PROGRAMOWANIE W JĘZYKU C#

TOMASZ WOJNAROWSKI

WYRAŻENIA LAMBDA

- Wyrażenie lambda jest funkcją anonimową, której możesz użyć do tworzenia typów delegatów lub drzew wyrażeń.
- Za pomocą wyrażenia lambda, można napisać funkcje lokalne, które mogą być przekazywane jako argumenty lub zwracane jako wartość wywołania funkcji.
- Wyrażenia lambda są szczególnie przydatne w przypadku wyrażeń zapytanie LINQ.

WYRAŻENIA LAMBDA

- Wyrażenie lambda jest funkcją anonimową, której możesz użyć do tworzenia typów delegatów lub drzew wyrażeń.
- Za pomocą wyrażenia lambda, można napisać funkcje lokalne, które mogą być przekazywane jako argumenty lub zwracane jako wartość wywołania funkcji.
- Wyrażenia lambda są szczególnie przydatne w przypadku wyrażeń zapytanie LINQ.
- Jeśli wyrażenie lambda pobiera jakieś parametry możesz nie podawać ich typów ponieważ kompilator skojarzy je z kontekstu danego wyrażenia. Jednak przy słowach kluczowych jak ref i out musisz też podać ich typ.
- Wyrażenie lambda może zmienić wartości na zawsze jeśli są one przesłane do metody za pomocą słów kluczowych ref i out.

LENIWE INICJALIZOWANIE

- Klasa System.Lazy<T>
- Obiekt zostanie zainicjowany w momencie pierwszego odwołania
- Klasa Lazy opakowuje obiekt danego typu, dzięki czemu jego inicjalizacja następuje dopiero w momencie pierwszego użycia
- Inicjalizacja z opóźnieniem służy przede wszystkim do zwiększenia wydajności, pozwala uniknąć niepotrzebnych obliczeń i zmniejszyć wymagania dotyczące pamięci programu

LENIWE INICJALIZOWANIE

```
static void Main(string[] args)
{
    Lazy<string> HelloWorld = new Lazy<string>(() => "Witaj Świecie");
    Console.WriteLine("Czy zmienna została zainicjalizowana ? {0}", HelloWorld.IsValueCreated);
    Console.WriteLine(HelloWorld.Value);
    Console.WriteLine("Czy zmienna została zainicjalizowana ? {0}", HelloWorld.IsValueCreated);
}
```

SINGELTON

```
private static Singleton instance;

private Singleton() { }

public static Singleton Instance
{
    get
    {
        if (instance == null)
        {
            instance = new Singleton();
        }
        return instance;
    }
}
```

SINGELTON

```
private static Singleton instance;

private Singleton() { }

public static Singleton Instance
{
    get
    {
        if (instance == null)
        {
            instance = new Singleton();
        }
        return instance;
    }
}
```

SINGELTON ZALETY

- Przenośność pomiędzy platformami
- Leniwe konstruowanie
- Możliwość zastosowania dla dowolnej klasy

SINGELTON WADY

- Nieznany czas destrukcji
- Efektywność (zawsze wykonywany jest if)
- Nieczytelność (wszyscy muszą wiedzieć że tworzony jest przez Instance)

ATRYBUTY

- Atrybut to znacznik, który służy do przekazywania informacji do środowiska wykonawczego o zachowaniu różnych elementów, takich jak: klasy, metody, struktury, typy wyliczeniowe.
- Do deklaracji atrybutów używamy nawiasów klamrowych
- Dziedziczą po klasie a

ATRYBUTY

```
[assembly: AssemblyTitle("Lab5-04-Attributes")]
[assembly: AssemblyDescription("Zmiana atrybutów")]
[assembly: AssemblyConfiguration("")]
[assembly: AssemblyCompany("Converse")]
[assembly: AssemblyProduct("Lab5-04-Attributes")]
[assembly: AssemblyCopyright("Copyright ©  2017")]
[assembly: AssemblyTrademark("")]
[assembly: AssemblyCulture("")]
[assembly: ComVisible(false)]
[assembly: Guid("e7903d06-2ab1-44fe-8fbb-5ddf368241cb")]
[assembly: AssemblyVersion("1.0.0.0")]
[assembly: AssemblyFileVersion("1.0.0.0")]
```

