Zaawansowane programowanie obiektowe - wykład 8

Opracowanie: dr Piotr Jastrzębski

Delegaty i wyrażenia lambda

Delegaty

- Delegat to obiekt „wiedzący”, jak wywołać metodę.
- Typ delegacyjny definiuje rodzaj metody, jaki mogą wywoływać egzemplarze delegatu. Dokładniej: określa typ zwrotny metody i typy jej parametrów.

Definicja typu delegacyjnego o nazwie Transformer:

```
public delegate int Transformer (int x);
```

Typ Transformer jest zgodny z każdą metodą o typie zwrotnym int przyjmującą jeden parametr typu int np. taką:

```
static int Square (int x) { return x * x; }
```

Przypisanie metody do zmiennej typu delegacyjnego powoduje utworzenie egzemplarza delegatu:

```
Transformer t = Square;
```

Egzemplarz ten można wywoływać tak jak zwykłą metodę:

```
int answer = t(3); // wynik to 9
```

```
public delegate int Transformer(int x);
class Program
{
    static int Square(int x) { return x * x; }
    static void Main(string[] args)
    {
        Transformer t = Square; // utworzenie egzemplarza
        int result = t(3); // wywołanie delegatu
        Console.WriteLine(result); // 9
        Console.ReadKey();
    }
}
```

kod “po staremu”

```
static int Square(int x) { return x * x; }
static void Main(string[] args)
{
    int result = Square(3);
    Console.WriteLine(result); // 9
    Console.ReadKey();
}
```

Zapisy skrócone

Instrukcja:

```
Transformer t = Square;
```

jest skróconą formą zapisu instrukcji:

```
Transformer t = new Transformer (Square);
```

Wyrażenie:

```
t(3)
```

jest skróconą formą zapisu:

```
t.Invoke(3)
```

Po co delegaty?

- Projektując różne rozbudowane systemy istniała potrzeba, aby funkcja wywoływana mogła komunikować się z funkcją wywołującą.
- Delegaty są obiektami dziedziczącymi bezpośrednio niejawnie z `MulticastDelegate`. Ta z kolei dziedziczy z klasy `Delegate` oraz `Object`.

Podpisanie metod do delegatów

Mając zadeklarowany delegat, możemy podpiąć do niego dowolną metodę. Ważne jest to, aby zgodne były argumenty oraz typ zwracany.

Delegaty w C# obsługują tzw. multicasting tzn. możliwe jest podpięcie wielu metod do jednego delegata.

Podpinanie metod pod delegat odbywa się za pomocą:

- pierwsza metoda podpinana jest poprzez konstruktor, podczas tworzenia instancji delegata
- każdą następną metodę możemy dodać/usunąć za pomocą operatorów += oraz -=

```
public delegate void Dzialanie(int x, int y);
public class Matma
{
    public void Dodaj(int l1, int l2) { Console.WriteLine(l
    public void Odejmij(int l1, int l2) { Console.WriteLine
}
class Program
{
    static void Main(string[] args)
    {
        Matma matma = new Matma();
        Dzialanie dzialanie = new Dzialanie(matma.Dodaj);
        dzialanie(5,5);
        dzialanie += matma.Odejmij;
        dzialanie(7, 4);
        Console.ReadKey();
    }
}
```

Delegaty jako funkcje wywołań zwrotnych

Delegaty używane są do tworzenia funkcji zwrotnych tzw. callbacków. W tym celu deklarujemy delegat wewnątrz interesującej nas klasy.

Callback - funkcja zwrotna, której idea używania jest przeciwna do standardowego używania funkcji. Zamiast wywoływać interesujące nas funkcje, nasz obiekt wywołuje delegat. To na co będzie wskazywał delegat w przyszłości, zależy już od innych obiektów i programistów pracujących na naszej klasie.

Dwie zasady:

- modyfikator `public`
- pamiętamy o hermetyzacji!

Kod-delegaty3.

Gotowe delegaty

- `Func`
- `Action`

Func

Delegat `Func` jest delegatem generycznym. Podczas deklaracji trzeba określić typ zwracany oraz typ parametrów. Konstruktor został przeciążony aż 17 razy.

