



## Język C#

- Język obiektowy
- Premiera – lipiec 2000 r.
- Anders Hejlsberg, Scott Wiltamuth, Peter Golde
- Standard ECMA-334
- Rozwija się – wersja 4.0 – czerwiec 2006

## Podobieństwa C# i Javy

```

if (prec == 0) //2th order
{
    for (i = 2; i <= nx - 2; i++)
        for (j = 2; j <= nz - 2; j++)
            pp[i][j] = (float)(2.0 * p[i][j] - pm[i][j]
                + ((dtr * dtr) / (ds * ds)) * V[i][j]
                + V[i][j] * (p[i + 1][j] + p[i - 1][j]
                + p[i][j + 1] + p[i][j - 1]
                - 4.0 * p[i][j]) + s[i][j]);
}
else //4th order
{
    for (i = 2; i <= nx - 2; i++)
        for (j = 2; j <= nz - 2; j++)
            pp[i][j] = (float)((2.0 -
                5.0 * Math.Pow((V[i][j] * dtr) / ds, 2))
                * p[i][j] - pm[i][j] + (4.0 / 3.0)
                * Math.Pow((V[i][j] * dtr) / ds, 2)
                * (p[i + 1][j] + p[i - 1][j] + p[i][j + 1]
                + p[i][j - 1]) - (1.0 / 12.0)
                * Math.Pow((V[i][j] * dtr) / ds, 2)
                * (p[i + 2][j] + p[i - 2][j] + p[i][j + 2]
                + p[i][j - 2]) + s[i][j]);
}

```

## Java -> C#, C# -> Java???

[<http://tech.pgs-soft.com/2011/11/07/eclipse-poprawia-jawe-witaj-xtend/>]

Eclipse poprawia Javę. WitajXtend!

Dodano: 7 listopada 2011 Autor: Tomasz Piechaczek

Fundacja Eclipse po cichu wprowadziła na rynek nowy język programowania – Xtend. Jego podstawowym zadaniem jest ułatwienie tworzenia czytelnego kodu. Twórcy zapewniają, że pomoże on również uzupełnić biblioteki Javy o konieczne lecz nie zawarte w pakiecie funkcje.

Kolejnym uproszczeniem jest intuicyjny dostęp do pól obiektów. Teraz zamiast pisać `osoba.setName("Robert")` możemy wykorzystać składnię `osoba.name = "Robert"`. Opcjonalne zastosowanie średników, nawiasów oraz lukier składniowy to tylko kilka z wprowadzonych zmian.

Mechanizm atrybutów został z C# zapożyczony do języka Java w wersji 1.5 (adnotacje), jakkolwiek samo Reflection API istniejące od pierwszego wydania języka stanowiło inspirację dla twórców C# [(nie?) pewne źródło w Internecie]

## Przykładowy program

```
using System;
using System.Collections.Generic;
using System.Text;

namespace WitajSwiecie
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Czesc");
        }
    }
}
```

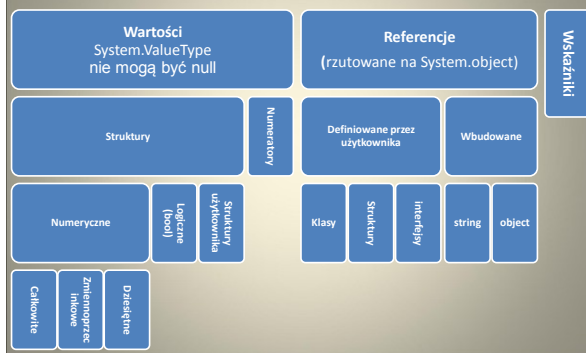
Import innych przestrzeni nazw

przestrzeń nazw programu (czy też biblioteki DLL)

Klasa główna programu

main !!!

## Typy C#



## Typy wbudowane C#

C# Type	.NET Framework Type
bool	System.Boolean
byte	System.Byte
sbyte	System.SByte
char	System.Char
decimal	System.Decimal
double	System.Double
float	System.Single
int	System.Int32
uint	System.UInt32
long	System.Int64
ulong	System.UInt64
object	System.Object
short	System.Int16
ushort	System.UInt16
string	System.String

## Typy zmiennoprzecinkowe

IEEE 754

Typ	I. bitów	precyzja	zakres
float	32	23b => 7 cyfr	od $1.5 \cdot 10^{-45}$ do $3.4 \cdot 10^{38}$
decimal	64	52b => 15 cyfr	od $5.0 \cdot 10^{-324}$ do $1.7 \cdot 10^{308}$

```
float a = (float)0.00001;
float b = (float)10000.0;
float c = a + b;
c = c - b;

Console.WriteLine(c);
```

0

Typ decimal:

checked – sł. kluczowe: sprawdzanie przekroczenia zakresu

- precyzja 128 bitów,
- nie kumuluje błędów podczas operacji zmiennoprzecinkowych,
- przeznaczony do rzetelnego opisu danych o przewidywalnym przecinku (np. finanse),
- brak wsparcia przez operacje sprzętowe CPU – każda operacja to kilka instrukcji.

