Języki i paradygmaty programowania imperatywnego

Wykład II

Paradygmat obiektowy

- Paradygmat programowania obiektowego zakłada tworzenia programów z wykorzystaniem obiektów - elementów łączących stan (czyli dane) i zachowanie (czyli procedury, metody)
- Obiektowy program komputerowy wyrażony jest jako zbiór takich obiektów, komunikujących się pomiędzy sobą w celu wykonywania zadań

Paradygmat obiektowy

Cechy języków programowania, które czynią je obiektowymi:

- Abstrakcja
- Hermetyzacja
- Polimorfizm
- Dziedziczenie

Paradygmat obiektowy

Klasa jest to definicja pewnego rodzaju bytów. Bytów które posiadają określone cechy, zachowanie i umiejętności

Pragmatycznie - klasa to pewien nazwany zbiór metod i zmiennych

Klasa jest definicją bytu, albo mówiąc inaczej typem bytu, ale dopiero instancja klasy, tj. obiekt, odzwierciedla konkretny byt

Paradygmat obiektowy

Cechy obiektu:

- tożsamość (nazwa) - umożliwia identyfikację i odróżnienie od innych obiektów
- stan – określany poprzez aktualną wartość danych składowych
- zachowanie – określane poprzez zestaw metod wykonujących operacje na danych składowych

Klasy i obiekty w języku C#

Definicja klasy

[atrybuty] [modyfikatory]
class identyfikatorKlasy [:lista klas bazowych]
{
    ciało klasy
}[;]

Klasy i obiekty w języku C#

Klas w technologii .NET dziedziczy po wspólnej klasie bazowej **System.Object**

Dziedziczenie to ma charakter domyślny i niejawny (nie wymaga umieszczenia w definicji klasy)

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

namespace ConsoleApplication12
{
   class Program
   {
      static void Main(string[] args)
      {
         klasaTestowa k1 = new klasaTestowa();
         klasaTestowa k2 = new klasaTestowa();

         Console.WriteLine(k1.ToString());
         Console.WriteLine(k1.GetType());
         Console.WriteLine(k1.Equals(k2));
         Console.WriteLine(k1.GetHashCode());
      }
   }
}
```

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

namespace ConsoleApplication12
{
   class klasaTestowa
   {

   }
}
```

Przykład

Przykład 2-1.1

Visual Studio 2-1.1

Klasy i obiekty w języku C#

abstract – określa że dana klasa ma być wykorzystywana tylko w roli klasy bazowej dla innych klas

sealed – określa że klasa nie może być przedmiotem dziedziczenia

static – określa że dana klasa zawiera wyłącznie składowe statyczne (.NET 2.0)

Przykład

Przykład 2-1

Visual Studio 2-1

Składowe kasy

Rodzaj składowej	Miejsce stosowania	Opis
Stała	klasa, struktura	Symbol reprezentujący niezmienną wartość
Pole	klasa, struktura	Zmienna zawierająca wartość danych
Właściwość	klasa, struktura	Obsługuje dostęp do pewnej wartości w ramach danej klasy
Konstruktor	klasa, struktura	
Metoda	klasa, struktura, interfejs	Funkcja związana z daną klasą
Zdarzenie	klasa, struktura, interfejs	Narzędzie umożliwiające informowanie przez daną klasę lub obiekt o zmianie stanu
Typ	klasa, struktura, interfejs	Klasy, interfejsy, struktury, delegacje

Składowe reprezentujące dane

Składowe reprezentujące dane (stałe, pola) reprezentują faktyczną zawartość lub stan klasy

Stałe pozwalają na przechowanie wartości które nie będą ulegać zmianą

Pola należy używać dla danych prywatnych klasy

Właściwości powinny być wykorzystywane do kontroli dostępu do danych składowych

Składowe reprezentujące dane - stałe

Do definiowania stałych wykorzystuje się słowo kluczowe **const**

Słowo kluczowe **const** może być wykorzystane do zdefiniowania wielu stałych w jednym wyrażeniu

```
public const double pi = 3.14159265358, e = 2.718281828459;
public const bool zawszePrawda = true;
```

Stała musi być typu prostego (np. int, double, bool, …)

Dostęp do stałych odbywa się tylko poprzez klasę

Składowe reprezentujące dane - stałe

```
class StaleMatematyczne  {
      public const double pi = 3.14159265358;
      public const double e = 2.718281828459;
   }
   class Program {
      static void Main(string[] args) {
         Console.WriteLine("Stała Pi wynosi = {0}",StaleMatematyczne.pi);
         Console.WriteLine("Satła e wynosi = {0}", StaleMatematyczne.e);

         StaleMatematyczne stale = new StaleMatematyczne();
         //Odwołanie do stałej za pośrednictwem obiektu
         //Console.WriteLine("Stala Pi wynosi = {0}", stale.pi);
      }}
```

Składowe reprezentujące dane - pola

Pole jest wykorzystywane do przechowywania danych wewnątrz klasy. Wartość pola określana jest w czasie wykonywania programu, nie musi być typu prostego

