Delegaty, Zdarzenia, Programowanie Funkcyjne

infoShare Academy

HELLO

Mikołaj Toczek

infoShare ACADEMY

Agenda

Delegaty

Zdarzenia

Programowanie funkcyjne


01.
Delegaty


infoShare
ACADEMY

Delegaty – co to?

Delegacja (ang. *delegation*) - w programowaniu obiektowym, działanie w którym obiekt zamiast sam wykonać pewną operację zleca ją do wykonania innemu obiektowi (delegatowi).

Delegata (ang. *delegate*) - w języku C#, typ który reprezentuje referencje (wskaźniki) do metod z parametrami określonego rodzaju i sprecyzowanym zwracanym typem.

Przechowywane w delegacie referencje mogą zmieniać się w trakcie działania programu (tak jak w przypadku każdej innej zmiennej), przy czym sygnatura i zwracany typ przypisanej do delegaty funkcji muszą być kompatybilne z oryginalną deklaracją (tak jak do zmiennej oznaczonej jako *int* nie możemy później przypisać daty).

Delegaty – funkcje anonimowe

Za pomocą wyrażeń lambda, do delegat można również przypisywać funkcje anonimowe

- zarówno zwracające wartość :

```
public delegate int SomeOperationOnThreeIntegers(int n1, int n2, int n3);
```

```
SomeOperationOnThreeIntegers sumOfThreeIntegers = (int n1, int n2, int n3) => n1 + n2 + n3;
SomeOperationOnThreeIntegers productOfThreeIntegers = (int n1, int n2, int n3) => n1 * n2 * n3;
SomeOperationOnThreeIntegers maxOfThreeIntegers = (int n1, int n2, int n3) => Math.Max(Math.Max(n1, n2), n3);
```

- jak i *void*:

```
public delegate void EmployeeDaySchedule();
```

```
EmployeeDaySchedule johnsDaySchedule = () => Console.WriteLine("I sleep all day");
```

Delegaty predefiniowane

W .NET dostępne są **delegaty predefiniowane**, które ze względu na użycie parametrów generycznych są pomocne i wystarczają w większości przypadków, w których chcielibyśmy użyć delegaty.

Mianowicie:

- **Action** - wskazuje na metodę, która może (choć nie musi) posiadać parametr/y wejściowy/e ale typ zwracany to *void* (robi "coś", ale nic nie zwraca)
- **Func** - wskazuje na metodę, która posiada niezerową liczbę parametrów wejściowych i niepusty typ zwracany
- **Converter** - wskazuje na metodę, która przyjmuje parametr typu *A* i zwraca obiekt typu *B* (konwertuje z *A* na *B*)
- **Predicate** - wskazuje na metodę, która przyjmuje obiekt, a zwraca *bool* (używane do sprawdzania czy obiekt spełnia jakiś warunek)
- **Comparison** - wskazuje na metodę, która przyjmuje dwa parametry tego samego typu i zwraca *int* (dokonuje porównania parametrów wejściowych)

Delegaty - ćwiczenie

Uzupełnij program DelegatesExercise, tak aby spełniał założenia opisane w opcjach menu.

- w tym celu zmodyfikuj klasę CarService tak, by oznaczone metody przyjmowały delegaty jako parametry

- w ciele tych metod użyj odpowiednich funkcji LINQ

- w klasie Program.cs zmodyfikuj kod tak aby wywoływać metody z CarService z odpowiednimi parametrami-delegatami (buduj je za pomocą wyrażeń lambda)

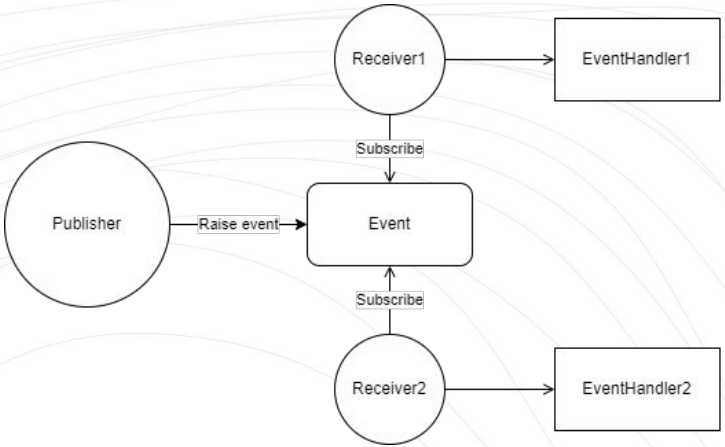
02. Zdarzenia

Zdarzenia

Dostępne w .NET **zdarzenia** (ang. *events*) to, wykorzystująca delegaty, metoda na powiadamianie innych obiektów o czymś co zaszło w kontekście danego obiektu.

