Programowanie aplikacji okienkowych Windows

- C#

Wykład

Język C# (I)

- Opracowany dla platformy .NET Framework
- Nie jest od razu kompilowany do postaci kodu maszynowego - po kompilacji powstaje IL (Intermediate Language), który jet interpretowany przez tzw NET CLR (Common Language Runtime)
- Program w C# jest tworzony w modelu obiektowym - poprzez budowanie nowych typów (głównie klas) oraz deklarowanie ich obiektów
- C# nie posiada wbudowanych bibliotek - korzysta z tzw Assemblies .NET Framework, zawierających kolekcje klas adresowane hierarchicznie

Język C# (II)

- Konwencja dla rozlokowania plików: każdą klasę powinno się umieszczać w osobnym pliku o nazwie odpowiadającej nazwie tej klasy, natomiast każdy plik powinien się znajdować w katalogu o nazwie odpowiadającej przestrzeni nazw, do której należy ta klasa.
- Z punktu widzenia kompilacji - ułożenie oraz nazwy plików są dowolne
- Program w C# to zbiór zdefiniowanych klas, przeważnie wywiedzionych z kolekcji

C# przykład

- Wejście do programu - metoda Main()

```
using System;
class Przyklad {
        static void Main( )
        {
                System.Console.WriteLine( " Tekst na konsole" ) ;
        }
}
```

Prezentacja - Main

Dokumentowanie (I)

- Nowa konwencja - bazująca na tagach XML zawartych w komentarzach kodu.
- Preferowane narzędzie: NDoc, generujący dokumentacje w formatach: HTML, XML, JavaDoc, LaTeX, MSDN, VS.NET. W poszczególnych przypadkach wymaga kompilatorów dokumentacji Microsoft (np HTML Help Compiler dla plików .msc)
- Obecna wersja funkcjonuje nad .NET 1.1, i w takiej wersji należy dostarczyć assembly.

Dokumentowanie (II)

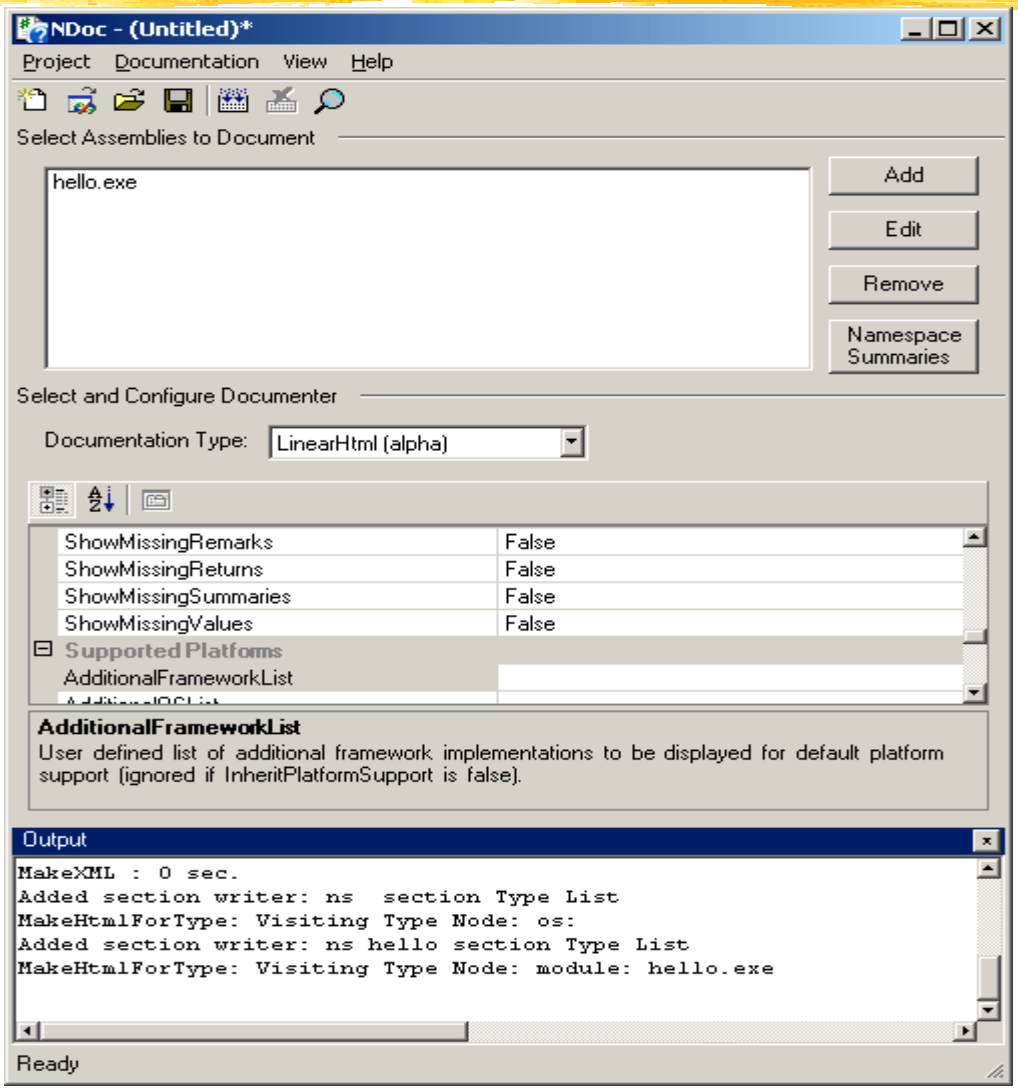
- Wybrane tagi XML stosowane w komentarzach:
  - <example> - przykład („rozwijanlny”), zawarty w opisie kodu
  - <include> - dołączenie do opisu treści zawartej w pliku zewnętrznym
  - <param> - parametr metody
  - <returns> - opis wartości zwracanej przez metodę
  - <see> - referencja do obiektu opisywanego
  - <see also> - podanie obiektu podobnego do obiektu opisywanego
  - <summary> - główny opis obiektu

