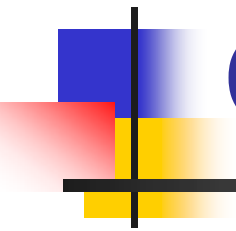
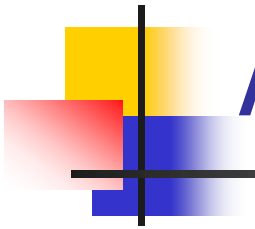


C# - innowacja czy plagiat?

Piotr Kowalski

Seminarium magisterskie „Zagadnienia programowania obiektowego”

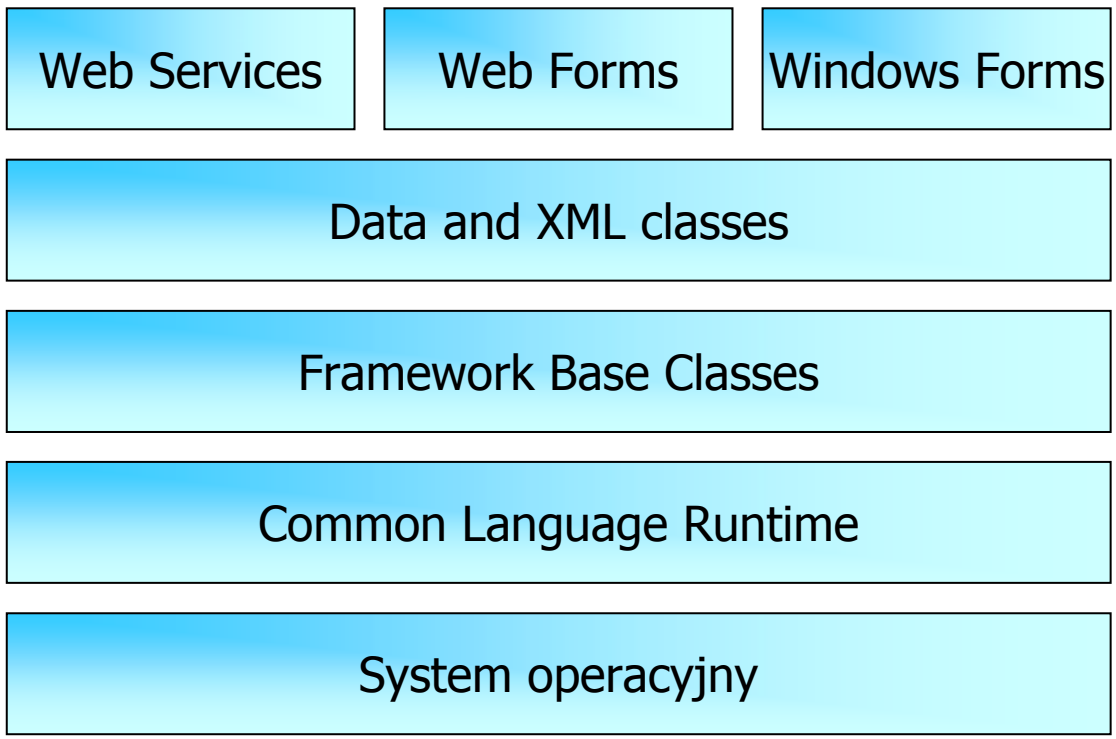
Wydział Matematyki, Mechaniki i Informatyki
Uniwersytet Warszawski



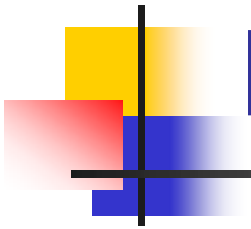
Agenda

- Wprowadzenie do platformy .NET
- Język C#
- Wnioski

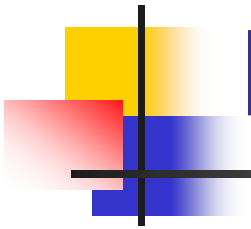
Platforma .NET



Nie tylko Windows

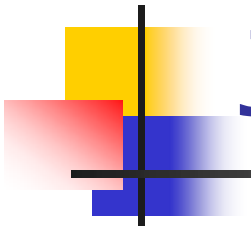


- Project Mono i DotGNU – open source CLR dla systemów Unixowych
- Mono 1.0 spodziewane na przełomie 2003/2004
- Wsparcie od HP i Intela
- Plany Microsoftu – licencja *„shared source”*



