# **Delegates**

**Додавання методів у делегат**

У прикладах вище змінна делегата вказувала на один метод. У реальності ж делегат може вказувати на безліч методів, які мають ту саму сигнатуру і тип, що повертаються. Усі методи в делегаті потрапляють у спеціальний список - список виклику або invocation list. І при виклику делегата всі методи з цього списку послідовно викликаються. І ми можемо додавати в цей список не один, а кілька методів. Для додавання методів у делегат застосовується операція ***+=***:

delegate void Message();

Message message = Hello;

message += HowAreYou; // теперь message указывает на два метода

message(); // вызываются оба метода - Hello и HowAreYou

void Hello() => Console.WriteLine("Hello");

void HowAreYou() => Console.WriteLine("How are you?");

У цьому випадку до списку виклику делегата message додаються два методи - ***Hello*** і ***HowAreYou***. І під час виклику ***message*** викликаються відразу обидва ці методи.

Однак варто зазначити, що в реальності відбуватиметься створення нового об'єкта делегата, який отримає методи старої копії делегата і новий метод, і новий створений об'єкт делегата буде присвоєно змінній ***message***.

Під час додавання делегатів слід враховувати, що ми можемо додати посилання на один і той самий метод кілька разів, і в списку виклику делегата тоді буде кілька посилань на один і той самий метод. Відповідно під час виклику делегата доданий метод буде викликатися стільки разів, скільки його було додано.

Консольний вивід:

***Hello***

***How are you?***

**Виклик делегата**

У прикладах вище делегат викликався як звичайний метод. Якщо делегат приймав параметри, то під час його виклику для параметрів передавалися необхідні значення:

Message mes = Hello;

mes();

Operation op = Add;

int n = op(3, 4);

Console.WriteLine(n);

void Hello() => Console.WriteLine("Hello");

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

delegate int Operation(int x, int y);

delegate void Message();

Інший спосіб виклику делегата представляє метод ***Invoke()***:

Message mes = Hello;

mes.Invoke(); // Hello

Operation op = Add;

int n = op.Invoke(3, 4);

Console.WriteLine(n); // 7

void Hello() => Console.WriteLine("Hello");

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

delegate int Operation(int x, int y);

delegate void Message();

Якщо делегат приймає параметри, то в метод Invoke передаються значення для цих параметрів.

Слід враховувати, що якщо делегат порожній, тобто в його списку виклику немає посилань на жоден із методів (тобто делегат дорівнює Null), то під час виклику такого делегата ми отримаємо виняток.

**Узагальнені делегати**

Делегати, як і інші типи, можуть бути узагальненими, наприклад:

Operation<decimal, int> squareOperation = Square;

decimal result1 = squareOperation(5);

Console.WriteLine(result1); // 25

Operation<int, int> doubleOperation = Double;

int result2 = doubleOperation(5);

Console.WriteLine(result2); // 10

decimal Square(int n) => n \* n;

int Double(int n) => n + n;

delegate T Operation<T, K>(K val);

Тут делегат Operation типізується двома параметрами типів. Параметр ***T*** представляє тип значення, що повертається. А параметр ***K*** представляє тип переданого в делегат параметра. Таким чином, цьому делегату відповідає метод, який приймає параметр будь-якого типу і повертає значення будь-якого типу.

**Делегати як параметри методів**

Також делегати можуть бути параметрами методів. Завдяки цьому один метод як параметри може отримувати дії - інші методи.

Наприклад:

DoOperation(5, 4, Add); // 9

DoOperation(5, 4, Subtract); // 1

DoOperation(5, 4, Multiply); // 20

void DoOperation(int a, int b, Operation op)

{

Console.WriteLine(op(a,b));

}

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

int Subtract(int x, int y) => x - y;

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

delegate int Operation(int x, int y);

Тут метод ***DoOperation*** як параметри приймає два числа і деяку дію у вигляді делегата Operation. Всередині методу викликаємо делегат ***Operation***, передаючи йому числа з перших двох параметрів.