Dokumentowanie (III)

- Procedura dokumentowania z użyciem NDoc:
  - Przygotowanie kodu z odpowiednimi komentarzami (tagi),
  - Kompilacja kodu do pliku exe lub biblioteki (.NET Assembly),
  - Generowanie dokumentu opisującego kod w XML :
    csc plik.cs /doc:plik.xml
  - Dodanie (Add) w NDoc pary plików .xml, oraz .exe (lub.dll)
  - Wygenerowanie dokumentacji w wybranym formacie

Dokumentowanie (IV)

- Ndoc



Prezentacja: Dokumentowanie

Operatory i wyrażenia

- Operatory C#, których znaczenie może nie być oczywiste:
  - typeof: zwraca typ obiektu, np.
    System.Type typ = typeof(Klasa);
  - ! ~ : negacja logicza (!) lub bitowa (~)
  - ( ) operator rzutowania typu, np. (typ) wartość
  - is operator zgodności typów. Np. if(x is int) { ... }
  - as : pozwala rzutować w dół hierarchii klas. Przy niepowodzeniu zwraca null.

Prezentacja: OperatoryIWyrazenia

Typy deklaracji zmiennych (I)

- Typy (I):
  - Bezpośrednie (value type): są strukturami przechowywanymi (na stosie) jako wartości. Do tej kategorii należą takie typy jak np. int, bool, float, char ... oraz struktury.
  - Referencyjne (reference type): ich wartości są przechowywane na stercie (w pamięci rezerwowanej dynamicznie), na stosie znajdują się jedynie referencje (wskazania) do obiektów. Tak deklarowane są klasy, tablice i interfejsy.

