Eventy & Delegaty.

C#

Wydział Matematyki i Informatyki
Ćwiczenia: Michał Mnich

Wstęp teoretyczny.

Poniżej prezentuję wstęp teoretyczny, który może okazać się przydatny podczas pisania zadań. Wykład 2. powinien być wystarczający jako główne źródło.

Delegaty.

Delegaty są często porównywane do wskaźników funkcji z języka C++. Faktem jest, że C# czerpie wiele z języka C++ i Javy, jednak wiele rzeczy jest moim zdaniem ulepszonych i prostszych w użyciu, tak jak delegaty.

Podobnie jak podczas zjawisko polimorfizmu o tym, iż decyzja, jaki obiekt ma zostać utworzony, może zapaść w trakcie działania programu. Teraz wyobraź sobie sytuację, w której nie jesteś pewien, jaka metoda ma zostać wykonana, decyzja ta zostanie podjęta w trakcie działania aplikacji, np. w zależności od decyzji użytkownika.

Np. użyty algorytm zostanie wybrany przez program w trakcie działania na podstawie przeanalizowanych danych. Owszem — możesz wykorzystać instrukcję if lub switch, ale przy bardziej rozbudowanych aplikacjach praktyczniejsze będzie użycie delegatów.

Można powiedzieć, że delegaty przechowują referencję (adres w pamięci) do danych metod. Korzystając z delegatów, możesz wykonać kod znajdujący się w metodach.

INFO: Delegaty to w rzeczywistości nowe typy danych. Mogą być zadeklarowane w klasie jako typ zagnieżdżony lub poza nią. Oto przykład:

```
public delegate int Foo(int X);
```

Tworzenie nowych delegatów jest proste, wygląda jak deklarowanie nowej metody. Jedyną różnicą jest słówko kluczowe delegate.

Delegaty są nowymi typami danych. Aby je wykorzystać, tworzymy więc nowe zmienne wskazujące na te typy, do których możemy przypisać metody odpowiadające sygnaturze delegatu.

INFO: Przykład użycia delegatów*:*

```
using System;

namespace DelegateApp
{
   public delegate int Foo(int X);

   class Program
   {
      static int Power(int X)
      {
         return X * X;
      }

      static int Div(int X)
      {
         return X / 2;
      }

      static void Main(string[] args)
      {
         Foo MyFoo = new Foo(Power);

         Console.WriteLine("10x10 = " + MyFoo(10));

         MyFoo = Div;

         Console.WriteLine("10/2  = " + MyFoo(10));
         Console.Read();
      }
   }
}
```

Zdarzenia.

Zdarzenia są specjalnym rodzajem delegatów multiemisji, który można wywołać tylko z poziomu klasy lub struktury, w której są zadeklarowane (Klasa wydawcy). Jeśli inne klasy lub struktury subskrybują zdarzenie, ich metody obsługi zdarzeń będą wywoływane, gdy Klasa wydawcy zgłasza zdarzenie. Zdarzenia mogą być oznaczone jako publiczne, prywatne, chronione, wewnętrzne, chronione wewnętrznie lub chronione prywatnie. Te Modyfikatory dostępu definiują sposób, w jaki użytkownicy klasy mogą uzyskać dostęp do zdarzenia. Aby uzyskać więcej informacji, zobacz Modyfikatory dostępu.

INFO: Przykład użycia Zdarzeń*:*

```
public delegate void Notify();  // Delegata

public class ProcessBusinessLogic
{
   public event Notify ProcessCompleted; // Zdarzenie

   public void StartProcess()
   {
      Console.WriteLine("Process Started!");
      // some code here..
      OnProcessCompleted();
   }

   protected virtual void OnProcessCompleted() //Metoda protected virtual
   {
      //if ProcessCompleted is not null then call delegate
      ProcessCompleted?.Invoke();  //Zgłaszanie zdarzenia
   }
}
```

