprezentuje cos co można wywołać z podanymi parametrami
  //   może to być metoda obiektu lub metoda statyczna klasy
  // (przypomina wskazanie na funkcję w innych językach)

public class Generator {
  public event MojEventHandler zdarzenie1;
    // tak deklarujemy zdarzenie; jest tu pamiętana lista obserwatorów!

  public void generuj() {
    for(int i=0; i<3; i++) zdarzenie1("i="+i);
      // tak sie emituje zdarzenie !!!
  }
}
```

```
public class Obserwator
{
  private string nazwa;
  private Generator g;

  public Obserwator(string _nazwa, Generator _g) {
    nazwa=_nazwa; g=_g;
    g.zdarzenie1+= new MojEventHandler(obsluga_zdarzenie1);
      // w ten sposob dodaje sie obserwatora danego zdarzenia !!!
      //   (operator +=  i -=)
      // można dodawać metody zwykłe i statyczne
      //   event += new typ_delegatowy( ob.metoda );
      //   event += new typ_delegatowy( klasa.metoda_statyczna );
  }

  public void obsluga_zdarzenie1(string s) {
    System.Console.WriteLine ("obsluga_zdarzenie1/"+nazwa+"; "+s);
  }
}
```

```
public class MojaKlasa {
  static public void Main ()
  {
    System.Console.WriteLine ("--------------------");
    Generator g1= new Generator();
    new Obserwator("A",g1);
    new Obserwator("B",g1);
      // jesli nie bedzie zadnego obserwatora to jest zglaszany wyjatek!
      // (w metodzie generuj trzeba sprawdzac czy zdarzenie1==null)
    g1.generuj();
  }
}

/*
wydruk:
--------------------
obsluga_zdarzenie1/A; i=0
obsluga_zdarzenie1/B; i=0
obsluga_zdarzenie1/A; i=1
obsluga_zdarzenie1/B; i=1
obsluga_zdarzenie1/A; i=2
obsluga_zdarzenie1/B; i=2
*/
```

# klasy .NET jako obiekty COM

1. COM - Component Object Model, jest to system obiektów rozproszonych (w obrębie 1 maszyny), z epoki przed .NET

2. wiele języków programowania (pod Win) potrafi używać obiektów COM, np Word/VBA, VB, tcl/tcom

3. można spowodować że klasa .NET stanie się obiektem COM w następujący sposób:
   (a) zdefiniować klasę CCC2 i interfejs ICCC2 (to są nazwy przykładowe)
   (b) skompilować do złożenia lib2.dll "csc /t:library lib2.cs"
   *Uwaga: złożenie musi mieć tzw "silną nazwę", w tym celu trzeba: utworzyć plik z kluczem "sn -k q1.key", do kodu źródłowego dodać atrybut "[assembly:AssemblyKeyFileAttribute("q1.key")]", i dopiero potem kompilować ...*
   (c) zarejestrować obiekt COM: "regasm /codebase /tlb:lib2.tlb lib2.dll"
   *Uwaga: ta komenda spowoduje powstanie pliku .tlb z opisem interfejsu naszego obiektu COM, to jest tzw "type library"*

```
using System;
using System.Reflection;
[assembly:AssemblyKeyFileAttribute("q1.key")]

public interface ICCC2 { // ten interfejs jest NIEZBĘDNY !!!
  void przypisz(String[] ptab);
  String[] odczytaj();
};
public class CCC2: ICCC2 {
  private String[] tab;
  public CCC2 (){
    tab= new String [4];
    tab[0]= "qqq0"; tab[1]= "qqq1"; tab[2]= "qqq2"; tab[3]= "qqq3";
  }
  public void przypisz(String[] ptab)
  {
    int dlu=ptab.Length;
    tab= new String [dlu];
    for(int i=0; i<dlu; i++) tab[i]=ptab[i];
  }
  public String[] odczytaj(){
    return tab;
  }
};
```

```
# sposób użycia obiektu COM spod tcl/tcom:
lappend auto_path {D:\TclTk\tcom\lib\tcom}
package require tcom

::tcom::import "F:/dotnet/q5/lib2.tlb"
  # ladujemy opis interfejsu naszego obiektu COM

set w [::tcom::ref createobject "CCC2"]
  # utworzenie obiektu (instancji klasy CCC2)
set w2 [::lib2::ICCC2 $w]
  # ICCC2 to nazwa interfejsu;
  # do obiektu klasy CCC2 bedziemy sie odwolywac przez interfejs ICCC2

$w2 odczytaj
$w2 przypisz {qqq www eee 123123123123 "asd fasdf asdf"}
$w2 odczytaj
```