**Iteratorsта оператор yield**

# ***Iterators*** можна використовувати для перебору колекцій, таких як списки та масиви.

Метод ітератора або аксесор ***get*** виконує власну ітерацію над колекцією. Метод-ітератор використовує оператор повернення ***yield*** для повернення кожного елемента по одному за раз. Коли досягається оператор ***return***, запам'ятовується поточне місце у коді. Виконання буде перезапущено з цього місця при наступному виклику функції-ітератора.

Оператор ***yield*** використовується в ітераторі для отримання наступного значення з послідовності під час її повторення.

s

Ітератор по суті є блоком коду, який використовує оператор ***yield*** для перебору набір значень. Цей блок коду може представляти тіло методу, оператора або блок ***get*** у властивостях.

Ітератор використовує дві спеціальні інструкції:

* ***yield return*** : визначає елемент, що повертається
* ***yield break*** : вказує, що послідовність більше немає елементів

Розглянемо невеликий приклад:

Numbers numbers = new Numbers();

foreach (int n in numbers)

{

Console.WriteLine(n);

}

class Numbers

{

public IEnumerator<int> GetEnumerator()

{

for (int i = 0; i < 6; i++)

{

yield return i \* i;

}

}

}

У класі ***Numbers*** метод ***GetEnumerator()*** фактично представляє ітератор. За допомогою оператора ***yield return*** повертається деяке значення (у разі квадрат числа).

У програмі за допомогою циклу ***foreach*** ми можемо перебрати об'єкт ***Numbers*** як звичайну колекцію. При отриманні кожного елемента в циклі ***foreach*** спрацьовуватиме оператор ***yield return***, який повертатиме один елемент і запам'ятовуватиме поточну позицію.

Завдяки ітераторам ми можемо піти далі та легко реалізувати перебір числа у циклі ***foreach***:

foreach(var n in 5) Console.WriteLine(n);

foreach (var n in -5) Console.WriteLine(n);

static class Int32Extension

{

public static IEnumerator<int> GetEnumerator(this int number)

{

int k = (number > 0)? number: 0;

for (int i = number - k; i <= k; i++) yield return i;

}

}

У цьому випадку ітератор реалізований як метод розширення типу int або ***System.Int32.*** У методі ітератора фактично повертаємо всі цілі значення від 0 до поточного числа. Консольний висновок:

0

1

2

3

4

5

-5

-4

-3

-2

-1

0

Інший приклад: нехай у нас є колекція ***Company***, яка представляє компанію та яка зберігає в масиві personnel штат співробітників – об'єктів ***Person***. Використовуємо оператор ***yield*** для перебору цієї колекції:

class Person

{

public string Name { get; }

public Person(string name) =>Name = name;

}

class Company

{

Person[] personnel;

public Company(Person[] personnel) => this.personnel = personnel;

public int Length => personnel.Length;

public IEnumerator<Person> GetEnumerator()

{

for (int i = 0; i < personnel.Length; i++)

{

yield return personnel[i];

}

}

}

Метод ***GetEnumerator()*** представляє ітератор. І коли ми будемо здійснювати перебір в об'єкті ***Company*** у циклі ***foreach***, то буде йти звернення до виклику ***yield return*** ***personnel[i];***

При звернення до оператора ***yield return*** зберігатиметься поточне розташування. І коли метод foreach перейде до наступної ітерації Для отримання нового об'єкта ітератор почне виконання з цього місця розташування.

Ну і в основній програмі в циклі ***foreach*** виконується власне перебір завдяки реалізації ітератора:

var people = new Person[]

{

new Person("Tom"),

new Person("Bob"),

new Person("Sam")

};

var microsoft = new Company(people);

foreach(Person employee in microsoft)

{

Console.WriteLine(employee.Name);

}

Хоча при реалізації ітератора у методі ***GetEnumerator()*** застосовувався перебір масиву в циклі ***for***, але це не обов'язково робити. Ми можемо просто визначити кілька дзвінків оператора yield ***return***:

public IEnumerator<Person> GetEnumerator()

{

yield return personnel[0];

yield return personnel[1];

yield return personnel[2];

}

У такому разі при кожному дзвінку оператора ***yield return*** ітератор також запам'ятовуватиме поточне місцезнаходження та при наступних викликах починатиме з нього.

**Іменований ітератор**

Вище для створення ітератора ми використали метод GetEnumerator. Але оператор ***yield*** можна використовувати всередині будь-якого методу, тільки такий метод повинен повертати об'єкт інтерфейсу ***IEnumerable***. Подібні методи ще називають іменованими ітераторами .

Створимо такий іменований ітератор у класі ***Company*** та використовуємо його:

class Person

{

public string Name { get; }

public Person(string name) =>Name = name;

}

class Company

{

Person[] personnel;

public Company(Person[] personnel) => this.personnel = personnel;

public int Length => personnel.Length;

public IEnumerable<Person> GetPersonnel(int max)

{

for (int i = 0; i < max; i++)

{

if (i == personnel.Length)

{

yield break;

}

else

{

yield return personnel[i];

}

}

}

}

Визначений тут ітератор – метод ***IEnumerable GetPersonnel(int max)*** як параметр приймає кількість об'єктів, що виводяться. У процесі роботи програми може скластися, що його значення буде більшим, ніж довжина масиву особи. І щоб не сталося помилки, використовується оператор yield break . Цей оператор перериває виконання ітератора.

Застосування ітератора:

var people = new Person[]

{

new Person("Tom"),

new Person("Bob"),

new Person("Sam")

};

var microsoft = new Company(people);

foreach(Person employee in microsoft.GetPersonnel(5))

{

Console.WriteLine(employee.Name);

}

Виклик ***microsoft.GetPersonnel(5)*** повертатиме набір з не більше ніж 5 об'єктів ***Person***. Але так як у нас всього три таких об'єкти, то в методі ***GetPersonnel*** після трьох операцій спрацює оператор ***yield break.***