Typy deklaracji zmiennych (II)

- Typy (cd):
  - Wskaźnikowe (pointer type): używane do jawnego manipulowania pamięcią. Zabronione w blokach nienadzorowanych. Operatory, służace do obróbki wskaźnikó - jak w C
- Każdy typ bezpośredni posiada (niejawny) przypisany do niego typ referencyjny

Uwagi do deklaracji typów

- Przypisanie poprzez operator '=' typu referencyjnego polega na skopiowaniu wskazania na obiekt (nie powstaje nowy egzemplarz obiektu)
- Przypisanie typu bezpośredniego polega na skopiowaniu wartości całego obiektu.
- Dodatkowo występują:
  - void - typ pusty (np zwracany przez metodę)
  - null : słowo kluczowe oznaczające pustą referencję

Dostęp do zmiennych (I)

- Rodzaje dostępu (I):
  - public : składowa lub typ zadeklarowany jako publiczny są dostępne z dowolnego miejsca. Domyślny dla interfejsów.
  - private : składowa zadeklarowana jako prywatna jest dostępna tylko z wnętrza typu, w którym została zadeklarowana. Domyślny dla składowych klas i struktur.
  - protected : składowa zadeklarowana jako chroniona jest dostępna z wnętrza klasy, w której została zadeklarowana lub z wnętrza klasy pochodnej.

Dostęp do zmiennych (II)

- Rodzaje dostępu (cd):
  - internal : typ lub składowa typu są dostępne tylko z wnętrza assembly, w którym nastąpiła ich deklaracja (podczas kompilacji pliki .cs z kodem źródłowym programu są kompilowane w moduły - zgodnie z podziałem na przestrzenie nazw - a następnie grupowane w assemblies)
  - protected internal : składowa zadeklarowana z takim rodzajem dostępu jest widoczna z wnętrza klasy, w której została zadeklarowana (lub klasy pochodnej od niej) oraz z wnętrza assembly, w którym się znajduje.

Typy predefiniowane (I)

- int (System.Int32). Liczba całkowita ze znakiem, 4 bajty.
- uint (System.UInt32). Liczba całkowita bez znaku, 4 bajty.
- short (System.Int16). Liczba całkowita krótka ze znakiem, 2 bajty.
- ushort (System.UInt16). Liczba całkowita krótka bez znaku, 2 bajty.
- long (System.Int64). Liczba całkowita długa ze znakiem, 8 bajtów.
- ulong (System.UInt64). Liczba całkowita długa bez znaku, 8 bajtów.

Typy predefiniowane (II)

- byte (System.Byte). Pojedynczy bajt, bez znaku.
- sbyte (System.Sbyte). Pojedynczy bajt ze znakiem.
- char (System.Char). pojedynczy znak Unicode, 2 bajty. Literał tego typu zapisywany jest w apostrofach, np. ’b' lub '\u0053' (unicode). Dostępne są escape chars, np. '\n’, '\t’, ' \" ', '\".
- bool (System.Boolean) typ logiczny, wartości true lub false
- float (System.Single) liczba zmiennopozycyjna pojedynczej precyzji, 4 bajty.

Typy predefiniowane (III)

- double (System.Double) liczba zmiennopozycyjna podwójnej precyzji, 8 bajtów.
- decimal (System.Decimal) liczba w systemie dziesiętnym (12 bajtów). Przechowuje 28 cyfr oraz pozycję punktu dziesiętnego w tych liczbach. Ma mniejszy zakres w porównaniu z liczbami zmiennopozycyjnymi jednak zapewnia bardzo dużą precyzję przechowywania liczb o podstawie 10. Liczba zapisywana jako dziesiętna wymaga przyrostka 'm' lub 'M', np. decimal liczba = 10.1m;

Typy predefiniowane (IV)

- object (System.Object) typ bazowy dla wszystkich innych typów (z wyjątkiem wskaźnikowych)
- string (System.String) reprezentuje łańcuch (zmiennej długości) znaków Unicode. Deklaracje i operatory dla łańcuchów - jak w Javie

Typy predefiniowane - uwagi

- Łańcuchy (string) można tworzyć bez użycia operatora new, np. string str = " lancuch ”
- Istnieje dla łańcuchów sygnatura ‘@’ wyrażenia literalnego (dosłownego) - w nim nie będą brane pod uwagę escape chars, np: @"\dir\dane.txt" jest odpowiednikiem "\\dir\\dane.txt”
- W przypadku wartości typów zmiennopozycyjnych (float i double) możliwe jest zapisanie ?0 (zero), ?? (nieokreślony) oraz NaN (not a number).