MSIL

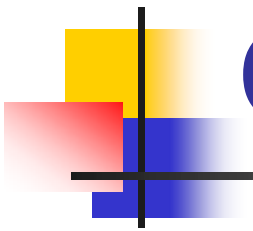
- Microsoft Intermediate Language
- Wykonywany przez maszynę CLR
- Mechanizm analogiczny do JVM
- Kompilator Just-In-Time na żądanie



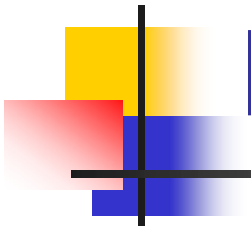
Języki .NET

- Czyli języki programowania, które „tłumaczą” się do MSIL
  - C#
  - C++ i zarządzany C++
  - Visual Basic
  - Java (także sam bajtkod)
  - Wsparcie dla innych języków powstaje w ramach m.in. projektów open source

Okoliczności powstania C#



- Kierownictwo prac: Anders Hejlsberg i Scott Wiltamuth
- Założenia
  - Efektywność C
  - Obiektowość C++
  - Bezpieczeństwo Javy
  - Łatwość programowania Visual Basic'a
- Pierwsza odsłona: lipiec 2000



Podstawowe byty C#

- Odpowiedniość z typami CTS
  - Klasy
  - Wyjątki
  - Interfejsy
  - Struktury
  - Typ wyliczeniowy

Typy C# a CTS

- Boolean
- Byte
- SByte *
- Char
- Decimal
- Double
- Single
- Int16
- UInt16 *
- Int32
- UInt32 *
- Int64
- UInt64 *
- Object
- String

Wszystko jest obiektem

```
int n = 32;
string Tekst = n.ToString();

string InnyTekst = 32.ToString();
```

Każdy typ danych dziedziczy automatycznie z **System.Object**.

Zarządzanie typami

- **is** – sprawdza, czy obiekt jest danego typu

```
bool ToJestPies = (piesek is Pies);
```

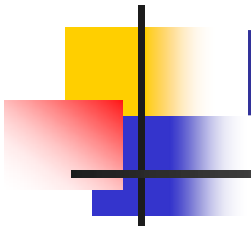
- **typeof** – zwraca typ obiektu (jako obiekt System.Type)

```
Type TypObiektu = typeof(piesek)
```

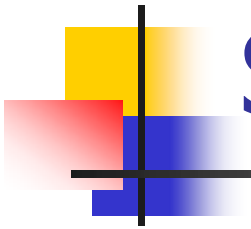
- **sizeof** – zwraca rozmiar danego typu (dostępne tylko w kodzie potencjalnie niebezpiecznym)

```
int rozmiar = sizeof(Pies)
```

Klasy w C#

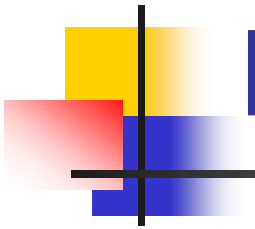


- Nowy rodzaj klasy: **sealed** (zapieczętowana), dziedziczenie z takich klas jest zakazane; można także zakazywać przedefiniowywania poszczególnych metod przy dziedziczeniu
- Słowo kluczowe **base** analogiczne do **this**, odnoszące się do nadrzędnej klasy
- Klasy można zagnieżdżać



Składowe klas

- Pola (fields)
- Stałe
- Metody
- Konstruktory
- Destruktory
- Właściwości (properties)
- Operatory
- Zdarzenia (events)
- Indeksery (indexers)



Pola

- Nowy modyfikator: **readonly**

  public **readonly** int n = 3;

  wartość można ustawić w konstruktorze lub przy deklaracji (**const**: tylko przy deklaracji)

- Brak automatycznej synchronizacji pól zadeklarowanych jako **static**

Sprytne pola (SmartFields)

- Automatyczne upublicznianie prywatnych składowych ze sprawdzaniem poprawności

```
public int x {
  get {
        return _x;
  }
  set {
        TestValid(value);
        _x = value;
} }
```

Metody

- Niedozwolone parametry domyślne
- Przekazywanie parametrów jako **ref** i **out** (konieczne także przy wywołaniu)

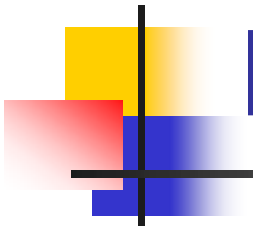
```
public static void GetVal(out int p) {
   p = 5;
}
...
   int q;
   GetVal(out q);
```

Metody c.d.