## Definiowanie zmiennych

```
int a;           // zmienna nieutworzona - użycie to błąd!
int b = 0;       // zmienna utworzona i zainicj.
                // domyslny konstruktor, jeśli istnieje...
int c = new int(); // zmienna utworzona konstruktorem
c = 10;

string aa = "C:\\plik.txt"; // string i znaki \
string ab = @"C:\plik.txt"; // string surowy
```

## Zmienne referencyjne

```
int a = 7;
int b = a;
a++;
b--;
Console.WriteLine(a);
Console.WriteLine(b);
```

8  
6

```
class Klasa
{
    public int c;
}

Klasa aa = new Klasa();
aa.c = 5;
Klasa bb = aa;
aa.c++;
bb.c--;

Console.WriteLine(aa.c);
Console.WriteLine(bb.c);
```

5  
5

struct zamiast class -  
nie jest referencyjne!

## Tablice

```
int a;
int[] b;
int[] c = null;
int[] d = new int[100];
int[] e = new int[] {10, 11, 12};
int[,] f = new int[9, 9];
int[,] maska = new int[2, 3] {
    { 1, 2, 1 },
    { 1, 0, 1 } };

Console.WriteLine(maska.Length);
Console.WriteLine(maska.GetLength(0));
Console.WriteLine(maska.GetLength(1));
```

623

maska[5][5].Length zwraca liczbę wszystkich elementów !

## Tablice prostokątne a nieregularne

### Tablice prostokątne:

- szybki dostęp do danych,
- łatwa obsługa,
- zwarty obiekt w pamięci –  
ograniczenie rozmiaru obiektu do .NET 4.0 włącznie nawet przy 64bit – 2GB;
- bardzo wolna serializacja

### Tablice nieregularne (ang. jagged):

- tablice tablic – odpowiednik tablic wielowymiarowych w C/C++,
- luźno rozmieszczone w pamięci,
- alokacja wymaga podejścia jak w C/C++,
- tablica może być częściowo zaalokowana → wybrane podtablice = null,
- szybsza serializacja.

## Tablice nieregularne

```
// ograniczony rozmiar
public ushort[,] mapa = null;

// wiekszy rozmiar, .NET 4.5
public ushort[][] mapa_j = null;

// nie daje rady zaalokowac dla duzych wartosci
mapa = new ushort[max_dlugosc_g, max_szerokosc_g];

// alokacja mapy typu jagged
mapa_j = new ushort[max_dlugosc_g][];

int u;
for (u = 0; u < max_dlugosc; u++)
    mapa_j[u] = new ushort[max_szerokosc_g];

// wypelnienie mapy zerami
int i, j;
for (i = 0; i < max_dlugosc; i++)
    for (j = 0; j < max_szerokosc_g; j++)
        mapa_j[i][j] = 0;
```

## Tablica mieszana

```
int[,] maski5x5 = {

    new int[5, 5] { { 0, 0, 0, 0, 0 },
                    { 0, 0, 0, 0, 0 },
                    { 0, 0, 1, 0, 0 },
                    { 0, 0, 0, 0, 0 },
                    { 0, 0, 0, 0, 0 } },

    new int[5, 5] { { 0, 0, 0, 0, 0 },
                    { 0, 0, 1, 0, 0 },
                    { 0, 1, 1, 1, 0 },
                    { 0, 0, 1, 0, 0 },
                    { 0, 0, 0, 0, 0 } }
};

int Konwolucja_z_maska(int[, ] poj_maska);
Konwolucja_z_maska(maski5x5[0]);
```

## Dziedziczenie

W C# klasa może dziedziczyć tylko po jednej klasie, ale za to po wielu interfejsach

```
public class Bazova
{
    int baza;
};

public class Nova : Bazova, ISerializable, IDisposable
{
    int drugi;
    void ISerializable.GetObjectData(
        SerializationInfo info, StreamingContext context)
    {
        info.SetType(typeof(int));
    }

    public void Dispose()
    {
        Dispose();
        GC.SuppressFinalize(this);
    }
}
```

## Dziedziczenie

Modyfikatory:

- **private** – dostęp do składowych klasy tylko dla jej metod,
- **protected** – dostęp do składowych klasy tylko dla jej metod oraz metod klas potomnych
- **internal** – dostęp do składowych klasy tylko w danej jednostce kompilacji (dll, exe),
- **public** – pełny dostęp, także poza jednostką kompilacji (C++ - odpowiednik extern)
- **protected internal** – dostęp do składowych klasy tylko dla jej metod oraz metod klas Potomnych w obrębie danej jednostki kompilacji.

