Programowanie obiektowe w C# - skrypt:

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: Klasy i obiekty
Klasa to zbiór powiązanych ze sobą funkcji (metod) oraz zmiennych (nazywanych wówczas polami). Klasa definiuje wszystkie swoje elementy, czyli określa jak jej metody działają oraz co oznaczają poszczególne jej pola. W celu korzystania z klasy należy stworzyć jej instancję zwaną obiektem. Klasa jest jedynie definicją, natomiast obiekt jest fizycznym jej przedstawicielem w pamięci. Wszelkie operacje wykonywane są na obiektach. Należy zaznaczyć, że można stworzyć wiele obiektów będących instancjami tej samej klasy. Wówczas każdy z tych obiektów posiada swoje pola oraz metody (wiele obiektów jednej klasy może mieć te same pola przechowujące inne wartości).

Poniżej przedstawiono sposób deklaracji klasy:
```
<zakres_widoczności> class <nazwa_klasy>
{
...
<definicje_zmiennych_oraz_metod>
...
}
```

Przykład:
```
class Plane
{
	...
	public void Print()
	{
		Console.WriteLine("Plane");
	}
}
```

Stworzenie obiektu klasy:
```
Plane o1 = new Plane();
```

Wywołanie metody klasy:
```
o1.Print();
```

Konstruktor klasy – to specjalna metoda, która nie zwraca żadnego parametru. Określa ona jakie czynności zostaną wykonane w momencie tworzenia obiektu klasy (np. inicjalizuje zmienne – pola obiektu):
```
class Plane
{
	string color;
	public Plane(string col)
	{
```

```
            color = col;
        }
        public void Print()
        {
            Console.WriteLine("Plane is " + color);
        }
}
...
Plane o1 = new Plane("green");
o1.Print();
```

Destruktor – metoda pozwalająca na określenie czynności wykonywanych podczas usuwania obiektu z pamięci. Język C# jest językiem zarządzanym co oznacza, że platforma .NET dba o usuwanie z pamięci niepotrzebnych obiektów (już nie używanych):

```
class Plane
{
        ...
        ~Plane()
        {
            Console.WriteLine("Cleaned");
        }
}
...
Plane o1 = new Plane("green");
o1.Print();
o1 = null;
```

Zagadnienie 2: Właściwości

Właściwości klasy, służą do zarządzania polami (zmiennymi) definiowanej klasy. Definiują one sposób dostępu do zmiennych obiektu z zewnątrz. Poniżej przedstawiono przykład właściwości (sekcja set – dotyczy ustawiania wartości; get – dotyczy pobierania wartości; value – specjalna zmienna zawierająca wartość przypisywaną do właściwości):

```
class Plane
{
        ...
        string color;
        public string Color
        {
            get
            {
                return color.ToUpper();
            }
            set
            {
                if (value == "green" || value == "red")
                    color = value;
                else
                    Console.WriteLine("Plane must be green or red!");
            }
        }
        ...
}
...
Plane o1 = new Plane("green");
```

```
o1.Print();
o1.Color = "yellow";
o1.Color = "red";
o1.Print();
Console.ReadLine();
```

Zagadnienie 3: Przeciążanie funkcji

Jeśli, programista chce stworzyć funkcję, która przyjmuje w zależności od potrzeby różną liczbę parametrów (lub różne typy parametrów) istnieje metoda tzw. Przeciążania definicji funkcji. Polega ona na definiowaniu kilku funkcji o tej samej nazwie posiadających różne parametry lub różną ich liczbę. Środowisko języka C# samo na podstawie kontekstu wywołania funkcji zadecyduje, którą wersję funkcji, należy wykonać:

```
static void Add(int p1, int p2)
{
	Console.WriteLine(p1+p2);
}
static void Add(string p1, string p2)
{
	Console.WriteLine(p1 + p2);
}
static void Add(int p1, int p2, int p3)
{
	Console.WriteLine(p1 + p2 + p3);
}
...
Add(1, 2);
Add("aaa", "bbb");
Add(1,2,3);
...
```

Należy pamiętać, że tak jak każdą funkcję, również konstruktory obiektów można przeciążać.

