В.В. Подбельский

Иллюстрации к курсу лекций по дисциплине «Программирование»

Лямбда-выражения в методах класса Array

Анонимный метод и лямбда выражение

```
// Объявление делегата-типа:
delegate int MyDel( int z );

// Анонимный метод в объявлении экземпляра делегата:
MyDel del = delegate(int x)   { return x * x; } ;

// Лямбда выражение в объявлении экземпляра делегата:
MyDel del2 =   (int x) => { return x * x; } ;
```

Синтаксис лямбда выражений

Синтаксис полного объявления лямбда-выражения:

(*спецификация параметров*) => { *операторы* }

```
delegate int MyDel( int z );   // Объявление делегата-типа
```

Соответствующие делегату MyDel **лямбда-выражения**

С телом-оператором:

```
(int x) =>   { return x * x; } ;
  (x)   =>   { return 2 * x ; } ;
   x    =>   { return x + 1; } ;
```

С телом в виде выражения:

```
x    =>   x * 10;
```

```
using System;
class Lambda  {
delegate bool Del(int i);
static void Main()   {
    Del del = new Del((z) => z > 0 ? true : false);
    Console.WriteLine("Sign: del(-5) = " + del(-5));
    del = c => { return c % 2 == 0 ? true : false; } ;
    Console.WriteLine("Parity: del(4) = " + del(4));
    }
}
```

Результат выполнения программы:

Sign: del(-5) = False

Parity: del(4) = True

```
class Params_Lambda  {
delegate void myDel(params int[] g);
static void Main( )   {
    int prod = 1;
    myDel mydel = new myDel((int[] g) =>
    {   int sum = 0;
        foreach (int f in g)
            { sum += f; prod *= f; }
        Console.WriteLine("Сумма
параметров = " + sum);
    });
    mydel(5,1,2,3);
}   }
```

Результат выполнения программы:

Сумма параметров = 11

Произведение параметров = 30

```
class Captured   {
delegate void DelegateType(int Param);
static void Main()  {
DelegateType variable;
{   // Внутренний блок и локальная переменная:
    int multy = 3;
    variable = x =>
    Console.WriteLine("multy = {0}; x = {1}; x*multy = {2}",
        multy, x, x*multy);
}   // Конец блока существования переменной multy.
    int p = 5;
    variable(p); // Вычисление лямбда-выражения
}   }
```

Результат выполнения программы:

multy = 3; x = 5; x*multy = 15

Захват изменяемых переменных

```
delegate void Del( );
static void Main( ) {
   Del [ ] arDel = new Del[3];
   for (int k = 0; k < arDel.Length; k++)
      arDel[k] = () => Console.Write(k+"\t");
   foreach(Del d in arDel)
      d();

/* Результаты:
3   3   3
*/
```

Захват временных изменяемых переменных

```
delegate void Del( );
. . . .
Del [] arDel = new Del[3];
. . . .
for (int k = 0; k < arDel.Length; k++) {
        int temp = k;
        arDel[k] = () => Console.Write(temp+"\t");
        }
    foreach(Del d in arDel)
        d();

/* Результаты:
0   1   2
*/
```

Захват изменяемых переменных в foreach

```
delegate void Del( );
. . . .
Del [] arDel = new Del[3];
. . . .
foreach (int j in new int [] {1,2,3})
        arDel[j-1] = () => Console.Write(j+"\t");
foreach(Del d in arDel)
        d();

/* Результаты:
1   2   3
*/
```

Некоторые статические методы класса Array с параметрами-делегатами

Array.ConvertAll( )

Array.ForEach( )

Array.Sort( )

Array.Find( )

Array.FindAll( )

Оператор цикла foreach предназначен для перебора элементов коллекций, реализующих интерфейсы:

System.Collections.IEnumerable или
System.Collections.Generic.IEnumerable<T>

Метод Array.ConvertAll

```
public static TOutput[] ConvertAll<TInput, TOutput>(
    TInput[] array,
    Converter<TInput, TOutput> converter )
```

TInput, Toutput – типизирующие параметры;

Converter<TInput, TOutput> - обобщенный делегат-тип:

```
public delegate TOutput Converter
    <in TInput, out TOutput> (TInput input)
```

Применение лямбда-выражения в Array.ConvertAll ( )

```
int[ ] fib = new int[ ] { 0, 1, 1, 2, 3, 5, 8, 13 };
double[ ] del = Array.ConvertAll(fib, e => e/10.0);
foreach (double elem in del)
    System.Console.Write(elem+"\t");
```

Результат:

0 0,1 0,1 0,2 0,3 0,5 0,8 1,3

Метод Array.ForEach( )

public static void ForEach<T>(T[] array,
Action<T> action)

T – типизирующий параметр;

Action<T> - обобщенный делегат-тип:

public delegate void Action<**in** T>(T obj)

Примеры использования метода Array.ForEach( )

```
int[] fib = new int[] { 0, 1, 1, 2, 3, 5, 8, 13 };
Array.ForEach(fib, Console.Write);
```

Результат выполнения:

011235813

```
Array.ForEach(fib, y => Console.Write(y+"\t"));
```

Результат выполнения:

0 1 1 2 3 5 8 13

Метод Array.Sort( )

public static void Sort<T>(T[] array,

Comparison<T> comparison)

T – типизирующий параметр;

Comparison<T> - обобщенный делегат-тип:

public delegate int Comparison<**in** T>(T x, T y)

Пример использования метода Array.Sort( )

```
int[ ] temp = new int[ ] { 10, 11, 15, 22, 43, 5, 8, 13 };
Array.Sort(temp, (x, y) =>   // тип не указан
      {  if (x%5 == 0 & y%5 != 0 ) return 1
         if (x == y) return 0;
         return -1;  // верный порядок
      } );
Array.ForEach(temp, y => Console.Write(y + "\t"));
```

Результат выполнения:

13 8 43 22 11 5 15 10

Метод Array.Find( )

public static T Find<T>(T[ ] array, Predicate<T> match)

T – типизирующий параметр;

Predicate <T> - обобщенный делегат-тип:

public delegate bool Predicate<in T>(T obj)

Пример использования метода Array.Find( )

```
int[ ] temp = new int[ ] { 10, 11, 15, 22, 43, 5, 8, 13 };
int res = Array.Find(temp, t => t%5==0 & t%3==0);
Console.WriteLine("res = "+res);
```

Результат выполнения:

res = 15

Метод Array.FindAll( )

```
public static T [ ] FindAll<T>( T[ ] array,
                    Predicate<T> match)
```

T – типизирующий параметр;

Predicate <T> - обобщенный делегат-тип:

```
public delegate bool Predicate<in T>(T obj)
```

Пример использования метода Array.FindAll( )

```
int[ ] temp = new int[] { 10, 11, 15, 22, 43, 5, 8, 13 };
int[ ] row = Array.FindAll(temp, t => t % 5 == 0);
Array.ForEach(row, y => Console.Write(y + "\t"));
```

Результат выполнения:

```
10      15      5
```