TESTY AUTOMATYCZNE

- Są to programy do automatycznego testowania aplikacji
- Raz napisane mogą być wykorzystywane wielokrotnie
- Chronią przed błędami regresywnymi
- Wspomagają refaktoryzację

RODZAJE TESTÓW AUTOMATYCZNYCH

- Testy jednostkowe
- Testy integracyjne
- Testy akceptacyjne
- Testy statystyczne
- Testy wydajnościowe

TESTY JEDNOSTKOWE

- Metoda testowania oprogramowania, która dla zadanych parametrów wejściowych bada poprawność danych wyjściowych.

- Testy jednostkowe często są nazywane testami czarnej skrzynki dlatego, że tester nie musi znać kodu zawartego w badanej funkcji. Dla testera ważny jest wyłącznie wynik wywołania funkcji – nie musi analizować kodu w niej zawartego.

WŁAŚCIWOŚCI TESTÓW JEDNOSTKOWYCH

- Automatyzacja
- Kompletność
- Powtarzalność
- Niezależność
- Łatwość utrzymania
- Czytelność

PIERWSZY TEST

```
[TestClass()]
public class MathTests
{
    [TestMethod()]
    public void AddTest()
    {
        decimal x = 4;
        decimal y = 3;//Arrange

        var result = Math.Add(x,y);//Act

        Assert.AreEqual(result,7);//Assert
    }
```

KLASA ASSERT

Metoda	Opis
AreEqual	Sprawdza czy dwie zmienne określonego typu są takie same, za pomocą operatora równości.
AreNotEqual	Sprawdza czy dwie zmienne określonego typu są różne.
AreSame	Sprawdza czy dwa zmienne referencyjne wskazują na ten sam obiekt
AreNotSame	Sprawdza czy dwa zmienne referencyjne wskazują na różne obiekty
Fail	Zwraca błąd bez sprawdzania żadnych warunków
Inconclusive	Wskazuje że metoda nie może zostać zweryfikowana.
IsFalse	Sprawdza czy zadany warunek ma wartość false wtedy metoda zwraca true.
IsTrue	Sprawdza czy zadany warunek ma wartość true wtedy metoda zwraca true.
IsInstanceOfType	Sprawdza czy obiekt jest żądane typu.
IsNull	Sprawdza czy obiekt jest nullem.
ThrowsException	Sprawdza czy działania rzuca ściśle zdefiniowany wyjątek.

PROGRAMOWANIE RÓWNOLEGŁE

- Współbieżność – wykonywanie wielu czynności jednocześnie.
- Przetwarzanie równoległe – Wykonanie zadanych zadań za pomocą wielu wątków.
- Programowanie asynchroniczne – pozwala wykonywać operacje odległe w czasie nie blokując głównego wątku aplikacji.

WĄTEK

- Jest jednostka odpowiedzialna za wykonanie odpowiedniego kodu.
- Wykonywany jest niezależnie i równocześnie z innymi wątkami
- Współdzieli z wątkami pamięć

WĄTKI

```
Thread thread = new Thread(StartCalculate);
thread.Start();
Console.WriteLine("No to ruszamy.");
```

USYPIANIE, PRZERYWANIE

```
Thread thread = new Thread(StartCalculate);
thread.Start();
Console.WriteLine("No to ruszamy.");
Thread.Sleep(300);
Console.WriteLine("Usypianie");
thread.Abort();
Console.WriteLine("Przerywamy.");
```

USYPIANIE, PRZERYWANIE

```
Thread thread = new Thread(StartCalculate);
thread.Start();
Console.WriteLine("No to ruszamy.");
Thread.Sleep(300);
Console.WriteLine("Usypianie");
thread.Abort();
Console.WriteLine("Przerywamy.");
```