INFO: Przykład użycia nasłuchiwania zdarzeń*:*

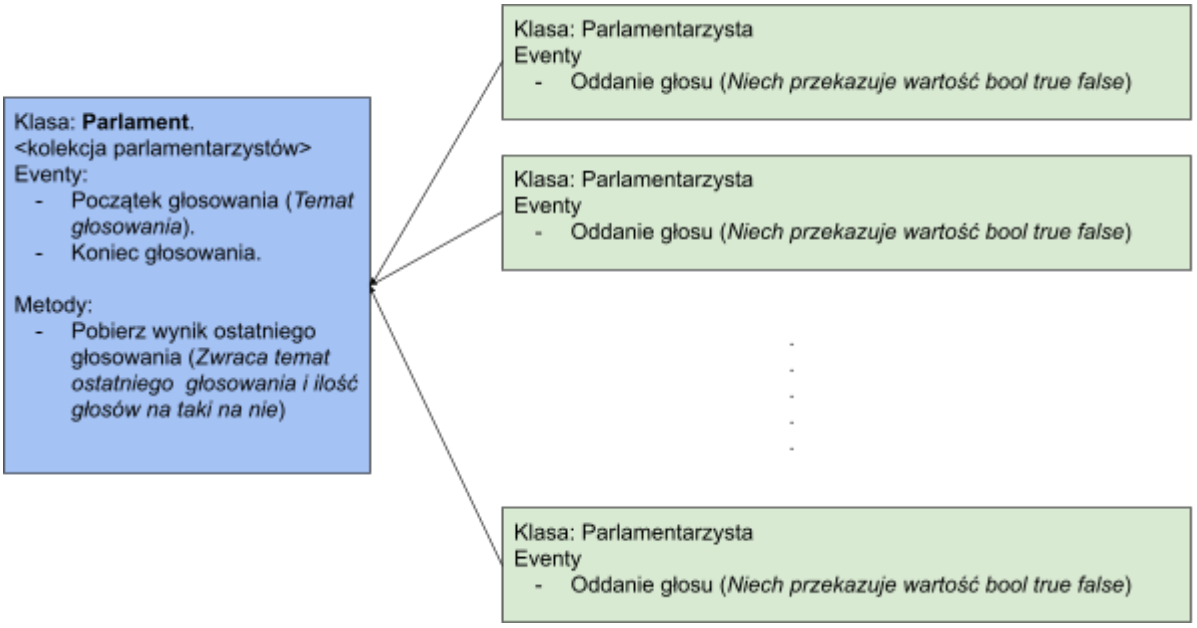
```
class Program
{
   public static void Main()
   {
      ProcessBusinessLogic bl = new ProcessBusinessLogic();
      bl.ProcessCompleted += bl_ProcessCompleted; // register with an event
      bl.StartProcess();
   }

   // Tutaj mamy metodę, która uruchomi się kiedy event zostanie aktywowany
   public static void bl_ProcessCompleted()
   {
      Console.WriteLine("Process Completed!");
   }
}
```

Przed wykonaniem zadania prosze uważnie przejrzęc wprzykłady z wykładu a także, uruchomić powyższy prosty przykład.

ZD: Zadania będzie polegało na utowrzniu symulatora parlamentu.
Wymagania funkcjonalne to. Każdy parlamentarzysta słucha zdarzenia Początek głosowania i koniec głosowania (Nazwy zdarzeń proszę ustawić po angielsku). Parlament powinien niech nasłuchuje na zdarzenie oddanie głosu przez parlamentarzystę.

Schemat użycia:



Wejście:
Zadajemy liczbę parlamentarzystów oraz podajemy temat.
Następnie uruchamiamy symulację

Działanie:
Każdy parlamentarzysta głosuje w sposób losowy (C# RAND) poprzez wpisanie do konsoli GŁOS [nr Parlamentarzysty] Nie wolno głosować przed uruchomieniem eventu początek głosowania ani po evencie koniec głosowania. Prosze obłsużyć to odpowiednimi komunikatami. Głosowanie rozpoczynamy komendą POCZATEK [tytol]. Koniec komendą KONIEC.

Wyjście:
NIech pojawi sie wiadomość: głosowanie nad (tutaj podać temat) Głosów za (liczba) Głosów przeciw (liczba)

Prosze w całym projekcie sotoswac sie do zasad stylizacji kody Camel Code (https://en.wikipedia.org/wiki/Camel_case)

KOD do zadania.

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

namespace cw01
{
    class Program
    {
        static void Main(string[] args)
        {
            ProcessBusinessLogic logic = new ProcessBusinessLogic();
```

```
            logic.ProcessCompleted+= LogicOnProcessCompleted;
            logic.ProcessCompleted2+=LogicOnProcessCompleted2;
            logic.ProcessCompleted3+= ProcessCompleted3;
            logic.StartProcess();

            Console.ReadKey();
        }

        private static void ProcessCompleted3(object sender, MyEventArgs e)
        {
            Console.WriteLine("event03");
            MyEventArgs args = e;
            Console.WriteLine(args.SomeArgs);
        }

        private static void LogicOnProcessCompleted2(object sender, EventArgs e)
        {
            Console.WriteLine("event02");

        }

        private static void LogicOnProcessCompleted()
        {
            Console.WriteLine("event01");
        }
    }

    public delegate void Notify();  // Delegata

    public class ProcessBusinessLogic
    {
        public event Notify ProcessCompleted; // Zdarzenie
        public event EventHandler ProcessCompleted2; // Zdarzenie
        public  EventHandler<MyEventArgs> ProcessCompleted3; // Zdarzenie

        public void StartProcess()
        {
            Console.WriteLine("Process Started!");
            // some code here..
            OnProcessCompleted();
            OnProcessCompleted2();
            OnProcessCompleted3();
        }

        protected virtual void OnProcessCompleted() //Metoda protected virtual
        {
            //if ProcessCompleted is not null then call delegate
            ProcessCompleted?.Invoke();  //Zgłaszanie zdarzenia
        }

        public virtual void OnProcessCompleted2() //Metoda protected virtual
        {
            //if ProcessCompleted is not null then call delegate
            ProcessCompleted2?.Invoke(this, EventArgs.Empty);  //Zgłaszanie zdarzenia
        }

        protected virtual void OnProcessCompleted3() //Metoda protected virtual
        {
            //if ProcessCompleted is not null then call delegate
            ProcessCompleted3?.Invoke(this, new MyEventArgs(){ SomeArgs = "hello"});  //Zgłaszanie zdarzenia
        }

    }

    public class MyEventArgs : EventArgs
    {
        public string SomeArgs { get; set; }
    }

}
```