Zmienne - modyfikatory zmiennych (I)

- Deklaracja : modyfikatory typ identyfikator = wartosc
- Modyfikatory (I):
  - static: składowa statyczna (istniejąca na rzecz całego typu, a nie pojedynczego obiektu tego typu). Składowe statyczne nie wymagają istnienia obiektów. Dana składowa statyczna istnieje jeszcze przed pojawieniem się pierwszego obiektu danej klasy
  - const: stała (składowa, której wartość jest obliczana w czasie kompilacji i nie może być zmieniana w czasie działania programu). Typ stałej musi być jednym z typów predefiniowanych

Zmienne - modyfikatory zmiennych (II)

- Modyfikatory (II):
  - readonly: wartość składowej nie będzie mogła być modyfikowana po początkowym nadaniu jej wartości. W odróżnieniu od stałych (const) wartości danych tego typu są obliczane podczas działania programu, a nie podczas kompilacji
  - volatile: pole typu nietrwałego. Zawsze odczyt lub przypisanie wartości takiej zmiennej powoduje fizyczne wywołanie odczytu lub zapisu do pamięci (brak optymalicacji). Przeznaczona do modyfikacji przez moduły obce.

Tworzenie zmiennych

- Aby utworzyć egzemplarz typu referencyjnego konieczne jest użycie operatora new. Alokuje on pamięć na stercie oraz wywołuje konstruktor podanego typu, w celu utworzenia obiektu.
- Wartości domyślne zmiennych:
  - null - dla referencji (gdy tworzymy zmienną typu referencyjnego bez przypisania do niej obiektu)
  - 0 - dla wszystkich typów liczbowych (np. int, float) i wyliczeniowych
  - false - dla typu logicznego bool.

Konwersje zmiennych

- Konwencja - klasyczna (jawnie - podaniem typu w nawiasie) lub niejawnie (przypisanie bez podania typu)
- W przypadku klas możliwe jest tworzenie operatorów konwersji jawnej i niejawnej do każdego z typów z osobna. Zostaną one użyte, przy konieczności dokonania konwersji z naszego obiektu na inny typ.
  Operator do konwersji niejawnej:

  ```
  public static implicit operator TypDocelowy( ... )
  ```

  Operator do konwersji jawnej:

  ```
  public static explicit operator TypDocelowy ( ... )
  ```

Konwersje zmiennych - przykład operatorów

- W klasie Cl1 - konwersja jawna do int i konwersja niejawna do char :

```
public class Cl1 {
        double liczba;
        char znak;
        public static explicit operator int ( A obiekt ) {
                return (int) obiekt.liczba ;
        }
        public static implicit operator char ( A obiekt ) {
                return obiekt.znak;
        }
}
```

Tablice (I)

- Zdefiniowane w System.Array
- Indeksowanie za pomocą operatora []
- Indeksy rozpoczynają się od zera
- Po utworzeniu tablicy jej długość nie może być już zmieniona
- W przypadku przekroczenia zakresu tablicy zgłaszany jest wyjątek *IndexOutOfRangeException*

Prezentacja: Tablice1

Tablice (II)

- Tworzenie tablic - przykłady:

```
double [] t1 = new double[ 20 ] ;
int[ ] t2 = { 11, 22 , 33, 44 } ;
MyClass [] t3= new MyClass[10];
int [,,] t4 = new int [2][2][3];
```

- Podobnie jak w przypadku C czy Javy tablica typów referencyjnych wymaga wypełnienia obiektami (utworzenia ich). Dla powyższego przykładu:

```
for ( int i=0; i < t3.Length; i++ )
t3[i] = new MyClass ();
```

Tablice nieregularne, długość tablicy

- Możliwe jest tworzenie tablic o nieokreślonej długości ostatnich wymiarów. Następnie - dla każdej pozycji w tablicy - konstruowanie nowej tablicy o różnym każdorazowo wymiarze:
  - int [ ] [ ] tab5 = new int [6] [ ];
  - for(int i=0; i < 6; i++)
    tab5[a] = new int[i] ;
- Długość tablicy pobieramy za pośrednictwem składowej Length: tab3.Length. Długość dowolnego wymiaru tablicy - za pośrednictwem metody getLength(int): tab4. getLength(2)

Prezentacja: Tablice2

Inicjalizacja tablic (I)