## Parametry wywołania funkcji

```
static int FProsta(int a)
{
    a = a * 2;
    return a + 1;
}

static void Main(string[] args)
{
    int b = 1;
    int c = 0;
    Console.WriteLine(b);
    c = FProsta(b);
    Console.WriteLine(b + " " + c);
    Console.ReadKey();
}
```

```
1
1 3
```

## Parametry wywołania funkcji - ref

```
static int FzRef(ref int a)
{
    a = a * 2;
    return a + 1;
}

static void Main(string[] args)
{
    int b = 1;
    int c = 0;
    Console.WriteLine(b);
    c = FzRef(ref b);
    Console.WriteLine(b + " " + c);
    Console.ReadKey();
}
```

```
1
2 3
```

## Parametry wywołania funkcji - out

```
static int FzOut(out int a)
{
    a = 5; // a trzeba zainicjować!
    return a + 1;
}

static void Main(string[] args)
{
    int b = 1;
    int c = 0;
    Console.WriteLine(b);
    c = FzOut(out b);
    Console.WriteLine(b + " " + c);
    Console.ReadKey();
}
```

```
1
5 6
```

## ref a tablice

```
static void FzRef(ref int[] a)
{
    a[0] = 8;
    a = new int[] { 10, 20 };
    a[1] = a[1] * 7;
}

static void FbezRef(int[] a)
{
    a[0] = 8;
    a = new int[] { 10, 20 };
    a[1] = a[1] * 7;
}

static void Main(string[] args)
{
    int[] b = new int[] { 1, 2 };
    int[] c = new int[] { 1, 2 };
    Console.WriteLine(b[0] + " " + b[1]);
    Console.WriteLine(c[0] + " " + c[1]);
    FzRef(ref b);
    FbezRef(c);
    Console.WriteLine(b[0] + " " + b[1]);
    Console.WriteLine(c[0] + " " + c[1]);
    Console.ReadKey();
}
```

Z ref i bez ref –  
można skutecznie  
modyfikować zawartość tabeli

W funkcji z ref można utworzyć  
nowy obiekt,  
bez ref też, ale nie jest on  
przypisywany po wykonaniu  
funkcji

Z ref jest trochę wolniej

```
1 2
1 2
10 140
8 2
```

## Wątki

```
using System;
using System.Threading;

class MojaZWatkciem
{
    public int a;

    public void Watkowa()
    {
        while (true)
        {
            a++;
            Thread.Sleep(1000)
        }
    }
}

class Program
{
    static void Main(string[] args)
    {
        MojaZWatkciem mzw = new MojaZWatkciem();

        Thread watek =
            new Thread(new ThreadStart(mzw.Watkowa));
        watek.Start();

        for (int i = 0; i < 1000; i++)
        {
            Console.WriteLine(" "+mzw.a);
            Thread.Sleep(500);
        }
    }
}
```

## Pula wątków

```
class MojaZWatkciem
{
    public int a;
    public void Watkowa(object o)
    {
        while (true)
        {
            a++;
            Thread.Sleep(1000);
        }
    }
}

class Program
{
    static void Main(string[] args)
    {
        MojaZWatkciem mzw = new MojaZWatkciem();
        ThreadPool.QueueUserWorkItem(new WaitCallback(mzw.Watkowa));

        for (int i = 0; i < 1000; i++)
        {
            Console.WriteLine(" "+ mzw.a);
            Thread.Sleep(500);
        }
    }
}
```

## Synchronizacja wątków

- nieblokujące metody synchronizacji dostępu do obiektów  
`Interlocked`, `Thread.MemoryBarrier`, `volatile`,  
`Thread.VolatileRead/VolatileWrite`
- metody blokowania wątków:
  - `Sleep()`, `Join()`, `Suspend()`, `Resume()`, `Task.Wait()`
- metody blokowania przy użyciu obiektów synchronizacyjnych:
  - na wyłączność:
    - `lock`, `Monitor.Enter/Exit`, `Mutex`, `SpinLock`,  
`ReaderWriterLock`
  - ogólne (globalne):
    - `Semaphore`, `SemaphoreSlim`,
- definiowanie metod synchronizowanych,
- metody wykorzystujące sygnalizowanie:
  - `Monitor.Wait/Pulse`, `CountdownEvent`, `ManualResetEvent`,  
`Barrier`

