POLITECHNIKA POZNAŃSKA
Poznan University of Technology

Skrypt do zajęć: **Modelowanie i Programowanie Obiektowe**
Podstawy języka C#

Andrzej Stroiński 14.10.2011

Zajęcia laboratoryjne przeprowadzane są na komputerach z systemem operacyjnym Windows 7 z wykorzystaniem oprogramowania Visual Studio 2010 w wersji Ultimate. Poniżej omówiono kilka często wykorzystywanych skrótów:
VS – Visual Studio
LPM – Lewy przycisk myszy

PPM – Prawy przycisk myszy

Zagadnienie 1: C# bez Visual Studio.

1. Stwórz plik z kodem źródłowym o nazwie HelloWorld.cs.
2. Zaimplementuj prosty program wypisujący "HelloWorld!" na konsoli. Wykorzystaj informacje z poprzedniego skryptu: *MiPO - Visual Studio 2010.pdf*
3. Skompiluj program wykorzystując polecenie konsoli:
   `>csc.exe .\HelloWorld.cs`
4. Uruchom program:
   `>.\HelloWorld.exe`
5. W celu kompilacji z wykorzystaniem zewnętrznych bibliotek:
   `>csc /r:System.DLL HelloWorld.exe`

Zagadnienie 2: Pierwsza aplikacja
Poniżej zamieszczono kod prostej aplikacji konsolowej, która wyświetla na standardowym wyjściu napis "Hello World". Zwróć uwagę, że aplikacja oczekuje na reakcję użytkownika przed zakończeniem działania.

```
// słowo kluczowe "using" importuje tzw. przestrzenie nazw (ang. "namespace"), które są
// zbiorami klas.
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

// stworzenie nowej przestrzeni nazw w pisanym programie, w której znajdować się
// będzie klasa Program z metodą Main.
namespace MiPO___lab1
```

```
{
    // definicja klasy. Program może składać się z wielu klas zawierających: metody,
pola,
    // właściwości.
    class Program
    {
        // statyczna metoda "główna" programu pozwalająca na
        // pobranie parametrów z linii poleceń
        static void Main(string[] args)
        {
            // Wypisanie na standardowe wyjście tekstu: "Hello, world!"
            Console.WriteLine("Hello, world!");
            // Wczytanie linii z konsoli.
            Console.ReadLine();
        }
    }
}
```

Zagadnienie 3: Typy danych i zmienne

Języki programowania dzieli się na języki typowane silnie, słabo oraz dynamicznie. Język C# jest językiem silnie typowanym co oznacza, że każdą zmienną należy przed jej użyciem zadeklarować i określić jej typ. Zmienna podczas działania programu nie może zmienić swojego typu. Języki słabo typowane dopuszczają zmianę typu w zależności od kontekstu w jakim zmienna występuje, natomiast języki dynamicznie typowane nie wymagają definicji zmiennej, gdyż jej typ zależy od aktualnej wartości. Typy w języku C# dzieli się na tzw. wbudowane (*built-in* – `char`, `int`, `float`...) oraz zdefiniowane przez użytkownika (*user-defined* – `class`, `interface`).
Najważniejsze typy predefiniowane to:

- `bool`: może przyjmować dwie wartości prawda lub fałsz.
- `int`: zmienna całkowitoliczbowa
- `float`: zmienna zmiennoprzecinkowa
- `string`: ciąg znaków
- `char`: pojedynczy znak

Innym podziałem typów jest:

- typy przekazywane przez wartość – zmienne tego typu przechowują wartość. W momencie podstawienia: `zmienna1 = zmienna2;` następuje kopiowanie wartości `zmiennej2` do `zmiennej1`. Przykładowe typy: `int`, `float`, `bool` ...
- typy referencyjne – zmienne typu referencyjnego przechowują referencje do obiektów (instancji konkretnych klas). Referencja zawiera adres, który umożliwia pobranie wartości zmiennej. W przypadku podstawienia `zmienna1 = zmienna2;` gdy `zmienna2` oraz `zmienna1` są typami referencyjnymi obie zmienne wskazują na ten sam obiekt (kopiowany jest do `zmienna1` adres obiektu z `zmienna2`). Słowa kluczowe wykorzystywane do deklaracji zmiennych: `class`, `interface`, `delegate`. Zmienne wbudowane typu referencyjnego to: `object`, `string`.