- Tablice jednowymiarowe
  - int[] n1 = new int[4] {2, 4, 6, 8};
  - int[] n2 = new int[] {2, 4, 6, 8};
  - int[] n3 = {2, 4, 6, 8};

  - string[] s1 = new string[3] {"John", "Paul", "Mary"};
  - string[] s2 = new string[] {"John", "Paul", "Mary"};
  - string[] s3 = {"John", "Paul", "Mary"};

Inicjalizacja tablic (II)

- Tablice wielowymiarowe
  - int[,] n4 = new int[3, 2] { {1, 2}, {3, 4}, {5, 6} };
  - int[,] n5 = new int[,] { {1, 2}, {3, 4}, {5, 6} };
  - int[,] n6 = { {1, 2}, {3, 4}, {5, 6} };

  - int[][] n7 = new int[2][] { new int[] {2,4,6}, new int[] {1,3,5,7,9} };
  - int[][] n8 = new int[][] { new int[] {2,4,6}, new int[] {1,3,5,7,9} };
  - int[][] n9 = { new int[] {2,4,6}, new int[] {1,3,5,7,9}};

Przekazywanie tablic do funkcji

```
class Klasa
{
    public static void WypiszKolekcje(params object[] tablica)
    {
        foreach (object pole in tablica)
        {
            System.Console.Write(pole + "\t");
        }
       System.Console.ReadLine();
    }

    static void Main()
    {
       WypiszKolekcje(100, ”tekst", 200);
    }
}
```

Typ enum

- Definiowany tradycyjnie - za pośrednictwem słowa kluczowego enum.
- Przykład:
  - public enum Kolory {czerwony, niebieski, bialy};
  - Kolory mojKolor = Kolory.czerwony;
- lub z uporządkowaniem podlegającym obliczeniom numerycznym:
  - public enum Kolory {czerwony=1, niebieski=2, bialy=3};

Definiowanie klas

- Deklaracja i definicja klasy:
  [modyfikatory] class nazwa [: nazwaNadklasy , interfejsy] {
  // Deklaracje składowych klasy
  }
- Uwagi: możliwe jest stosowanie słów kluczowych:
  - this - referencja do obiektu, z którego użyto this
  - base - odwołanie do składowych klasy bazowej

Modyfikatory klas

- Modyfikatory:
  - public lub internal - modyfikuje dostęp do klasy. Domyślnie klasa jest prywatna.
  - sealed - powoduje, że z danej klasy nie można dziedziczyć, np:
    sealed class Cl4 { ... }
  - abstract : tworzy klasę abstrakcyjną mogącą zawierać metody abstrakcyjne, uniemożliwiającą tworzenia swoich instancji
- Konstruktor standardowy jest bezparametrowy, destruktor (jak w C++) - zapisywany ze znakiem '~'

Metody - modyfikatory metod (I)

- Deklaracja metody:
  [modyfikatory] typ nazwa ([parametry]) { ... }
- Modyfikatory:
  - public, private, protected, internal, protected internal: modyfikuje dostęp do metody tak samo jak w przypadku zmiennych.
  - virtual: pozwala tworzyć metodę wirtualną (jak C++)
  - extern: wskazuje, że metoda jest zaimplementowana z użyciem kodu nienadzorowanego
  - static: metoda statyczna działa na rzecz całego typu, a nie pojedynczego obiektu.

Metody - modyfikatory metod (II)

- Modyfikatory argumentów metod (I):
  - ref: pozwala przekazać dany parametr przez referencję Modyfikator ref musi się pojawić na liście parametrów metody oraz podczas jej wywoływania).
  - out: służy przekazaniu wartości z metody na zewnątrz. Przekazując do metody zmienną w trybie out przekazujemy ją przez referencję, oczekując jednocześnie że wewnątrz metody zostanie jej nadana odpowiednia wartość. Modyfikator out musi się pojawić na liście parametrów metody oraz podczas jej wywołania.