## lock

```
class KlasaZObiektem
{
    private object obiekt_synchronizacyjny_kolejki = new object();

    public int DodajDoKolejki(string s)
    {
        lock (obekt_synchronizacyjny_kolejki)
        {
            // operacja (krytyczna) dodania
        }
        return 1;
    }

    public int UsunZKolejki(string s)
    {
        lock (obekt_synchronizacyjny_kolejki)
        {
            // operacja (krytyczna) usuniecia
        }
        return 1;
    }
}
```

`lock` – pozwala na dostęp do obiektu tylko jednej metodzie naraz w obrębie podanego bloku kodu ( {} )

## Monitory

```
public static class Monitor
{
    // [...]
    public static void Enter(object obj);
    public static bool TryEnter(object obj);
    public static void Exit(object obj);

    public static bool IsEntered(object obj);

    public static bool Wait(object obj);
    public static bool Wait(object obj, int msecTimeout);
    public static void Pulse(object obj);
    public static void PulseAll(object obj);
    // [...]
}
```

wait : 1. zwalnia blokadę obiektu,  
2. blokuje bieżący wątek aż do otrzymania pulsu z zewnątrz,  
3. próbuje pobrać obiekt na nowo

## Metody synchronizowane

```
class MojaKlasaSynchr
{
    public int FunkcjaNiezsynchronizowana(int a)
    {
        a++;
        return a;
    }

    public int FunkcjaZsynchronizowana(int b)
    {
        lock (this)
        {
            return FunkcjaNiezsynchronizowana(b);
        }
    }

    [MethodImpl(MethodImplOptions.Synchronized)]
    public int FunkcjaSynchronizowana(int c)
    {
        c++;
        return c;
    }
}
```

## Bariera pamięci

**Thread.MemoryBarrier()** – blokuje zmiany kolejności wykonywania instrukcji oraz cache'owanie podczas optymalizacji kompilatora

```
ThreadPool.QueueUserWorkItem(delegate
{
    while (true)
    {
        Thread.MemoryBarrier();
        a++;
        Thread.MemoryBarrier();
        a--;
    }
});

ThreadPool.QueueUserWorkItem(delegate
{
    while (true)
    {
        Thread.MemoryBarrier();
        a++;
        Thread.MemoryBarrier();
        a--;
    }
});
```

Niestety ...

```
0
474013
1138278
1868193
2592192
3275731
```

## Instrukcje atomowe

```
class MojaZWatkiem
{
    public int a;

    public void Watkowa()
    {
        while (true)
        {
            Interlocked.Increment(ref a);
            Thread.Sleep(1000);
        }
    }
}
```

•Add  
•CompareExchange  
•Decrement  
•Exchange  
•Increment  
•Read

Narzut operacji – 10ns

## Delegaty

```
class KlasaMoja
{
    public int a, b;
    public void UstawA(int x)
    {
        a = x;
    }
    public void UstawB(int x)
    {
        b = x;
    }
}

public delegate void DelNowaWartosc(int w);
static void Main(string[] args)
{
    KlasaMoja km = new KlasaMoja();
    DelNowaWartosc delegatka = null;
    delegatka += km.UstawA;
    delegatka += km.UstawB;
    delegatka(100);
    Console.WriteLine("„+km.a+”, „+km.b);
    Console.ReadKey();
}
```

Bezpieczeństwo – zastępują wskaźniki do funkcji

100, 100

## Wątek z delegaty bez notacji lambda

```
Thread[] threads = new Thread[numProcs];
for (int p = 0; p < numProcs; p++)
{
    double start = p * range + inclusiveLowerBound;
    double end = (p == numProcs - 1) ?
        exclusiveUpperBound : start + range;

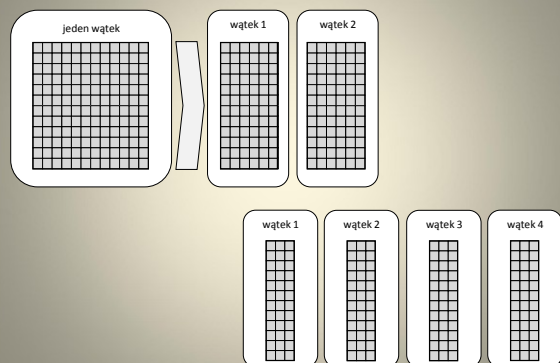
    threads[p] = new Thread(delegate()
    {
    })
}
```