- Zmienna liczba parametrów metody: słowo kluczowe **params**

- Może być tylko jedno jako ostatnie na liście parametrów, związane z tablicą

```
public int Print(params int[] Lista) {
  foreach(int v in Lista) …
}
```

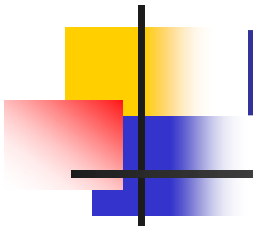


Konstruktory

- Konstruktory instancyjne
  Standardowo wywoływany domyślny konstruktor klasy nadrzędnej, inne wywołania trzeba wpisać explicite

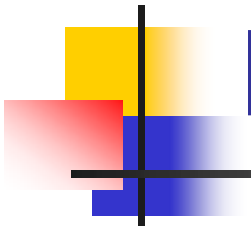
- Konstruktory klasowe (statyczne)
  Deklarowane za pomocą słówka **static**

Konstruktory c.d.



- Kolejność wołania konstruktorów:
  - Inicjalizacja pól statycznych klasy
  - Wywołanie statycznego konstruktora
  - Inicjalizacja pól obiektu
  - Wywołanie konstruktora dla obiektu

Dziedziczenie



- Słowa **virtual** i **override** – mechanizm wiązania dynamicznego z C++

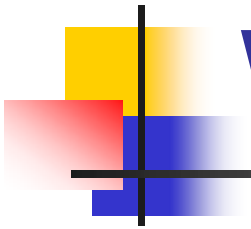
```
override public void Info() {
   System.Console.WriteLine(”Tekst”)
}
```

Dziedziczenie c.d.

- Słowo **new** dla metod – wyłączenie metody z poszukiwań wiązania dynamicznego

```
new public int Value() {
   return 5;
}
```

- Różnice przy wołaniu z klasy bazowej i z docelowej



Widoczność

- public
- protected – na poziomie klasy
- internal – na poziomie pakietu (assembly)
- protected internal
- private

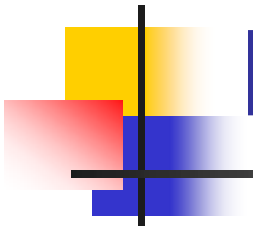
- Nie ma mechanizmu zaprzyjaźniania (friend)

Boxing i unboxing

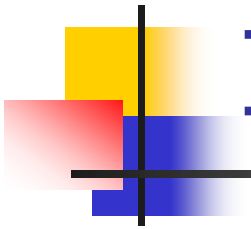
- Opakowywanie i rozpakowywanie typów prostych z obiektów

```
int i = 123;
object o = i; // boxing
int j = (int)o; // unboxing (explicite)
Console.WriteLine("j: {0}", j); // boxing
```

Przestrzenie nazw

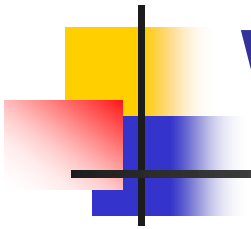


- Deklarowane jak klasy
- Mechanizm zbliżony do pakietów Javy (m.in. identyczne zasady nazewnictwa)
- Słowo kluczowe **using**
- Współpraca przestrzeni nazw i pakietów .NET (assemblies), dołączanie powiązań



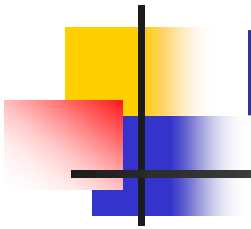
Interfejsy

- Klasy całkowicie abstrakcyjne, definiowane za pomocą słowa **interface**
- Definiuje tylko nagłówki metod
- Możliwe implementowanie wielu interfejsów przez jedną klasę (niemożliwe wielodziedziczenie)