Pola powinny być definiowane z atrybutem **private** **(hermetyzacja klasy)**

W odniesieniu do pól można dodatkowo użyć modyfikatorów:

- **static**
- **readonly**

Składowe reprezentujące dane - pola

```
class StaleMatematyczne  {
      public readonly double pi = 3.14159265358;
      public static double e = 2.718281828459;
   }
   class Program {
      static void Main(string[] args) {

         Console.WriteLine("Satła e wynosi = {0}", StaleMatematyczne.e);
         StaleMatematyczne stale = new StaleMatematyczne();
         Console.WriteLine("Stala Pi wynosi = {0}", stale.pi);
      }}
```

Składowe reprezentujące dane - pola

```
class osoba
   {
      private string _nazwisko;
      private string _imie;
      private DateTime _dataUrodzenia;

      public int identyfikator;
   }
```

```
static void Main(string[] args)
      {
         osoba o1 = new osoba();

         // odwołanie do składowych publicznych
         // UWAGA !!! takie rozwiązanie powoduje
         // że klasa nie spełnia wymogu hermetyzacji

         o1.identyfikator=1234567890;
         Console.WriteLine(o1.identyfikator);

         // odwołanie do składowej prywatnej
         // UWAGA !!! tak nie można
         // należy użyć właściwości lub metody
         o1._nazwisko = "Kowalski";
         Console.WriteLine(o1._nazwisko);
      }
```

Przykład

Przykład 2-2.1

Visual Studio 2-2.1

Składowe reprezentujące dane - właściwości

Właściwości pozwalają na odczytywanie i zapisywanie wartości wewnątrz klasy

Właściwości odpowiadają metodą określanym jako akcesory i modyfikatory w języku C++ lub Java

Składnia dostępu do właściwości obiektu klasy jest identyczna jak w przypadku pól udostępnianych za pośrednictwem obiektów

nazwa_obiektu.nazwa_właściwości

Składowe reprezentujące dane - właściwości

[atrybuty] <modyfikator> <typ danych> <nazwa właściwości>

```
{
    [modyfikator dostępu] get
    { ...
        return (wartość właściwości)
    }
    [modyfikator dostępu] set
    { ... kod przypisujący polu wartość }
}
```

Składowe reprezentujące dane - właściwości

```
class produkty {
      private double cena;
      private double stawkaVat=0.22;

      public double cenaProduktu {
         get { return cena; }
         set { cena = value * stawkaVat + value; }
      }}

   class Program {
      static void Main(string[] args) {
         produkty p1 = new produkty();
         p1.cenaProduktu = 10;
         Console.WriteLine(p1.cenaProduktu);
      } }
```

Przykład 2 – 2

Visual Studio 2 - 2

Składowe definiujące funkcjonalność - metody

Składowe definiujące funkcjonalność (metody, konstruktor, zdarzenia) opisują zachowanie się klasy

Metody odpowiadają za wykonywanie działań

Składowe definiujące funkcjonalność - metody

[atrybuty] <modyfikator> <typ danych>
    <nazwa metody> ([lista parametrów])
{
    ...
    [return (zwracana wartość]

}

Przykład 2–3

Visual Studio 2-3

Składowe definiujące funkcjonalność - metody

Modyfikator	Opis
static	Metoda statyczna jest częścią stanu klasy. odwołanie przybiera postać **nazwaKlasy.metoda(parametry)**
virtual	Określa że dana metoda może zostać przykryta przez podklasę. Modyfikator virtual nie może być łączony z modyfikatorami private oraz static
override	Określa że dana metoda przykrywa metodę odziedziczoną po klasie bazowej
new	Określa, że element przesłania inny element z klasy bazowej, odmiennego typu. Praktycznie rzadko stosowane
sealed	Uniemożliwia klasom potomnym przykrywanie danej metody
abstract	Metoda nie zawiera szczegółów implementacji i musi zostać zaimplementowana w podklasach
extern	Oznacza że dana metoda jest implementowana zewnętrznie

Przykład 2-4

Visual Studio 2-4

Przykład 2-5

Visual Studio 2-5

Przekazywanie parametrów do metod

Parametry do metod domyślnie są przekazywane przez wartość

Możliwe jest przekazanie parametru do metody przez referencję. Przekazanie parametru przez referencję wymaga użycia modyfikatora **ref** lub **out**

Przekazywanie parametrów do metod

```
class produkty  {
      public static void incjacja(out double x)    {  x = 2.22;  }

      public static void zmiana(ref double y)     { y += y;  }
   }
   class Program  {
      static void Main(string[] args)  {
         double z;
         produkty.incjacja(out z);
         Console.WriteLine(z);
         produkty.zmiana(ref z);
         Console.WriteLine(z);
      } }
```

Przekazywanie parametrów do metod

Przykład 2-6

Przykład 2-6 Visual Studio

Przekazywanie parametrów do metod

Przykład 2-7

Przykład 2-7 Visual Studio

Przekazywanie parametrów do metod

Przykład 2-8

Przykład 2-8 Visual Studio

Przekazywanie parametrów do metod

Przykład 2-9

Przykład 2-9 Visual Studio

Dziękuję za uwagę