Zagadnienie 4: Zakresy widzialności

Zakresy widzialności określają, jaki zasięg mają definiowane zmienne, metody oraz klasy. W ogólności, zakres widzialności definiuje z jakiego miejsca w kodzie do określonych zmiennych czy obiektów będzie można się odwołać.

Zakres widzialności określa się podczas definicji. Przykład:

```
public int num;
public static void Add(int p1, int p2)...
public class Plane
{...}
```

Dostępne zakresy widoczności:
public – metody, zmienne oraz klasa są dostępne z dowolnego miejsca w kodzie
protected - metody, zmienne oraz klasa dostępne są tylko z kodu klasy lub klasy po niej dziedziczącej (dziedziczenie omówione jest dalej)
internal - metody, zmienne oraz klasa dostępne są tylko z kodu tego samego projektu
protected internal – tak samo jak internal, z wyjątkiem klas dziedziczących które mają dostęp nawet z innych projektów.
private – dostęp tylko z aktualnej klasy

Zagadnienie 5: Klasy, pola i metody statyczne
Czasami przydatne może być używanie klasy, bądź jej elementów (pola, metody) bez tworzenia jej instancji. Służy do tego słowo kluczowe static. Nie można tworzyć obiektów klas statycznych, zatem służą one głównie grupowaniu zmiennych oraz metod. Metody statyczne oraz pola statyczne, można wykorzystywać, bez tworzenia obiektów danej klasy. Istotnym jest, że pole statyczne danej klasy jest tylko jedno, i dostępne poprzez klasę, a nie za pomocą referencji do obiektu.

Klasy niestatyczne mogą posiadać metody i pola statyczne, natomiast klasy statyczne niemogą mieć pól i metod niestatycznych.

Przykład:
```
public static class Calculator
{
	public static string name = "nazwa";

	public static int Add(int param1, int param2)
	{
		return param1 + param2;
	}
}
...
Console.WriteLine(Calculator.Add(1, 2));
Console.ReadLine();
```

Zagadnienie 6: Dziedziczenie
Kluczowym aspektem programowania obiektowego jest mechanizm zwany dziedziczeniem. Polega on na tym, że podczas definiowania nowej klasy (typu), można wskazać klasę nadrzędną (rodzica) po której nowotworzona klasa ma "dziedziczyć" pewne własności, metody oraz pola. Elementy, które będą przeniesione do klasy podrzędnej określają zakresy widzialności. Poniżej zamieszczono przykład:

```
public class Animal
{
	public void Describe()
	{
		Console.WriteLine("To jest zwierze!");
	}
}

public class Hamster : Animal
{

}
```

W przykładzie zdefiniowano klasę `Animal`, która posiada jedną publiczną metodę `Describe`. Metoda ta wypisuje na standardowe wyjście tekst `"To jest zwierze!"`. Druga zdefiniowana klasa to `Hamster` i dziedziczy ona po klasie Animal (dziedziczenie ":"). Oznacza to, że metoda `Describe` dostępna będzie również w klasie `Hamster`.

Jeśli do klasy `Hamster` doda się następujące linie:
```
public override void Describe()
{
```

```
        Console.WriteLine("To jest chomik!");
}
```

Spowoduje to, że w nowej klasie zostanie nadpisana metoda klasy nadrzędnej `Describe` jej nową wersją (słowo kluczowe `override`). Ponadto, należy zmienić definicję metody w klasie nadrzędnej dodając słowo kluczowe `virtual` (metody nieoznaczone jako wirtualne nie mogą zostać przedefiniowane w klasie podrzędnej).

```
public virtual void Describe()
{
        Console.WriteLine("To jest zwierze!");
}
```

Teraz wywołanie metody `Describe` klasy `Hamster` spowoduje wypisanie `"To jest chomik!"`. Aby wywołać metodę klasy nadrzędnej `Animal` można wykorzystać słowo kluczowe `base`:

```
public class Hamster : Animal
{
        public override void Describe()
        {
                base.Describe();
                Console.WriteLine("To jest chomik!");
        }
}
```

Linki:

[1] MSDN: http://msdn.microsoft.com/
[2] C# Practical Learning: http://www.functionx.com/csharp/
[3] C# for beginners: http://www.csharphelp.com/2006/12/c-tutorial-for-beginners/
[4] C# tutorial: http://csharpcomputing.com/Tutorials/
[4] C# tutorials: http://csharp.net-tutorials.com/