## Wątek z delegaty - pula

```
using (ManualResetEvent mre = new ManualResetEvent(false))
{
    // Create each of the threads.
    for (int p = 0; p < numProcs; p++)
    {
        double start = p * range + inclusiveLowerBound;
        double end = (p == numProcs - 1) ?
            exclusiveUpperBound : start + range;

        ThreadPool.QueueUserWorkItem(delegate
        {
        })
    }
}
```

## Zrównoleganie



## Zrównoleglanie (1)

```
int inclusiveLowerBound = 0;
int exclusiveUpperBound = _do_ilu_liczymy_w_tabeli;
int size = exclusiveUpperBound - inclusiveLowerBound;
int numProcs = Environment.ProcessorCount; // czy wszystkie CPU?
int range = size / numProcs;
int remaining = numProcs;
Thread[] threads = new Thread[numProcs];
for (int p = 0; p < numProcs; p++)
{
    int start = p * range + inclusiveLowerBound;
    int end = (p == numProcs - 1) ? exclusiveUpperBound : start + range;

    threads[p] = new Thread(delegate()
    {
        int i, j; // ZMIENNE LOKALNE DLA KAZDEGO WATKU !!!

        for (i = start; i < end; i++)
            for (j = 0; j < _y; j++)
                dane[i, j] = 1;
    });
}
int pp;
for(pp = 0; pp < numProcs; pp++)
    threads[pp].Start();
for(pp = 0; pp < numProcs; pp++)
    threads[pp].Join();
```

## Zrównoleglanie (2)

```
int inclusiveLowerBound = 0;
int exclusiveUpperBound = _do_ilu_liczymy_w_tabeli;
int size = exclusiveUpperBound - inclusiveLowerBound;
int numProcs = Environment.ProcessorCount; // czy wszystkie CPU?
int range = size / numProcs;
int remaining = numProcs;

using (ManualResetEvent mre = new ManualResetEvent(false))
{
    for (int p = 0; p < numProcs; p++)
    {
        int start = p * range + inclusiveLowerBound;
        int end = (p == numProcs - 1) ? exclusiveUpperBound : start + range;

        ThreadPool.QueueUserWorkItem(delegate
        {
            int i, j; // ZMIENNE LOKALNE DLA KAZDEGO WATKU !!!

            for (i = start; i < end; i++)
                for (j = 0; j < _y; j++)
                    dane[i, j] = 1;

            if (Interlocked.Decrement(ref remaining) == 0)
                mre.Set();
        });
    }
    mre.WaitOne(); // oczekiwanie na zakonczenie ostatniego watku
}
```

## Zrównoleglanie (3)

```
int exclusiveUpperBound = rx;
int inclusiveLowerBound = 0;
int numProcs = Environment.ProcessorCount;
int nextIteration = inclusiveLowerBound;
int remaining = numProcs;

using (ManualResetEvent mre = new ManualResetEvent(false))
{
    for (int p = 0; p < numProcs; p++)
    {
        ThreadPool.QueueUserWorkItem(delegate
        {
            int i, j;

            while( (i = Interlocked.Increment(ref nextIteration) - 1)
                < exclusiveUpperBound )
            {
                for (j = 0; j < _y; j++)
                    dane[i, j] = 1;
            }
            if (Interlocked.Decrement(ref remaining) == 0)
                mre.Set();
        });
    }
    mre.WaitOne(); // oczekiwanie na zakonczenie ostatniego watku
}
```

## Zrównoleglanie (A)

```
using (ManualResetEvent mre = new ManualResetEvent(false))
{
    for (int p = 0; p < numProcs; p++)
    {
        ThreadPool.QueueUserWorkItem(delegate
        {
            int i, j, k;

            while ((i = Interlocked.Increment(ref nextIteration) - 1)
                < exclusiveUpperBound)
            {
                for (j = 0; j < n; j++)
                {
                    xc[i, j] = 0;
                    for (k = 0; k < m; k++)
                        xc[i, j] += xa[i, k] * xb[k, j];
                }
            }
            if (Interlocked.Decrement(ref remaining) == 0)
                mre.Set();
        });
    }
    mre.WaitOne(); // Wait for all threads to complete.
}
```

## Zrównoleglanie (B)

```
Parallel.For(0, m, i =>
{
    int j, k;
    for (j = 0; j < n; j++)
    {
        xc[i, j] = 0;
        for (k = 0; k < m; k++)
            xc[i, j] += xa[i, k] * xb[k, j];
    }
});
```

Toub, S.: Patterns of Parallel Programming. Understanding and Applying Parallel Patterns with the .Net Framework 4 and Visual Csharp. Parallel Computing Platform Microsoft Corporation. Version February 16, 2010.