Należy pamiętać, że w kwadratowych dzióbkach końcowy parametr jest zawsze tym zwracanym. Wszystkie jakie dopiszemy wcześniej będą wejściowymi:

```
Func<out>
Func<in, out>
Func<in, in, out>
```

Formalna definicja delegata Func wygląda następująco:

```
public delegate TResult Func<in T, out TResult>(T arg);
```

```
public class Matematyka
{
    public int Dodawanie(int x, int y)
    {
        return x + y;
    }
}
class Program
{
    static void Main()
    {
        Matematyka m = new Matematyka();
        Func<int, int, int> dodawanie = m.Dodawanie;
        Console.WriteLine(dodawanie(1, 2));
        Console.ReadKey();
    }
}
```

Action

- stosowany do funkcji z typem zwracanym `void`

Definicja formalna:

```
public delegate void Action<in T>(T arg);
```

Wyrażenia lambda

```
(parametry) => {zwracana wartość}

(int a, int b) => {return a + b;}
```

Przykład

```
Func<int, int, int> foo = (int a, int b) => { return a + b;

Console.WriteLine(foo(5, 4));
```

Równoważne postacie:

```
Func<int, int, int> foo = (a, b) => { return a + b; };
Func<int, int, int> foo = (int a, int b) =>  a + b;
Func<int, int, int> foo = (a, b) =>  a + b;
Func<int, int> foo = a => a * 5;
```

```
Func<int> foo = () =>
{
    int a = 5;
    return a;
};

Console.WriteLine(foo());
```

```
int x = 5;

Func<int> foo = () => x;

Console.WriteLine(foo());
```

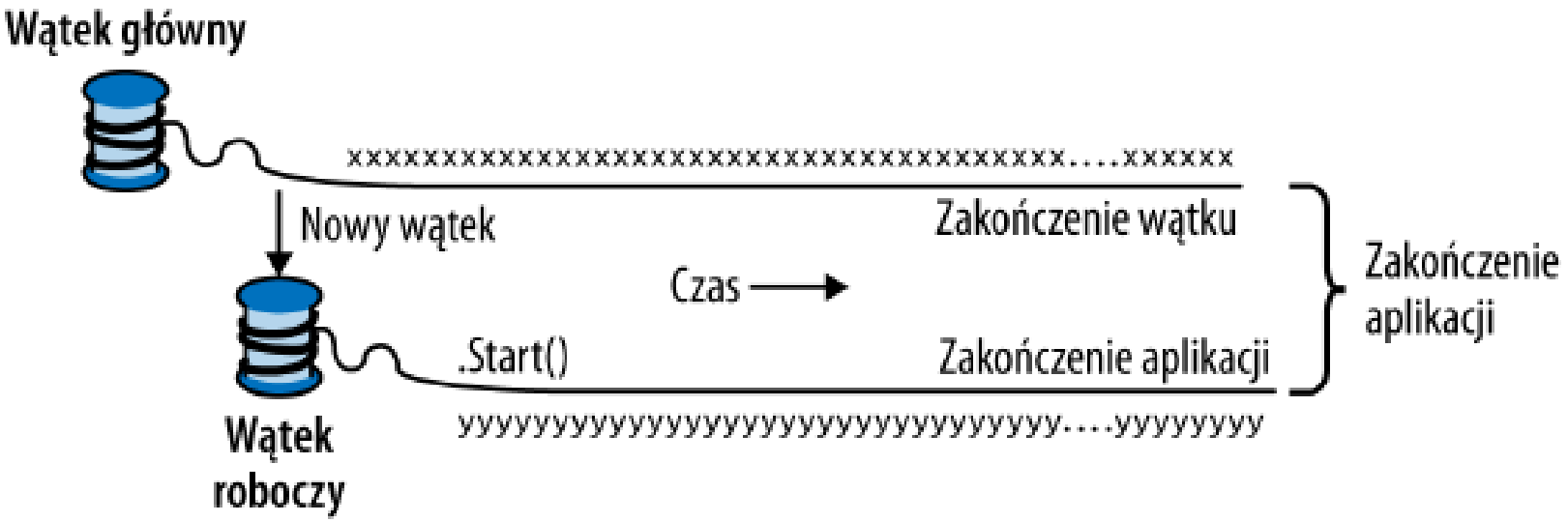
```
int x = 5;

Func<int> foo = () => x;

x = 6;

Console.WriteLine(foo());
```

Wątki i kod asynchroniczny

Wątki i kod asynchroniczny

Współbieżność - właściwość zbioru procesów, które są wykonywane w tym samym czasie.