При виклику методу ***DoOperation*** ми можемо передати в нього як третій параметр метод, який відповідає делегату ***Operation***.

**Повернення делегатів із методу**

Також делегати можна повертати з методів. Тобто ми можемо повертати з методу якусь дію у вигляді іншого методу.

Наприклад:

Operation operation = SelectOperation(OperationType.Add);

Console.WriteLine(operation(10, 4)); // 14

operation = SelectOperation(OperationType.Subtract);

Console.WriteLine(operation(10, 4)); // 6

operation = SelectOperation(OperationType.Multiply);

Console.WriteLine(operation(10, 4)); // 40

Operation SelectOperation(OperationType opType)

{

switch (opType)

{

case OperationType.Add: return Add;

case OperationType.Subtract: return Subtract;

default: return Multiply;

}

}

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

int Subtract(int x, int y) => x - y;

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

enum OperationType

{

Add, Subtract, Multiply

}

delegate int Operation(int x, int y);

У цьому випадку метод ***SelectOperation()*** як параметр приймає перерахування типу OperationType. Це перерахування зберігає три константи, кожна з яких відповідає певній арифметичній операції. І в самому методі залежно від значення параметра повертаємо певний метод. Причому оскільки тип методу, що повертається, - делегат Operation, то метод має повернути метод, який відповідає цьому делегату - у нашому випадку це методи ***Add***, ***Subtract***, ***Multiply***.

**Додавання та видалення методів у делегаті**

Хоча в прикладі наш делегат приймав адресу на один метод, насправді він може вказувати одразу на кілька методів. Крім того, за потреби ми можемо видалити посилання на адреси певних методів, щоб вони не викликалися під час виклику делегата. Отже, змінимо в класі ***Account*** метод ***RegisterHandler*** і додамо новий метод ***UnregisterHandler***, який видалятиме методи зі списку методів делегата:

public delegate void AccountHandler(string message);

public class Account

{

int sum;

AccountHandler? taken;

public Account(int sum) => this.sum = sum;

// Регистрируем делегат

public void RegisterHandler(AccountHandler del)

{

taken += del;

}

// Отмена регистрации делегата

public void UnregisterHandler(AccountHandler del)

{

taken -= del; // удаляем делегат

}

public void Add(int sum) => this.sum += sum;

public void Take(int sum)

{

if (this.sum >= sum)

{

this.sum -= sum;

taken?.Invoke($"Со счета списано {sum} у.е.");

}

else

taken?.Invoke($"Недостаточно средств. Баланс: {this.sum} у.е.");

}

}

У першому методі об'єднує делегати ***taken*** і ***del*** в один, який потім присвоюється змінній ***taken***. У другому методі зі змінної taken видаляється делегат ***del***.

Застосуємо нові методи:

Account account = new Account(200);

// Добавляем в делегат ссылку на методы

account.RegisterHandler(PrintSimpleMessage);

account.RegisterHandler(PrintColorMessage);

// Два раза подряд пытаемся снять деньги

account.Take(100);

account.Take(150);

// Удаляем делегат

account.UnregisterHandler(PrintColorMessage);

// снова пытаемся снять деньги

account.Take(50);

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

void PrintColorMessage(string message)

{

// Устанавливаем красный цвет символов

Console.ForegroundColor = ConsoleColor.Red;

Console.WriteLine(message);

// Сбрасываем настройки цвета

Console.ResetColor();

}

З метою тестування ми створили ще один метод - ***PrintColorMessage***, який виводить те саме повідомлення тільки червоним кольором. Посилання на цей метод також передається в метод ***RegisterHandler***, і таким чином його отримає змінна ***taken***.

У рядку ***account.UnregisterHandler(PrintColorMessage);*** цей метод видаляється зі списку викликів делегата, тому цей метод більше не спрацьовуватиме. Консольний вивід матиме таку форму:

Со счета списано 100 у.е.

Со счета списано 100 у.е.

Недостаточно средств. Баланс: 100 у.е.

Недостаточно средств. Баланс: 100 у.е.

Со счета списано 50 у.