Zmienna to kontener, który może przechowywać wartości lub obiekty. Deklaracja zmiennej ma następującą postać:

```
<zakres_widoczności> <typ> <nazwa_zmiennej>
```
lub
```
<typ> <nazwa_zmiennej> = <wartość>
```

Przykład deklaracji kilku zmiennych (`<zakres_widoczności>` można pominąć):
```
int calkowita;
string znaki;
float fl;
int calkowita2 = 5;
string znaki2 = "student";
float fl2 = 10.5;
```

1. Zmodyfikuj kod aplikacji HelloWorld, aby pobierała jako parametr numer albumu studenta, wypisywała go na standardowym wyjściu.
   ```
   Console.WriteLine("Witaj studencie o albumie: " + args[0]); // operator +
                                                                  pozwala na
                                                               "sklejanie" ciągów
                                                                  znaków
   ```
2. Zadeklaruj zmienne, które pozwolą na wykonanie sumy ich wartości
   ```
   int num1;
   int num2;
   int sum;
   ```
3. Wczytaj z konsoli wartości liczb do zsumowania wykorzystując funkcję przekształcającą ciąg znaków na liczbę całkowitoliczbową `int.Parse()`
   ```
   num1 = int.Parse(Console.ReadLine());
   num2 = int.Parse(Console.ReadLine());
   ```
4. Oblicz i wypisz na standardowym wyjściu sumę liczb całkowitych
   ```
   sum = num1 + num2;
   Console.WriteLine("Suma liczb = " + sum);
   ```

Zagadnienie 4: Tablice
Deklaracja zmiennych typu tablicowego na następującą postać:
```
int[] numbers;
```

Aby możliwe było korzystanie z tablicy należy ją zainicjalizować:
```
numbers = new int[10];
```

Uzupełnienie tablicy wartościami:
```
numbers[0] = 1;
numbers[1] = 21;
numbers[2] = 1;
```

Wypisanie wszystkich elementów tablicy:
```
foreach (int num in numbers)
      Console.WriteLine(num);
```

Zagadnienie 5: Funkcje warunkowe
Do sterowania przetwarzaniem w języku C#, podobnie jak ma to miejsce w każdym języku imperatywnym do sterowania wykonaniem programu wykorzystuje się dwie podstawowe konstrukcje: `if` oraz `switch`.

Pierwsza z nich ma następującą budowę:
```
if (<warunek_który_ma_być_spełniony>)
{
      <operacje_jeśli_warunek_prawdziwy>
} else
{
      <operacje_jeśli_warunek_fałszywy>
}
```
Warunkiem jest dowolne wyrażenie logiczne np:
```
arg1 == arg2    // arg1 równy arg2
arg1 > arg2     // arg1 większy arg2
arg1 <= arg2    // arg1 mniejszy lub równy arg2
arg1 != arg2    // arg1 nierówny arg2
```

Wyżej wymienione warunki mogą być połączone w ramach jednej konstrukcji warunkowej operatorami logicznymi "i" (&&) oraz "lub" (||).

1. Dodaj do aplikacji warunek, który pozwoli na wypisywanie na wyjściu jedynie sumy liczb zawierającej się w przedziale między 1, a 10.
   ```
   if (num1 > 1 && num2 < 10)
    {
          Console.WriteLine("Suma liczb = " + sum);
    } else
    {
          ;
    }
   ```
2. Dodaj warunek, aby wypisywana była suma liczb, nawet jeśli nie zawiera się w określonym przedziale, gdy numer albumu wynosi 99999.
   ```
   if ((num1 > 1 && num2 < 10) || args[0] == "99999")
   ```

Druga z konstrukcji (`switch`) jest równoważna kilku instrukcjom `if`. Jej podstawowym zadaniem jest określenie w czytelny sposób zachowania programu w zależności od wartości jaką przyjmuje zmienna warunkowa. Poniżej znajduje się przykład zastosowania:

```
int num = 1;
switch (num) // zmienna warunkowa
{
      case 0:
            Console.WriteLine("Liczba zero!");
            break;
      case 1:
            Console.WriteLine("Liczba jeden!");
            break;
}
```

Zagadnienie 6: Pętle
W języku C# udostępniono cztery konstrukcje realizujące pętle (wielokrotne wykonanie tego samego kodu).

