Programowanie pod Windows
Zestaw 2

Język C# - podstawowe elementy

2024-02-27

Liczba punktów do zdobycia: **10/16**
Zestaw ważny do: 2024-03-12

1. (**2p**) Zademonstrować w praktyce i rozumieć sens następujących elementów języka. Jeśli w nawiasie po elemencie języka występuje kilka możliwych elementów do których może on się odnosić, proszę wybrać co najmniej jedną z propozycji, ale niekoniecznie wszystkie.

   - modyfikator `static` dla klas
   - modyfikator `static` dla składowych klas (pól, metod)
   - modyfikator `sealed` dla klas
   - modyfikator `abstract` dla klas
   - modyfikator `abstract` dla składowych klas (metod)
   - słowa kluczowe `virtual` i `override` dla składowych klas (metod)
   - słowo kluczowe `partial` w definicji klasy
   - słowo kluczowe `readonly` w deklaracji pola klasy
   - modyfikatory `in`, `ref` oraz `out` na liście parametrów metody
   - modyfikator `params` na liście parametrów metody

2. (**2p**) Czym różni się mechanizm finalizerów (zwanych czasem destruktorami) od mechanizmu zwalniania zasobów za pomocą implementacji interfejsu `IDisposable`?

   Zaprezentować oba w praktyce: przygotować klasę która ma finalizer i inną klasę implementującą interfejs `IDisposable`. W obu podejściach wywołać `Console.WriteLine` z metody "sprzątającej".

   Zaprezentować lukier syntaktyczny opakowujący użycie obiektu implementującego `IDisposable` w blok ze słowem kluczowym `using`.

   Zaobserwować, że w przypadku interfejsu `IDisposable` programista ma pełną kontrolę nad momentem w którym wykonuje się metoda `Dispose`. Zaobserwować że programista nie ma wpływu na to kiedy wykona się finalizer klasy.

   Czy można wymusić wywołanie metody sprzątającej pamięć (odśmiecacz)? Czy to dobry pomysł, żeby wymuszać to we własnym kodzie?

3. (**1p**) Pokazać jak definiować właściwości (ang. *properties*)

   - właściwość z polem kopii zapasowej (ang. *property with backing field*)
   - właściwość implementowana automatycznie (ang. *auto-implemented property*)

4. (**2p**) Rozszerzyć poprzedni przykład o demonstrację właściwości posiadających skutki uboczne. Formalnie, niech będzie dana klasa

```
public class Person
{
        public string Name { get; set; }
        public string Surname { get; set; }
}
```

Klasę zmodyfikować tak, żeby udostępniała zdarzenie (ang. *event*) informujące subskrybenta o tym że zmieniła się wartość którejś z właściwości.

Czy potrzebne są do tego dwa osobne zdarzenia? Które podejście jest lepsze - jedno zdarzenie o ogólniejszej sygnaturze czy wiele osobnych zdarzeń, po jednym zdarzeniu dla każdego pola?

Formalnie, klient chciałby z powiadamiania korzystać w sposób przedstawiony poniżej - tak zmodyfikować kod klasy `Person` żeby było to możliwe.

Zaprezentować dwa warianty

- powiadomienie pojawia się zawsze kiedy nadana jest wartość właściwości (formalnie - kiedy wywołuje się akcesor `set`)
- powiadomienie pojawia się tylko wtedy kiedy nowa wartość pola jest różna od poprzedniej

5. (**2p**) Zaimplementować klasę reprezentującą siatkę (tablicę) dwuwymiarową, `Grid`, z dwoma indekserami (przeciążeniami operatora indeksowania):

   - dwuwymiarowym, zwracającym określony element tablicy, tak aby klient klasy mógł napisać:

     ```
     ...
     Grid grid = new Grid( 4, 4 );

     elem[2, 2] = 5;         // akcesor "set"
     int elem = grid[1, 4]; // akcesor "get"
     ```

   - jednowymiarowym, zwracającym listę elementów zadanego wiersza tablicy, tak aby klient klasy mógł napisać:

     ```
     ...
     Grid grid = new Grid( 4, 4 );
     int[] rowdata = grid[1]; // akcesor "get"
     ```

   Oba operatory indeksowania powinny przyjmować jako parametry liczby całkowite. Konstruktor klasy powinien przyjmować jako parametry liczbę wierszy i liczbę kolumn siatki. Wewnętrznie, siatka może użyć prywatnej składowej, zwykłej tablicy dwuwymiarowej, do przechowywania danych.

   Przy okazji przeczytać o tym czym się różnią zwykłe tablice wielowymiarowe od tzw. tablic postrzępionych (ang. *jagged arrays*).

```
static void Main( string[] args )
{
        Person person                = new Person();
        person.PropertyValueChanged += Person_PropertyValueChanged;
        person.Name                  = "Jan";

        Console.ReadLine();
}

private static void Person_PropertyValueChanged(
        object source,
        string propertyName,
        object propertyValue )
{
        Console.WriteLine(
                "właściwość {0}, nowa wartość {1}",
                propertyName,
                propertyValue );
}
```

6. (**1p**) Pokazać jak przeciąża się operatory - zdefiniować klasę wektora dwuwymiarowego i dodać do niej standardowe operatory arytmetyki na wektorach (wystarczą ze dwa).

Wiktor Zychla