Svar

Thursday, August 22, 2024

Link *til spørgsmål på itslearning*

Øvelse 1: Terminologi

Reflekter over begreberne:

- **"Lambda" = Lambda expression || Lambda funktion**
- **"Lambda expression" : Defineres med => og har form: (parameter liste) => Lambda funktionens krop**
- **"Lambda statement" : Statements sættes i Lambda funktionens krop, hvis flere: { sequence of statements}**
- **"Action" : Delegate der ikke returnerer en værdi**
- **"Func" : Delegate der returnerer en værdi**
- **"Predicate" : Delegate der returnerer en bool**
- **"Delegate" : Type der refererer til en metode med bestemt signatur og returtype**
    - **"Signatur" : parameter mængde (og type, hvis ikke generisk)**

Øvelse 2: Metodesignaturer og delegate-typer

Kig nøje på følgende 13 metodesignaturer:

```
1. double Calculate(string expression)
2. int CompareTo(int other)
3. int Factorial (int n)
4. void Delete(int index)
5. bool Exist(int id)
6. string GetUserChoice()
7. int Fibonacci(int n)
8. void NotifySubscribers(string message)
9. string GetAllPets()
10.   bool IsMatchesFinished()
11.   void ShowScore(string tournamentName)
12.   bool Contains(int id)
```

Udfør følgende:

- Definér så få delegate-typer som muligt, der beskriver alle 13 metodesignaturer. Find selv på navnene på de delegate-typer, du finder

```
//1
delegate double DoubleDelegateStringParam(string s);

//2, 3, 7
delegate int IntDelegateIntParam(int x);

//4
delegate void VoidDelegateIntParam(int x);

//5, 12
delegate bool BoolDelegateIntParam(int x);

//6, 9
```

```
delegate string StringDelegateNoParam();

//8, 11
delegate void VoidDelegateStringParam(string s);

//10
delegate bool BoolDelegateNoParam();
```

- Konvertér hver af de fundne delegate-typer til de indbyggede delegate-typer: Action, Func og Predicate. Nogle delegate-typer kan beskrives med mere end en af de indbyggede. Find dem alle

```
//1
delegate double doubleDelegateStringParam(string s);
Func<double, string> doubleDelegateStringParam = x;

//2, 3, 7
delegate int intDelegateIntParam(int x);
Func<int, int> intDelegateIntParam = x;

//4
delegate void voidDelegateIntParam(int x);
Action<int> voidDelegateIntParam = x;

//5, 12
delegate bool boolDelegateIntParam(int x);
Predicate<int> boolDelegateIntParam = x;
Predicate<T> DelegateGeneric = x;

//6, 9
delegate string stringDelegateNoParam();
Func<string> stringDelegateNoParam = x;

//8, 11
delegate void VoidDelegateStringParam(string s);
Action<string> voidDelegateStringParam = x;

//10
delegate bool BoolDelegateNoParam();
Func<bool> boolDelegateNoParam = x;
```

Øvelse 3: Anonyme delegates i BonusApp

Udfør følgende:

- Pak koden i BonusApp.Code.zip ud, og åbn løsningen i Visual Studio
- Opret et nyt testprojekt – MSTest Test Project.
- Kig koden igennem.

Øvelse 3.1: Konvertér til anonym delegate

Order-klassen indeholder en Bonus-property, som er erklæret med delegate-typen BonusProvider (defineret i Bonuses.cs). Det er denne property og dens type, du skal have fokus på i det følgende:

- Indsæt følgende testmetode:

```
[TestMethod]
public void GetBonusAnonymous_Test()
{
    order.Bonus = Bonuses.TenPercent;
    Assert.AreEqual(4.5, order.GetBonus());

    order.Bonus = Bonuses.FlatTwoIfAmountMoreThanFive;
    Assert.AreEqual(2.0, order.GetBonus());
}
```