Pętla `while`
Najprostsza pętla wykonująca się tak długo jak <warunek> jest spełniony:
```
while(<warunek>)
{
	<kod_iteracji_pętli>
}
```
Przykład:
```
int number = 10;
while(number > 0)
{
	Console.WriteLine("iteracja: " + number--);
}
```

Pętla `do`
Najprostsza pętla wykonująca się tak długo jak <warunek> jest spełniony, ale przynajmniej raz:
```
do
{
	<kod_iteracji_pętli>
} while (<warunek>);
```

Przykład:
```
int number = 10;
do
{
	Console.WriteLine("iteracja: " + number);
} while (number != 10);
```

Pętla `for`
Pętla wykonywana określoną liczbę razy:
```
for (<zmienna_iterator>; <warunek>; <krok>)
{
	<kod_iteracji_pętli>
}
```

Przykład:
```
int number;
for (number = 0; number < 10; number++)
{
	Console.WriteLine("Iteracja: " + number);
}
```

Pętla `for`
Pętla iterująca po wszystkich elementach `<kolekcja>`, dostęp do aktualnego elementu w każdej iteracji pętli wykonywany jest za pomocą zmiennej `<iterator>`:
```
foreach (<iterator> in <kolekcja>)
```

Przykład:
```
ArrayList list = new ArrayList();
list.Add("student 1");
```

```
list.Add("student 2");
list.Add("student 3");
foreach (string name in list)
     Console.WriteLine(name);
```

Zagadnienie 7: Funkcje i jej parametry

Funkcje służą enkapsulacji kawałku kodu, dzięki któremu można korzystać z raz napisanego kodu w wielu miejscach bez jego powtarzania. Zwiększa to przejrzystość oraz czytelność. Poniżej przedstawiono prototyp deklaracji funkcji:

```
<zakres_widoczności> <typ_zwracany> <nazwa_funkcji>(<opcjonalne_parametry>)
{
      <kod_funkcji>
}
```

Przykładem funkcji jest funkcja `Add`, która pobiera dwa parametry typu całkowitego (`int arg1, int arg2`), a następnie liczy ich sumę i zwraca w postaci liczby całkowitej (`int Add`):

```
public int Add(int arg1, int arg2)
{
      int result;
      result = arg1 + arg2;
      return result;
}
```

Wywołanie funkcji wymaga podania jej nazwy oraz parametrów:

```
int resultOfFunc = Add(1, 2);
```

W języku C# przekazywanie parametrów do funkcji może zostać zrealizowane na dwa sposoby:

- przez wartość (ang. by value) – przesyłanie kopii obiektu
- przez referencję (ang. by reference) – przekazywanie referencji do faktycznego obiektu

Domyślnie parametry do funkcji przekazywane są poprzez wartość (kopiowanie). Poniżej zamieszczony kod modyfikuje jedynie "lokalnie" w obrębie funkcji ciąg znaków, przez co po jej zakończeniu efekty jej działania nie są widoczne:

```
string s = "test";
Console.WriteLine(s);
AddToStr(s);
Console.WriteLine(s);
Console.ReadLine();
...
public static void AddToStr(string arg)
{
      arg = arg + "a";
      Console.WriteLine("Wew: " + arg);
}
```

Modyfikator - ref

Aby przekazać referencje (obiekt właściwy) parametru do funkcji należy użyć słowa kluczowego ref w definicji funkcji. Wówczas zmiany wykonane wewnątrz funkcji są widzialne po jej

zakończeniu, gdyż funkcja operuje na obiekcie właściwym a nie na jego kopii:

```
...
AddToStr(ref s);
...
public static void AddToStr(ref string arg)
{
      ...
}
```

Modyfikator - out

Modyfikator out, również służy do przekazywania parametrów do funkcji za pomocą referencji, ale umożliwia wywołanie funkcji z niezainicjalizowaną wartością parametru. Funkcja wówczas musi tą wartość zainicjalizować:

```
...
string s;
SetString(out s);
Console.WriteLine(s);
Console.ReadLine();

...
static void SetString(out string arg)
{
      arg = "test";
}
```

Modyfikator - param

To słowo kluczowe pozwala na zdefiniowanie funkcji o dowolnej liczbie parametrów danego typu np:

```
...
PrintNumbers(1,2,3,4);
...
static void PrintNumbers(params int[] nums)
{
      foreach (int num in nums)
            Console.WriteLine("" + num);
}
```

Bibliografia:

[1] Solutions as Containers: http://msdn.microsoft.com/en-us/library/df8st53z(v=VS.80).aspx
[2] Solutions, Projects and Items:http://msdn.microsoft.com/en-us/library/b142f8e7(v=VS.80).aspx
[3] MSDN: http://msdn.microsoft.com/
[4] MSDN: http://msdn.microsoft.com/
[5] C# Practical Learning: http://www.functionx.com/csharp/
[6] C# for beginners: http://www.csharphelp.com/2006/12/c-tutorial-for-beginners/
[7] C# tutorial: http://csharpcomputing.com/Tutorials/
[8] C# tutorials: http://csharp.net-tutorials.com/