# **Delegates, Events and Lambda**

# **Delegates**

# **(Делегати)**

***Делегат*** - це тип, який представляє собою посилання на методи з певним списком параметрів і типом повернення. Коли ви створюєте делегат, ви можете зв'язати його екземпляр з будь-яким методом із сумісною сигнатурою та типом повернення. Ви можете викликати метод через екземпляр делегату.

Делегати репрезентують такі об'єкти, які вказують на методи. Тобто делегати - це вказівники на методи і за допомогою делегатів ми можемо викликати ці методи.

Делегати використовуються для передачі методів як аргументів іншим методам. ***Обробники подій (Event handler)*** - це ніщо інше, як методи, що викликаються через делегати. Ви створюєте користувацький метод, і клас, наприклад, елемент управління вікна, може викликати ваш метод, коли відбувається певна подія.

Ось приклад коду, який показує, як створити делегата і використати його:

// Оголошення делегата з двома параметрами типу int і поверненням значення типу int

delegate int CalculationDelegate(int a, int b);

// Оголошення методу, який відповідає сигнатурі делегата

static int Add(int a, int b)

{

return a + b;

}

static void Main(string[] args)

{

// Створення екземпляру делегата та призначення йому методу Add

CalculationDelegate calc = new CalculationDelegate(Add);

// Виклик методу Add за допомогою делегата

int result = calc(2, 3);

Console.WriteLine(result); // Виводиться 5

}

В цьому прикладі ми оголосили делегат CalculationDelegate, який приймає два параметри типу int і повертає значення типу int. Ми також оголосили метод Add, який відповідає сигнатурі делегата.

У методі PerformCalculation ми приймаємо два параметри типу int і делегат типу CalculationDelegate, і викликаємо переданий метод з параметрами a та b.

У методі Main ми створюємо екземпляр делегата calc, який вказує на метод Add. За допомогою методу PerformCalculation ми викликаємо метод Add з параметрами 2 та 3, і отримуємо результат 5.

delegate int Operation(int x, int y);

int Add(int x, int y) => x + y;

int Multiply(int x, int y) => x \* y;

Operation operation = Add; // делегат указывает на метод Add

int result = operation(4, 5); // фактически Add(4, 5)

Console.WriteLine(result); // 9

operation = Multiply; // теперь делегат указывает на метод Multiply

result = operation(4, 5); // фактически Multiply(4, 5)

Console.WriteLine(result); // 20

У цьому випадку делегат ***Operation*** повертає значення типу int і має два параметри типу int. Тому цьому делегату відповідає будь-який метод, який повертає значення типу int і приймає два параметри типу int. У цьому випадку це методи ***Add і Multiply***. Тобто ми можемо присвоїти змінній делегата будь-який із цих методів і викликати.

## **Delegates Overview:**

* Делегати дозволяють передавати методи як параметри.
* Делегати можна використовувати для визначення методів зворотного виклику.
* Делегати можна об'єднувати в ланцюжок; наприклад, декілька методів можна викликати на одну подію.
* Методи не обов'язково повинні точно відповідати типу делегата. Для отримання додаткової інформації див. статтю Використання відхилень у делегатах.
* Лямбда-вирази - це більш стислий спосіб написання вбудованих блоків коду. Лямбда-вирази (у певних контекстах) компілюються до типів делегатів. Докладні відомості про лямбда-вирази див. у статті Лямбда-вирази.

# **Lambda expressions**

# **(Лямбда-вирази)**

Ви використовуєте лямбда-вираз для створення анонімної функції. Використовуйте оператор оголошення лямбда-виразу ***=>***, щоб відокремити список параметрів лямбда-виразу від його тіла.

Лямбда-вирази представляють спрощений запис анонімних методів. Лямбда-вирази дають змогу створити ємні лаконічні методи, які можуть повертати певне значення і які можна передати як параметри в інші методи.

Лямбда-вираз може мати одну з двох наступних форм:

* ***Лямбда виразу (Expression lambda)***, яка має в якості тіла вираз:

***(input-parameters) => expression***

* ***Лямбда операторів (Statement lambda)***, яка містить блок операторів у своєму тілі:

***(input-parameters) => { <sequence-of-statements> }***

Щоб створити лямбда-вираз, ви вказуєте вхідні параметри (якщо такі є) в лівій частині лямбда-оператора і вираз або блок операторів з іншого боку.

***Expression lambdas:***

Лямбда-вираз з виразом у правій частині оператора ***=>*** називається лямбда-вираз. Лямбда-вираз повертає результат виразу і має наступну основну форму:

***(input-parameters) => expression***

The body of an expression lambda can consist of a method call. However, if you're creating expression trees that are evaluated outside the context of the .NET Common Language Runtime (CLR), such as in SQL Server, you shouldn't use method calls in lambda expressions. The methods have no meaning outside the context of the .NET Common Language Runtime (CLR).