- Tilret testmetoden, så Bonuses.TenPercent og Bonuses.FlatTwoIfAmountMoreThanFive (gult) ændres til en tilsvarende in-line anonym delegate

- Kør testen, så den gennemføres uden fejl

Anonyme delegates - (Samme som 5.1, men længere - bedre til mere komplekse kodekroppe, f.eks if statements osv.)

```
[TestMethod]
0 references
public void GetBonusAnonymous_Test_S()
{
    //Øvelse 3.1

    // Arrange
    Order order = new Order();
    order.AddProduct(new Product { Name = "product1", Value = 45.0 });

    // Act & Assert

    //order.Bonus = Bonuses.TenPercent;
    order.Bonus = delegate (double amount)
    {
        return amount / 10.0;
    };
    Assert.AreEqual(4.5, order.GetBonus());

    //order.Bonus = Bonuses.FlatTwoIfAmountMoreThanFive;
    order.Bonus = delegate (double amount)
    {
        return amount = amount > 5.0 ? 2.0 : 0.0;
    };
    Assert.AreEqual(2.0, order.GetBonus());
}
```

Øvelse 4: Anonyme delegates og lambda

Kig nøje på de følgende tre anonyme delegates:

1:
```
delegate (int x, int y)
{
    int result = x + y;
    return result;
};
```

2:
```
delegate (int x)
{
    int result = 0;
    for (int i = 0; i < 10; i++) {
        result += x;
    }
};
```

3:
```
delegate (int x)
{
    return x * x;
};
```

Udfør følgende:
Konvertér disse 3 anonyme metoder til deres tilsvarende lambda-udtryk eller lambda-sætninger.

1:
Func<int, int, int> add = delegate (int x, int y) { return x + y; };

2:
Action<int> addXtoX10times = delegate (int x) { int result = x * 10; };
2.1: OBS: 2 Skulle sikkert have en return og være en Func
Func<int, int> addXtoX10times = delegate (int x) { return x * 10; };

3:
Func<int, int> xSquared = delegate (int x) { return x * x; };

```
//1:
Func<int, int, int> add = delegate (int x, int y) { return x + y; };
//2:
Action<int> addXtoX10times = delegate (int x) { int result = x * 10; };
//2.1: OBS: Skulle sikkert have en return og være en Func
Func<int, int> addXtoX10times2 = delegate (int x) { return x * 10; };
//3:
Func<int, int> xSquared = delegate (int x) { return x * x; };
```

// Som lambda udtryk
1:
Func<int, int, int> add = (x, y) => x + y;
2:
Action<int> addXtoX10times = x => { int result = x * 10; };
Func<int, int> addXtoX10times = x => x * 10;
3:
Func<int,int> xSquared = x => x*x;

```
// Med Lambda udtryk
//1
Func<int, int, int> addLambda = (x, y) => x + y;
//2
Action<int> addXtoX10timesLambda = x => { int result = x * 10; };
//2.1: OBS: Skulle sikkert have en return og være en Func
Func<int, int> addXtoX10times2Lambda = x => x * 10;
//3
Func<int, int> xSquaredLambda = x => x * x;
```

Øvelse 5.1: Konvertér til Lambda

Udfør følgende:

- Indsæt følgende testmetode i BonusApp-testprojektet:

```
[TestMethod]
public void GetBonusLambda_Test()
{
      order.Bonos = Bonuses.TenPercent; // Lav til lambda udtryk
      Assert.AreEqual(4.5, order.GetBonus());
      order.Bonus = Bonuses.FlatTwoIfAmountMoreThanFive; // Lav til
lambda udtryk
      Assert.AreEqual(2.0, order.GetBonus());
}
```

- Tilret testmetoden i testen, så Bonuses.TenPercent og Bonuses.FlatTwoIfAmountMoreThanFive (gult) ændres til et tilsvarende lambda-udtryk
- Kør testen, så den gennemføres uden fejl