Wyjątki

- Mechanizm try-catch-finally
- Wyjątki dziedziczą z klasy System.Exception
- Nie trzeba dopisywać słówka **throws** do nagłówków metod
- Dodatkowy atrybut InnerException

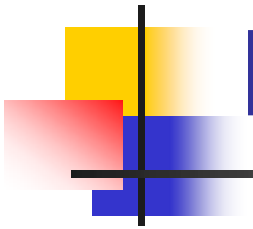


Rzutowanie

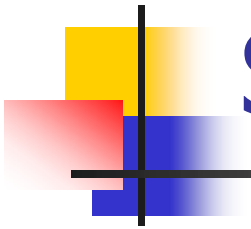
- Mechanizm oparty na C++
- Słówko **as**, które próbuje dokonać konwersji nie rzucając wyjątku w przypadku błędu (zamiast tego zwraca null)

```
Pies pies;
Zwierze zwierze = ...;
pies = zwierze as Pies;
```

Rzutowanie c.d.



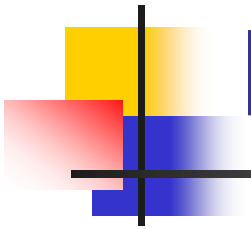
- Klasa System.Convert udostępnia szereg metod do konwersji między typami standardowymi
- Można przedefiniowywać rzutowania jak w C++ (jako operatory)



Struktury

- Struktury mogą dziedziczyć tylko z interfejsów
- Można zdefiniować konstruktor dla struktury
- Domyślna widoczność – private
- Mniej wymagające pamięciowo od klas (przechowywane na stosie); wada – trzeba ręcznie przekazywać referencje

Błędy arytmetyczne



- Słowa kluczowe **checked** i **unchecked** pozwalają zaznaczyć, które fragmenty kodu powinny wykonywać się z włączona kontrolą błędów arytmetycznych, a które nie
- W przypadku braku tych klauzul efekt zależy od ustawień kompilatora

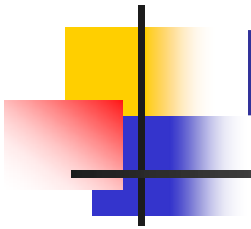
```
unchecked {
        a = b/c; // jeśli wiemy, ze c != 0
}
```

- Mechanizm poprawia wydajność kodu

Pętla foreach

- Dokonuje iteracji po kolekcji elementów (np. tablicy)

```
foreach(int i in Tablica) {
  // i jest tylko do odczytu
  Console.WriteLine(”{0}”, i);
}
```

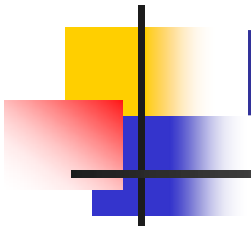


Delegaci

- Zastępują wskaźniki do funkcji

```
// najpierw trzeba zadeklarować typ
public delegate void Fun(int arg1, int arg2);
...
public void Work(Fun f) {
   f(5,6);
}
```

Delegaci c.d.



- Aby zdefiniować delegata piszemy metodę zgodną z jego definicją, np.

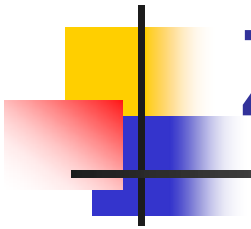
```
public void Funkcja(int arg1, int arg2) {
    Console.WriteLine(„{0},{1}”, arg1, arg2);
 }
```

- A następnie w kodzie tworzymy instancję:

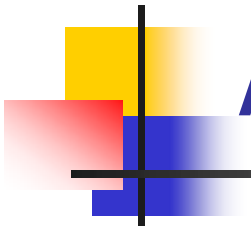
```
Fun f = new Fun(Funkcja);
```

- Możliwość łączenia delegatów w łańcuchy

Zdarzenia (events)

