Лабораторная работа №6

Делегаты

Цель работы: изучить работу с делегатами в языке C#.

Теоретическая часть

Делегаты

Делегат — это тип, который представляет ссылки на методы с определенным списком параметров и типом возвращаемого значения.

При создании экземпляра делегата этот экземпляр можно связать с любым методом с совместимой сигнатурой и типом возвращаемого значения. Метод можно вызвать (активировать) с помощью экземпляра делегата.

Делегаты используются для передачи методов в качестве аргументов к другим методам. Делегату можно назначить любой метод из любого доступного класса или структуры, соответствующей типу делегата.

Объявление

```
modifier delegate ReturnType DelegateName ([Parameter_1]);
```

Инициализация

```
DelegateName DelegateObjectName = new DelegateName(MethodName);
```

Вызов

```
DelegateObjectName([Parameter_1]);
```

После того, как был создан экземпляр делегата, можно выполнить вызов этого делегата как обычного метода. Этот вызов будет передаваться (делегироваться) методу или нескольким методам, которые скрыты за этим делегатом.

Параметры, передаваемые делегату вызывающим кодом, передаются в метод(-ы), а возвращаемое методом значение (при его наличии) возвращается делегатом обратно в вызывающий код.

Рассмотрим пример:

```
public delegate void MyDelegate(string customString);

public static void PrintMessage(string message)
{
 Console.WriteLine($"From {nameof(PrintMessage)}: {message}");
}


// Инициализируем делегат
MyDelegate delegateInstance = PrintMessage;

// Вызываем делегат
delegateInstance("Hello World!");
```

Анонимные функции

Анонимная функция — это "встроенный" оператор или выражение, которое может использоваться, когда тип делегата неизвестен. Ее можно использовать для инициализации именованного делегата или передать вместо типа именованного делегата в качестве параметра метода.

Для создания анонимной функции можно использовать лямбда-выражение или анонимный метод.

Пример использования лямбда-выражений

```
delegate int Operation(int x, int y);

Operation sum = (x, y) => x + y;

int result = sum(5, 10);

//C# v.10

var mul = (int x, int y) => x * y;

Operation comp = (x, y) =>

{

if (x > y) return 1;

else if (x == y) return 0;

else return -1;

};
```

```
delegate int UnaryOperation(int x);

UnaryOperation square = x => x * x;

delegate int MethodWithoutArguments();

MethodWithoutArguments getYear = () => DateTime.Now.Year;
```

Для создания анонимного метода используется оператор delegate.

```
Operation sum = delegate (int x, int y) { return x + y; };

UnaryOperation square = delegate (int x) { return x * x; };

MethodWithoutArguments getYear = delegate () { return
DateTime.Now.Year; };
```

Мультикаст-делегаты

Делегаты, включающие в себя более одного метода, называются мультикаст-делегатами. При вызове они выполняют каждый метод в заданном порядке, позволяя таким образом связывать несколько методов в цепочку.

Для работы мультикаст-делегатов те не должны возвращать какой-либо результат. В противном случае обработается результат последнего метода цепочки.

Практическая часть

1) Расширить класс `DynamicArray` методом фильтрации `Filter` элементов коллекции, который принимает входным параметром делегат типа `Func<T, bool>`, где `T` - тип элементов коллекции. Выходное значение вызываемого делегата определяет, остается ли элемент в коллекции (`true`), или должен быть отфильтрован (`false`). Вызывая этот делегат для каждого элемента, в коллекции должны остаться только элементы, подходящие заданному в делегате условию.

Пример вызова такого метода с использованием лямбда-выражения:

```
dynamicArray.Filter((o) => o.ToString() == “1”);
```

В результате вызова этого метода в коллекции должны остаться только элементы, строковое представление которых равно значению “1”.

2) Расширить класс `DynamicArray` методом `Sort` сортировки элементов коллекции, который принимает входным параметром делегат-”компаратор” типа `Func<T, T, int>`, где `T` - тип элементов коллекции. Выходное значение вызываемого делегата может быть в трёх состояниях:

- Меньше 0 - первый элемент меньше, чем второй элемент
- Равно 0 - элементы равны
- Больше 0 - первый элемент больше, чем второй элемент

Содержание отчета

1. Титульный лист
2. Цель работы
3. Задание
4. Код программы
5. Результат выполнения