Lambda udtryk - (Samme som anynome i 3.1, men kortere - bedre til korte metoder)

```
[TestMethod]
0 references
public void GetBonusLambda_Test_S()
{
    //Øvelse 5.1

    // Arrange
    Order order = new Order();
    order.AddProduct(new Product { Name = "product1", Value = 45.0 });

    //order.Bonus = Bonuses.TenPercent; // Lav til lambda udtryk
    order.Bonus = amount => amount / 10;
    Assert.AreEqual(4.5, order.GetBonus());
    //order.Bonus = Bonuses.FlatTwoIfAmountMoreThanFive; // Lav til lambda udtryk
    order.Bonus = amount => amount > 5.0 ? 2.0 : 0.0;
    Assert.AreEqual(2.0, order.GetBonus());
}
```

Øvelse 5.2: Lambda-udtryk i GetBonus-parameter

I den udleverede kode beregner GetBonus()-metoden (i Order-klassen) den samlede bonus for en ordre ved at kalde Order-klassens delegate Bonus af typen BonusProvider.

Du skal nu udvikle en alternativ måde at angive en bonusberegning helt uden brug af Order-klassens Bonus-property, men hvor du stadig bruger delegate-type BonusProvider. I stedet skal du gøre det muligt at angive en bonusberegning via en delegate-parameter til GetBonus().

Udfør følgende:

- Overload GetBonus() med metoden GetBonus(<delegate-type> <parameter-navn>), hvor det er muligt at angive bonusberegningen i parameteren
- Implementér metodens krop, så den anvender parameteren til at udføre bonusberegningen i stedet for klassens Bonus-property

```
7 references | 3/3 passing
public double GetBonus()
{
    return Bonus(GetValueOfProducts());
}

5 references | 2/2 passing
public double GetBonus(Func<double, double> bonusProvider)
{
    return bonusProvider(GetValueOfProducts());
}
```

- Indsæt testmetoden GetBonusByLambdaParameter_Test() forneden til testprojektet:

[TestMethod]
`public void GetBonusByLambdaParameter_Test()`

```
{
    // Anvend TenPercent lambda expression som parameter til
GetBonus
    Assert.AreEqual(4.5, order.GetBonus(Bonuses.TenPercent));
    // Anvend FlatTwoAmountMoreThanFive lambda expression som
parameter til GetBonus
    Assert.AreEqual(2.0,
order.GetBous(Bonuses.FlatTwoIfAmountMoreThanFive));
}
```

- Erstat henholdsvis Bonuses.TenPercent og Bonuses.FlatTwoIfAmountMoreThanFive (gult) i testmetoden med de tilsvarende lambda-udtryk, I har fundet frem til

- Kør testen, og tilret om nødvendigt

```
[TestMethod]
0 references
public void GetBonusByLambdaParameter_Test_S()
{
    // Arrange
    Order order = new Order();
    order.AddProduct(new Product { Name = "product1", Value = 45.0 });

    // Anvend TenPercent lambda expression som parameter til GetBonus
    Assert.AreEqual(4.5, order.GetBonus(amount => amount / 10));
    // Anvend FlatTwoAmountMoreThanFive lambda expression som parameter til GetBonus
    Assert.AreEqual(2.0, order.GetBonus(amount => amount > 5.0 ? 2.0 : 0.0));
}
```

Predicate

Som i sikkert ved fra typerne **Func** og **Action,** repræsenterer delegater en reference til en metode. Delegater bruges til at definere metoder, der kan kaldes gennem en variabel af delegattypen, hvilket muliggør en fleksibel og udvidelsesbar måde at kalde metoder på.

Predicate er en specifik type delegate i C#, som bruges til at definere metoder, der tager et input af en bestemt type og returnerer en boolsk værdi (*true* eller *false*). Denne boolske værdi bruges typisk til at bestemme, om et givet element opfylder et bestemt kriterium.