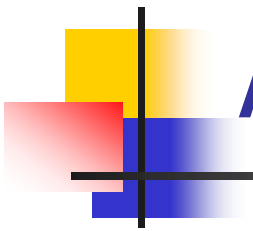


- Za pomocą mechanizmu delegatów można implementować mechanizm publish-subscribe dzięki możliwości łączenia delegatów w łańcuchy
- W przypadku wystąpienia zdarzenia, klasa za nie odpowiadająca wywołuje łańcuch delegatów, z których każdy reprezentuje jednego subscribera
- Słowo kluczowe **event**



Atrybuty

- Meta-dane trzymane razem z kodem, np. klucze publiczne, adresy WWW, informacje o autorze, licencji
- Wykorzystując mechanizm reflection można uzyskać dostęp do przechowywanych informacji
- Bogaty zbiór gotowych atrybutów, możliwość definiowania własnych
- Rozbudowany mechanizm kontroli wartości atrybutów wbudowany w CLR i IDE



Atrybuty c.d.

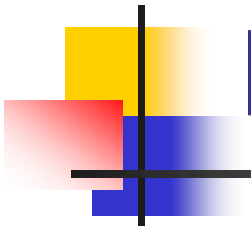
```
[BugFix(123, „Jan Suzin”, „20/03/2002”,
   Comment=„Usunięty deadlock”)]
protected int GetValue()
{
...
}
```

Indeksery (indexers)

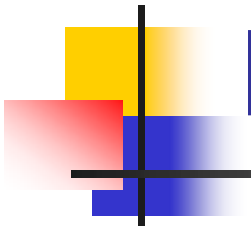
- Rozszerzenie mechanizmu przeładowywania operatora []

```
public object this[int n] {
   get { return items[n]; }
   set { items[n] = value; }
}
```

Kod niebezpieczny (unsafe)



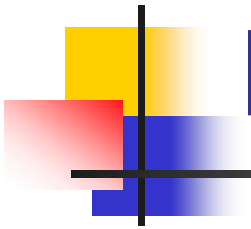
- C# daje możliwość pisania niskopoziomowego i niezarządzanego kodu
- Użycie klauzuli **unsafe** przy klasie, strukturze, interfejsie lub metodzie daje dostęp do operatorów **sizeof**, **&**, **->** oraz *****
- Słowo kluczowe **fixed** powoduje, że blok pamięci zarządzanej przestaje być poddawany odśmiecaniu i relokacji



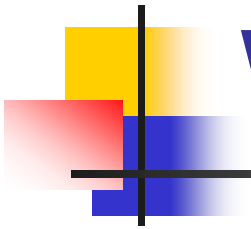
Drobnostki

- case może porównywać typ string
- nie można przykrywać zmiennych
- eliminacja domyślnej konwersji int -> bool

Framework Class Library



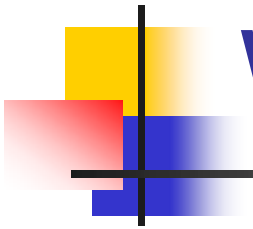
- Zbiór „użytecznych” klas
- Zawiera podstawowe struktury danych, klasy do operacji wejścia/wyjścia (pliki, sieć), dostępu do danych (XML), wielowątkowości
- Wsparcie dla wyrażeń regularnych
- Obecny mechanizm reflection
- Windows Forms jako platforma tworzenia interfejsu użytkownika



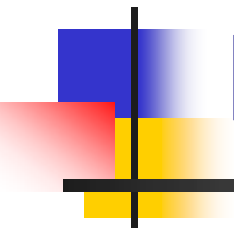
Wnioski

- Wiele „drobnych” innowacji do języka podyktowanych względami praktycznymi
- Pozostawienie programiście szerokiego wachlarza dostępnych środków
- Dodatkowe obostrzenia składni w celu wyeliminowania typowych błędów

Wnioski c.d.

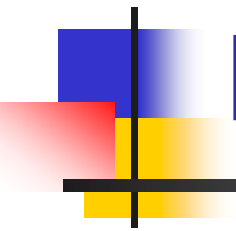


- Język nastawiony na jak najefektywniejsze tworzenie kodu
- Rezygnacja z „czystości” języka na rzecz jego użyteczności
- Integralna część platformy .NET



Moja opinia

Ani rewolucja, ani plagiat. Język o szerokich zastosowaniach dobrze wpasowujący się w obecne praktyczne potrzeby programistów.



Koniec

Dziękuję!