Obiekt publikujący zdarzenie (ang. *publisher*) nadaje informację o nim a inne obiekty nasłuchujące na owo zdarzenie reagują we wcześniej zdefiniowany sposób.

Taki mechanizm nazywamy wzorcem **obserwatora** (ang. *observer design pattern*).



Zdarzenia - ćwiczenie

Zmodyfikuj kod DelegatesExercise tak, aby:

a) klasa *CarService* informowała poprzez zdarzenie *CarsListAccessed* za każdym razem, kiedy pobierana jest lista samochodów (przy każdym wywołaniu *GetDefaultCarsList()*)
b) klasa *Program* nasłuchiwała na powyższe zdarzenie i przy każdym jego odpaleniu wypisywała na czerwono w konsoli *"Cars list accessed"*
c) wewnątrz funkcji reagującej na pierwsze zdarzenie, przy każdym jego propagowaniu inkrementacji ulega statyczny licznik, a jeżeli w danym uruchomieniu programu lista samochodów została pobrana już piąty raz, propagowany jest event *"CarsListAccessedFiveTimes"*
d) funkcja-odbiorca zdarzenia *"CarsListAccessedFiveTimes"* zamykała program, jeżeli owo zdarzenie się pojawi

Programowanie funkcyjne

infoShareAcademy.com

Programowanie funkcyjne

W programowaniu obiektowym (OOP), implementacja wymagań i realizacja założeń programu odbywa się poprzez tworzenie i manipulowanie złożonymi obiektami, które poza swoim zachowaniem przechowują również pewnego rodzaju stan, który zmienia się w trakcie wykonywania programu.

Programowanie funkcyjne to alternatywny do programowania obiektowego styl, w którym implementuje się założony program poprzez składanie wielu jak najprostszych, *czystych* funkcji w moduły realizujące jakąś funkcjonalność.

“Czysta” funkcja (ang. *pure function*) - to taka, która nie zależy od, ani nie zmienia globalnego stanu programu. Można więc powiedzieć, że jest to funkcja pozbawiona efektów ubocznych. Jej wynik zależy wyłącznie od parametrów wejściowych i zawsze będzie taki sam dla takiego samego zbioru owych parametrów.

Programowanie funkcyjne – przykład

```
Func<int, int, int> Add = (n1, n2) => n1 + n2;
Func<int, int, int> Subtract = (n1, n2) => n1 - n2;
Func<int, int, int> Multiply = (n1, n2) => n1 * n2;
Func<int, int, int> Divide = (n1, n2) => n1 / n2;

Func<string> GetTextFromConsole = () => Console.ReadLine();
Func<string, int> ParseTextToInt = text => int.Parse(text);
Action<int> WriteNumberToConsole = number => Console.WriteLine(number);
Action<string> WriteTextToConsole = text => Console.WriteLine(text);

Func<(int, int)> GetParameters = () =>
{
    WriteTextToConsole("Please provide two numbers");
    return (ParseTextToInt(GetTextFromConsole()), ParseTextToInt(GetTextFromConsole()));
};

Func<int, (int, int), int> GetSelectedOperationResultForInputs = (operationIndex, parameters) =>
{
    return operationIndex switch
    {
        1 => Add(parameters.Item1, parameters.Item2),
        2 => Subtract(parameters.Item1, parameters.Item2),
        3 => Multiply(parameters.Item1, parameters.Item2),
        4 => Divide(parameters.Item1, parameters.Item2),
        _ => throw new NotImplementedException()
    }; ;
};
```

Programowanie funkcyjne – przykład c.d.

```
Func<int> GetSelectedOperationIndex = () =>
{
    WriteTextToConsole("Select operation to perform: ");
    WriteTextToConsole("[1] Add");
    WriteTextToConsole("[2] Subtract");
    WriteTextToConsole("[3] Multiply");
    WriteTextToConsole("[4] Divide");

    return ParseTextToInt(GetTextFromConsole());
};

Action<int> WriteResultToConsole = (result) =>
{
    WriteTextToConsole("The result is: ");
    WriteNumberToConsole(result);
};

WriteResultToConsole(GetSelectedOperationResultForInputs(GetSelectedOperationIndex(), GetParameters()));
```

Warto zajrzeć do... dokumentacji

- Wyrażenia lambda
- Delegaty
- Zdarzenia

Czas na pytania!

Dziękuję za uwagę!

infoShareAcademy.com