Metody - modyfikatory metod (III)

- Modyfikatory argumentów metod (I):
  - params: metoda może przyjmować dowolną liczbę (takich) parametrów. Modyfikator params może być zastosowany tylko do ostatniego parametru metody, przykład:

```
public class Cl2 {
        public void metoda ( params int[ ] t6 ) { ...}
} // class

Cl2 obiekt = new Cl2( ) ;
obiekt.metoda (1,2,3);
```

Metody - modyfikatory metod (IV)

- Modyfikatory get i set:
- Umożliwiają niejawne wywoływanie metod automatyzujących proces przypisywania danych prywatnych do obiektu i pobierania ich z obiektu danej klasy. Zastępują stosowane np w Javie zbiory metod publicznych typu set..(), get..() służących do przypisywania wartości składowym prywatnym. Składowe prywatne nazywamy wtedy właściwościami
- Gdy zdefiniujemy tylko get - otrzymujemy właściwość read-only
- Oprócz get i set możliwe jest użycie this[...] dostarczającego funkcji indeksowania właściwości

Metody - modyfikatory metod (V)

- Przykład modyfikatorów get i set:

```
public class Cl3 {
      float  liczba ;          // pole  prywatne
      public  int  dane( )  {
            set {   liczba = (float)value ; }
            get {   return (int)liczba ; }
      }
}
Cl3 obiekt = new Cl3( );
obiekt. dane = 100 ;                  // użycie set
Console.Write( obiekt.dane );         // użycie get
```

Prezentacja: GetSet

Metody - modyfikatory metod (VI)

- Przykład modyfikatora indeksującego public
- class Cl4 {
- int[ ] dane = new int[rozm1 * rozm2] ;
- public int this [int x, int y] {
- get { return dane[x + y * rozm1] ; }
- set { dane[x+y * rozm1] = (int) value ; }
- }
- }
- ...
- Cl4 obiekt = new Cl4 (5,5);
- obiekt[0,0] = 0 ; // klasa indeksowana

Parametry domyślne metod

- Nie można ich definiować w metodzie, ale można zasymulować:

- ```
  class Klasa {
        public metoda(string lancuch) {
              // tu kod ....
        }
        public metoda() {
              metoda(”tekst_domyslny");
        }
  } //class
  ```
  ```

Parametry domyślne metod

- Nie można ich definiować w metodzie, ale można zasymulować:

- ```
  class Klasa {
        public metoda(string lancuch) {
              // tu kod ....
        }
        public metoda() {
              metoda(”tekst_domyslny");
        }
  } //class
  ```
  ```

Przeciążanie operatorów

- Oprócz omówionych już modyfikatorów get i set w C# istnieje klasyczny mechanizm przeciążania operatorów.
- Operatory, które można przeciążać to:
  + - * / % ! != == ~ << >> < <= > >= & | ^ ++ --
- Operatory przeciążone zachowują swój priorytet
- Istnieją pary operatorów, które muszą być jednocześnie przeciążone. Są to : (<,>), (==,!=) oraz (<=,>=)
- Przeciążenie następuje poprzez zdefiniowanie funkcji statycznej o nazwie ‘operator’:

  operator symbol_operatora( )

Struktury

- Struktury są zawsze typu sealed, mogę jednak implementować interfejsy dostępowe (!)
- Struktury zapisywane są podobnie do C/C++:
- [modyfikatory] struct nazwa [: interfejsy]
- {
- // składowe
- }
-

Przykład:

- struct dane {
- int a, b ;
- public void metoda() {...}
- }

Interfejsy (I)

- Jest abstrakcyjny - jako kolekcja metod abstrakcyjnych (identycznie jak w Javie)
- Deklaracja:
  [modyfikatory] interface nazwa [: interfejsyBazowe]
- {
- // Deklaracje składowych interfejsu
- }
- Klasa (lub struktura) może implementować wiele interfejsów. Interfejs może dziedziczyć po wielu innych interfejsach

Interfejsy (II)

- Przykład: klasa Cl5 implementuje interfejs In1:

```
public interface In1 {
      void metoda( ) ;  // z definicji abstrakcyjna i public
}

public class Cl5 : In1 {
     void metoda()
     {
           // implementacja metody interfejsu
     }
}
```

Klasy generyczne (I)