En Predicate<T>-delegate repræsenterer en metode, der tager en parameter af typen T og returnerer en bool.

Typisk bruges Predicate-delegater til at filtrere eller søge i samlinger af objekter.
Her er et simpelt eksempel på brugen af Predicate type:

```
using System;
using System.Collections.Generic;

public class Program
{
    // Predicate metode, der tjekker om et tal er ulige
    static bool IsOdd(int number)
    {
        return number % 2 != 0;
    }

    public static void Main()
    {
        List<int> numbers = new List<int> { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };

        // Brug af Predicate delegate for at finde alle ulige tal
        Predicate<int> isOdd = IsOdd;
        List<int> oddNumbers = numbers.FindAll(isOdd);

        Console.WriteLine("Ulige tal:");
        foreach (int number in oddNumbers)
        {
            Console.WriteLine(number);
        }
    }
}

```

Forklaring:
I dette eksempel definerer vi en liste af heltal.

- `Predicate<int> isOdd` er en delegat, der refererer til metoden `IsOdd`.
- `IsOdd` er en metode, der tager en int som input og returnerer true, hvis tallet er ulige, ellers false.
- Metoden `List<int>.FindAll()` bruger Predicate-delegaten isOdd til at finde alle elementer i listen, der opfylder betingelsen.
  Dette resulterer i en liste af ulige tal, som derefter printes til konsollen.

Øvelse 5.3: Action, Func og Predicate i GetBonus()

I stedet for at angive delegate-typen BonusProvider kan man bruge en af de allerede indbyggede delegate-typer Action, Func eller Predicate.

- Overvej hvorvidt det er Action, Func eller Predicate, som kan anvendes i stedet for BonusProvider
- Tilret `GetBonus(<delegate-type> <parameter-navn>)`-metoden, som I udviklede i forrige øvelse 5.2, så den anvender den valgte indbyggede delegate-type, som præcis erstatter BonusProvider
- Kør testen (fra øvelse 5.2) igen for at sikre dig, at rettelsen fungerer korrekt

```
5 references | 2/2 passing
public double GetBonus()
{
    return Bonus(GetValueOfProducts());
}

3 references | 1/1 passing
public double GetBonus(Func<double, double> bonusProvider)
{
    return bonusProvider(GetValueOfProducts());
}
```

Øvelse 5.4: Action, Func og Predicate i GetTotalPrice()

Metoden GetTotalPrice() er den eneste anden metode i Order-klassen, som anvender GetBonus().

Udfør følgende:

- Overload GetTotalPrice() med GetTotalPrice(<delegate-type> <parameter-navn>), hvor man kan angive bonusberegningen som parameter. Parameterens type skal være den indbyggede delegate-type, du har valgt foroven i øvelse 5.3

- GetTotalPrice(<delegate-type> <parameter-navn>) må ikke anvende klassens Bonus-property, så du skal implementere metodens krop, så den anvender GetBonus(<delegate-type> <parameter-navn>)

```
0 references
public double GetTotalPrice()
{
    return GetValueOfProducts() - GetBonus();
}

2 references | 0/1 passing
public double GetTotalPrice(Func<double, double> bonusProvider)
{
    return GetValueOfProducts() - GetBonus(bonusProvider);
}
```

- Indsæt testmetoden GetTotalPriceByLambdaParameter_Test() forneden til testprojektet:

```
[TestMethod]
public void GetTotalPriceByLambdaParameter_Test()
{
     Assert.AreEqual(40.5,
order.GetTotalPrice(Bonuses.TenPercent));
     Assert.AreEqual(43.0,
order.GetTotalPrice(Bonuses.FlatTwoIfAmountMoreThanFive));
}
```

- Erstat henholdsvis Bonuses.TenPercent og Bonuses.FlatTwoIfAmountMoreThanFive (gult) i testmetoden med de tilsvarende lambda-udtryk, du har fundet frem til
- Kør testen, og tilret om nødvendigt