WIELOWĄTKOWOŚĆ

```
Thread[] threadTable = new Thread[10];
for (int index=0; index < threadTable.Length; index++)
{
    threadTable[index] = new Thread(StartCalculate);
    threadTable[index].Start();
}
for (int index = 0; index < threadTable.Length; index++)
{
    threadTable[index].Join();
    Console.WriteLine("Wątek numer {0} zakończył działanie", threadTable[index].ManagedThreadId);
}
```

ZADANIA

- Warstwa abstrakcji ułatwiająca programowanie współbieżne (dla Thread)
- Zdefiniowana w System.Threading.Task
- Reprezentuje operację asynchroniczną

KLASA TASK

Metoda/Właściwość	Opis
AsyncState	Pobiera obiekt stanu podane podczas Task został utworzony lub wartość null, jeśli nie została podana.
CurrentId (statyczna)	Zwraca identyfikator aktualnie wykonywanych
Id	Zwraca identyfikator
IsCanceled	Sprawdza czy Task zakończył działanie z powodu anulowania
IsCompleted	Sprawdza czy Task zakończył działanie
IsFaulted	Sprawdza czy Task zakończył działanie z powodu nieobsługiwanego błędu
Status	Pobiera status Task (TaskStatus)
CreationOptions	Pobiera TaskCreationOptions użyty do utworzenia tego zadania.
Exception	Pobiera AggregateException powodujący Task przedwczesne zakończenie
Factory	Zapewnia dostęp do metod dotyczących tworzenia i konfigurowania Task.

KLASA TASK

```
List<Task> taskList = new List<Task>();

for (int index = 0; index < 15; index++)
{
    taskList.Add(new Task(StartCalculate));
}
taskList.ForEach(t => t.Start());
taskList.ForEach(t => t.Wait());
```

SYNCHRONIZACJA ZADAŃ

- WaitAll() sprawia że bieżący wątek oczekuje na wykonanie wszystkich zadań
- WaitAny() sprawia że bieżący wątek oczekuje na wykonanie jakiegolwiek zadania.

```
Task.WaitAll();
Console.WriteLine("Wszystkie wątki zakończyły działanie");
```

```
Task.WaitAny();
Console.WriteLine("Część wątków dalej działa.");
```

PRZERYWANIE ZADAŃ (CANCELLATIONTOKENSOURCE)

- zarządza i wysyła powiadomienie anulowanie tokenów anulowania indywidualnych.
- Token – właściwość przekazana do wątku która nasłuchuje anulowania.
- Token.IsCancellationRequested Metoda z działań, które odbierają token odwołania..
- Cancel Metoda zapewnis powiadomienia o anulowaniu. Powodujeustawienie CancellationToken.IsCancellationRequested na true

PRZERYWANIE ZADAŃ (CANCELLATIONTOKENSOURCE)

```
static CancellationTokenSource source = new CancellationTokenSource();
static CancellationToken cancellationToken = source.Token;

static void Main(string[] args)
{

    List<Task> taskList = new List<Task>();
    for (int index = 0; index < 15; index++)
    {
        taskList.Add(new Task(StartCalculate,cancellationToken));
    }
    taskList.ForEach(t => t.Start());
    Thread.Sleep(5000);
    source.Cancel();

}
```

KLASA PARALLEL

- Służy do zrównoleglenia wykonywanego kodu
- System.Threadings.Tasks
- Składa się ze statycznych metod klasy For, Foreach, Invoke

KLASA PARALEL (FOR)

```
Parallel.For(2, maxNumber, (action) =>
{
    if (((int)(action) % 5) == 0)
        Console.WriteLine("Liczba {0} jest podzielna przez 5;", (int)(action));
});
```

KLASA PARALEL (FOREACH)

```
int maxNumber = 99;
Parallel.ForEach(Enumerable.Range(2,maxNumber-1), (action) =>
{
    if (((int)(action) % 5) == 0)
        Console.WriteLine("Liczba {0} jest podzielna przez 5;", (int)(action));
});
```