- Dostępne od wersji 2.0
- Razem z wprowadzeniem klas generycznych rozszerzone zostały kolekcje list,kolejek, stosów .NET o wersje szablonowane
- Odpowiednik szablonów klas C++ - umożliwienie stworzenia klasy obsługującej różne typy danych
- Możliwe jest szablonowanie zarówno klas jak i metod

Klasy generyczne (II)

- Przykład:

```
public class Operacje<Szablon>
    {
            Szablon[]   dane;
            public void Wpisz(Szablon item) {...}
            public Szablon Pobierz() {...}
    }

Operacje <int>    obiekt  = new Operacje <int>();
obiekt.Wpisz(14);
int x = obiekt.Pobierz();
```

Częściowe definicje klas (I)

- Dostępne od wersji 2.0
- Klasy można tu tworzyć fragmentarycznie w wielu oddzielnych definicjach. Złożenie definicji tworzy dopiero pełną charakterystykę klasy.
- Wszystkie części tej samej kasy muszą być zdefiniowane w jednej przestrzeni nazw
- Gdy klasa taka dziedziczy - wskazanie nadklasy może się znajdować tylko w jednej części
- Do deklarowania częściowej definicji klasy służy słowo kluczowe ‘partial’

Częściowe definicje klas (II)

- Przykład - klasa Cl6 w dwóch blokach:

```
public partial class Cl6{ // Pierwszy blok
        private int dane1;
        public void metoda1() {....};
}

public partial class Cl6{ // Drugi blok
        private int dane2;
        public void metoda2() {....};
}
```

Dziedziczenie klas

- Składnia:

```
class nazwa [ : nazwaKlasyBazowej]
```

- Nie jest możliwe dziedziczenie wielokrotne (z wielu klas jednocześnie)
- Przykład:

```
class Cl7 : Cl6 {
        public Cl7 () { }
}
```

- Istnieje słowo kluczowe **base** umożliwiające dodanie do listy inicjalizacyjnej konstruktora także składowych zdefiniowanych w nadklasie (jako celu dla wartości z parametru konstruktora)

Operator is a klasy

- Operator ‘is’ służy do sprawdzenia, czy obiekt można rzutować na wybrany typ bez rzucania wyjątków:

```
object obiekt = new MojaKlasa();
( .... )
if (obiekt is MojaKlasa)
{
        ((MojaKlasa)obiekt).Metoda();
}
```

- Możemy też sprawdzać, czy obiekt implementuje wybrany interfejs:

```
if (obiekt is ITestable)
```

Operator as przy konwersji typów

- Operator ‘as’ wykonuje rzutowanie. Jeśli rzutowanie nie jest możliwe, zwraca null.
- Przykład:
- protected void clicked(object sender, EventArgs e)
- {
- Button b = sender **as** Button;
- if (b != null)
- {
- b.Text = „pressed”;
- }
- }

Przeciążanie metod

- Metoda wirtualna (virtual) może zostać ponownie zdefiniowana w klasie wywiedzionej. Na przesłonięcie w tej klasie pozwala słowo kluczowe override:

```
class Cl6 {
        public virtual void wypiszNazwę( ) {
                System.Console.Write(”tresc 1");
        }
}
class Cl7 : Cl6 {
        public override void wypiszNazwę( ) {
                System.Console.Write(”tresc 2");
        }
}
```

Klasy i metody abstrakcyjne

- Definicja metody wirtualnej zezwalała na jej redefinicję w klasie wywiedzionej,
- Definicja metody abstrakcyjnej wymusza zadeklarowanie jej klasy jako abstrakcyjnej i uniemożliwia tworzenie instancji bezpośrednio z tej klasy.
- Przykład:

```
abstract class Cl8 {
        public abstract void metoda( ) ;
}
```

Prezentacja: KlasyAstrakcyjne

Instrukcje strukturalne

- Identyczne jak w C/Java: if , switch, while, do..while, for.
- Dodatkowa instrukcja foreach:

```
foreach ( iterator in IEnumerable )
{
    ...
}
```

- Instrukcje sterujące: return, break, continue, goto.