***Statement lambdas:***

Лямбда операторів схожа на лямбду виразів, за винятком того, що її оператори беруться у фігурні дужки:

***(input-parameters) => { <sequence-of-statements> }***

Тіло лямбда-оператора може складатися з будь-якої кількості операторів, однак на практиці їх зазвичай не більше двох-трьох.

## **Events**

## **(Події)**

Подія - це повідомлення, надіслане об'єктом для сигналізації про виконання певної дії. Дія може бути спричинена взаємодією користувача, наприклад, натисканням кнопки, або може бути результатом іншої програмної логіки, наприклад, зміною значення властивості. Об'єкт, який викликає подію, називається ***відправником події (event sender)***. Відправник події не знає, який об'єкт або метод отримає (обробить) згенеровану ним подію. Зазвичай подія є членом відправника події; наприклад, подія ***Click*** є членом класу ***Button***, а подія ***PropertyChanged*** є членом класу, який реалізує інтерфейс ***INotifyPropertyChanged***.

Зазвичай, щоб викликати подію, ви додаєте метод, який позначений як ***protected*** та ***virtual*** (у C#) або ***Protected*** and ***Overridable*** (у Visual Basic).

Події сигналізують системі про те, що відбулася певна дія. І якщо нам треба відстежити ці дії, то якраз ми можемо застосовувати події.

Події оголошуються в класі за допомогою ключового слова ***event***, після якого вказується тип делегата, який представляє подію:

delegate void AccountHandler(string message);

***event*** AccountHandler Notify;

У цьому випадку спочатку визначається делегат ***AccountHandler***, який приймає один параметр типу ***string***. Потім за допомогою ключового слова ***event*** визначається подія з ім'ям ***Notify***, яка представляє делегат ***AccountHandler***. Назва для події може бути довільною, але в будь-якому разі вона має представляти деякий делегат.

Визначивши подію, ми можемо її викликати в програмі як метод, використовуючи ім'я події:

Notify("Произошло действие");

Оскільки подія ***Notify*** представляє делегат ***AccountHandler***, який приймає один параметр типу ***string*** - рядок, то під час виклику події нам треба передати в нього рядок.

Однак під час виклику подій ми можемо зіткнутися з тим, що подія дорівнює ***null*** у разі, якщо для неї не визначено обробника. Тому під час виклику події краще її завжди перевіряти на ***null***. Наприклад, так:

if(Notify !=null) Notify("Произошло действие");

Або так:

Notify?.Invoke("Произошло действие");

У цьому випадку оскільки подія представляє делегат, то ми можемо її викликати за допомогою методу ***Invoke()***, передавши в нього необхідні значення для параметрів.

Об'єднаємо все разом і створимо та викличемо подію:

class Account

{

public ***delegate*** void AccountHandler(string message);

public ***event*** AccountHandler? Notify; // 1.Определение события

public Account(int sum) => Sum = sum;

public int Sum { get; private set; }

public void Put(int sum)

{

Sum += sum;

Notify?.Invoke($"На счет поступило: {sum}"); // 2.Вызов события

}

public void Take(int sum)

{

if (Sum >= sum)

{

Sum -= sum;

Notify?.Invoke($"Со счета снято: {sum}"); // 2.Вызов события

}

else

{

Notify?.Invoke($"Недостаточно денег на счете. Текущий баланс: {Sum}"); ;

}

}

}

static void Main(string[] args)

{

Account account = new Account(100);

account.Notify += DisplayMessage; // Добавляем обработчик для события Notify

account.Put(20); // добавляем на счет 20

Console.WriteLine($"Сумма на счете: {account.Sum}");

account.Take(70); // пытаемся снять со счета 70

Console.WriteLine($"Сумма на счете: {account.Sum}");

account.Take(180); // пытаемся снять со счета 180

Console.WriteLine($"Сумма на счете: {account.Sum}");

void DisplayMessage(string message) => Console.WriteLine(message);

}

Тепер за допомогою події ***Notify*** ми повідомляємо систему про те, що було додано кошти і про те, що кошти знято з рахунку або на рахунку недостатньо коштів.

## **Events Overview:**

* У C# події (Events) - це механізм, який дозволяє класам повідомляти про те, що відбувається в їхньому внутрішньому стані. Інші класи можуть підписатись на події, щоб отримувати сповіщення, коли стан об'єкта змінюється.
* Події дозволяють розділити логіку обробки подій від логіки, що породжує події. Це дозволяє розширювати функціональність класу, не залежно від того, які інші класи використовують його події.
* Оголошення події в C# складається з двох частин: елемента-члена event і делегата, що визначає тип обробника подій.

public class MyClass

{

public event EventHandler MyEvent;

public void DoSomething()

{

// Метод, що породжує подію

OnMyEvent();

}

protected virtual void OnMyEvent()

{

// Виклик обробників події

MyEvent?.Invoke(this, EventArgs.Empty);

}

}