Wątek to ścieżka wykonywania, która może być realizowana niezależnie od innych ścieżek.



Rysunek 1:

Istota wielowątkowości zasadniczo jest prosta - chodzi przede wszystkim o maksymalne wykorzystanie dostępnych zasobów i zrównoleglenie kilku wykonywanych naraz akcji. Każda domyślnie stworzona aplikacja posiada tylko jeden wątek główny, co widoczne jest szczególnie w przypadku aplikacji z GUI. Jeśli w tym przypadku wątek główny zacznie wykonywać jakąś złożoną operację, nasz program nie będzie w stanie wykonać cyklicznej operacji odświeżania interfejsu i mimo, że formalnie program będzie wciąż działać, to aplikacja znajdzie się w klasycznym stanie braku odpowiedzi.

Priorytety:

- Lowest
- BelowNormal
- Normal
- AboveNormal
- Highest

Stan wątku:

- Running = 0
- StopRequested = 1
- SuspendRequested = 2
- Background = 4
- Unstarted = 8
- Stopped = 16
- WaitSleepJoin = 32
- Suspended = 64
- AbortRequested = 128
- Aborted = 256

Schemat: http://img2.altcontroldelete.pl/images/f74c9c91-c2f5-5aa2-bec3-bb8c702c13b4.png

Stan lokalny kontra współdzielony

- Program Watki5 - zmienne są oddzielne dla każdego wątku.
- Program Watki6 - współdzielenie danych.
- Program Watki7 - statyczność.
- Program Watki8 - nałożenie blokady.

Wyjątki a wątki

- Watki12 - źle.
- Watki13 - dobrze.

Taski

Nową klasą w C# 4.0 jest System.Threading.Tasks.Task. Umożliwia ona wykonanie operacji w tle(w osobnym wątku). Samo stworzenie Task'a jest analogiczne do Thread:

```
Task t1 = new Task(() => Console.Write("task 1"));
t1.Start();
```

Zaczekanie aż task zostanie wykonany:

```
Task t1 = new Task(() => Console.Write("task 1"));
t1.Start();
t1.Wait();
```

Synchroniczność

Operacja synchroniczna wykonuje swoją pracę przed przekazaniem kontroli z powrotem do komponentu, który ją wywołał. np. `List<T>.Add()`, `Console.WriteLine()`.

Z kolei operacja asynchroniczna wykonuje (większość) pracy po przekazaniu kontroli z powrotem do komponentu, który ją wywołał. np. `Thread.Start()`.

Programowanie asynchroniczne

Zasada w programowaniu asynchronicznym polega na tym, że długo wykonywane (lub potencjalnie długo działające) funkcje są tworzone w sposób asynchroniczny. Jest to kontrast do podejścia konwencjonalnego, polegającego na synchronicznym tworzeniu długo wykonywanych funkcji, a następnie wywoływaniu ich w nowym wątku lub też zadaniu w celu zastosowania współbieżności. Różnica w podejściu asynchronicznym polega na tym, że współbieżność jest inicjowana wewnątrz długo wykonywanej funkcji, a nie na jej zewnątrz.

async, await

W C# 5.0 odpowiedzialne jest za to słowo kluczowe await, które czeka na wynik działania metody. Metoda która zawiera w sobie słowo kluczowe await musi zostać poprzedzona słowem kluczowym async, które informuje o asynchronicznym wywołaniu metody. Asynchroniczne wywołanie metod nie oznacza, że będą wykonywane na kilku wątkach, wręcz przeciwnie metody są wywoływane na jednym wątku tak długo jak to tylko możliwe.

Bibliografia

- Wikipedia, wikibooks na licencji CC.
- Joseph Albahari, Ben Albahari, C# 7.0 w pigułce, wyd. Helion, 2018.
- http://www.altcontroldelete.pl/artykuly/wielowatkowosc-w-c-wprowadzenie/. dostęp online 5.04.2019.
- https://cezarywalenciuk.pl/blog/programing/post/kurs-obiektowosc-w-c-wyrazenia-lambda-delegaty-19, dostęp online 5.04.2019.
- https://4programmers.net/C_sharp/Wyra%C5%BCenie_Lambda, dostęp online 5.04.2019.
- http://www.pzielinski.com/?p=192, dostęp online 5.04.2019.
- http://kurs.aspnetmvc.pl/Csharp/Metody_asynchroniczne, dostęp online 5.04.2019.