Instrukcje strukturalne - uwagi

- Warunek może być zadany wyłącznie poprzez bool (nie ma konwersji pomiędzy bool i typami liczbowymi)
- W switch wymuszenie przejścia do następnego case musi być jawne (wykonane poprzez goto case [wartosc];). Wyjątek stanowi zastosowanie kilku case ‘pustych’ bezpośrednio po sobie
- W foreach klasa, po której obiekcie następuje iteracja musi implementować interfejs IEnumerable
- W for zmienne deklarowane w instrukcji obowiązują wyłącznie w jej bloku

Obsługa wyjątków (I)

- Identyczna jak w Javie.
- Podstawowa składnia:

```
try {
        ...
} catch {
        ...
} finally {
        ...
}
```

- Słowo kluczowe throw rzuca wyjątek zadany argumentem (przeważnie w tym momencie dynamicznie konstruowany do obiektu)

Prezentacja: Wyjątki1

Obsługa wyjątków (II)

- Przy łapaniu wyjątków obowiązuje hierarchia klas wyjątków. Na jej szczycie stoi System.Exception, zawierająca składowe Message i StackTrace
- Wszystkie instrukcje skoku (throw, return, break, continue, goto) są ograniczone przez instrukcję try.
- Skok poza blok try zawsze powoduje wykonanie bloku finally w danym bloku try.
- Nie można wykonać skoku z wnętrza na zewnątrz bloku finally.

Prezentacja: Wyjątki2

Przestrzenie nazw (I)

- Porządkują hierarchię klas - zapobiegają występowaniu konfliktów w nazwach klas
- Grupują klasy o podobnym zastosowaniu
- Znaczna część - systemowe. Inne - własne, zdefiniowane w programie
- Konieczność importu obcego namespace
- Separator namespace i klas: ‘.’
- Skoncentrowane wokół tzw. assembiles w Assembly Cache maszyny .NET

Przestrzenie nazw (II)

- Przykład deklaracji:

```
namespace Nazwa
{
    namespace NazwaWewnetrzna
    {
        class Klasa
        {
            ...
        }
    }
}
```

Import przestrzeni nazw

- Hierarchia adresowania:
  namespace.klasa.metoda
- Przykład adresowania namespace:
  System
- Przykład adresowania namespace+klasa:
  System.Console
- Przykład adresowania namespace+klasa+metoda:
  System.Console.WriteLine (”tekst”);
- Do importowania służy słowo kluczowe ‘using’ , np:
  using System;

  Możliwe jest importowanie tylko w trybie namespace, niemożliwe: namespace+klasa.

Metody delegowane

- Odpowiednik wskaźników do funkcji
- Słowo kluczowe: delegate
- W klasie możliwe jest zdefiniowanie tzw. delegacji metod z danej klasy, określającej typ parametrów oraz zwracanej przez metody wartości. Delegacja posiada nazwę.
- Poza klasą możliwe jest utworzenie tzw. Delegata - na bazie klasy delegującej swoje metody i z podaniem nazw tych metod. Delegat otrzymujący parametry automatycznie wywołuje metodę delegowaną.

Metody delegowane - przykład

```
public class Cl9 {
        public delegate int delegacja(int a, int b);
                                                // delegacja metod
        public int Dodaj(int a, int b) { metoda delegowana
                return a + b;
        }
}
Cl9 d = new Cl9 ();         // instancja klasy delegującej
Cl9.delegacja dodaj = new Cl9.delegacja(d.Dodaj);
int liczba = dodaj(4, 6);
```

Zdarzenia

- Mechanizm delegowania metod jest wykorzystywany do obsługi zdarzeń. Zarejestrowanie obsługi zdarzenia sprowadza się do stworzenia delegata
- Słowo kluczowe event informuje kompilator, że dany delegat może być wywołany tylko przez klasę, która go definiuje, a inne klasy mogą ten delegat jedynie subskrybować lub rezygnować z subskrybcji.

Obsługa zdarzeń - przykład

```
public delegate void EventHandler(object sender,
EventArgs e);
public class Przycisk {
      public event EventHandler zdarzenie;
}
Przycisk Button1
Button1.zdarzenie += new EventHandler(obsluga);
void obsluga(object sender, EventArgs e) {
      // obsługa zdarzenia
}
```