Занятие 7, ч1. Делегаты, события, анонимные методы

Тренер: Алексей Дышлевой

Основные вопросы

- Делегати. Створення та використання делегатів. Одиночні делегати. Непов’язані делегати. Ланцюжки делегатів.
- Події. Події в графічному середовищі
- Анонімні методи. Захоплені змінні. Анонімні методи як прив’язки параметрів делегатів.
- Розширюючі методи. Трансформації. Ланцюжки операцій. Користувацькі ітератори

Делегаты

- При разработке приложений с использованием Visual C# в большинстве случаев методы вызываются явно по имени. Функциональность и ее физическая реализация тесно связаны в методе. Однако, существуют сценарии, когда такой подход не годится.
- Одним из способов реализации такой функциональности является использовании операторов if или switch. Однако такой подход является статичным и зависит от различных методов, доступных при написании кода. Более расширяемый подход заключается в использовании делегатов.

Делегаты

- Сторонние разработчики, предоставляющие сборки с методами, которые можно вызвать асинхронно, часто позволяют указать метод в коде пользователя, когда их метод завершен. При таком сценарии заранее неизвестны имена всех возможных в этом случае методов, поэтому эффективным решением является использование делегата, который можно связать с одним или более методов.

- Делегат - ссылка на метод. Во время выполнения делегат можно связать с любым методом, соответствующим его сигнатуре, и тогда для реализации функциональности метода, вызывается делегат.

Делегаты

- Часто используемые для отделения функциональности от метода являются групповые (multicast) операции. Если операция является групповой, несколько методов могут реализовывать одну и ту же операцию. Можно добавить ссылки на все реализации методов делегату, а при вызове делегата, среда выполнения вызовет каждый метод по очереди.

Делегаты

- Делегат – это пользовательский тип, который инкапсулирует метод. Объект типа делегат – это безопасный в отношении типов объект, указывающий на другой метод или, возможно, список методов приложения, которые могут быть вызваны позднее. Объект делегата поддерживает три важных фрагмента информации:
- адрес метода, на котором он вызывается;
- аргументы метода (если они есть);
- возвращаемое значение метода (если оно есть)

Делегаты

- Делегат определяется с помощью ключевого слова delegate. Как и для любого типа для делегата можно указать модификатор доступа.
- При определении делегата создается новый тип, такой же как class или struct.
- public delegate void SomeDelegate(int value);

Делегаты

- При обработке данного типа делегат компилятор C# автоматически генерирует класс
- public class SomeDelegate: MulticastDelegate
- {
- public SomeDelegate(Object object, IntPtr method);
- public virtual void Invoke(Int32 value);
- public virtual IAsyncResult BeginInvoke(Int32 value, AsyncCallback callback, Object object);
- public virtual void EndInvoke(IAsyncResult result);
- }
- Наследовать напрямую от класса MulticastDelegate нельзя, для этого используется ключевое слово delegate.

Методы делегатов

- Метод Invoke используется для синхронного вызова каждого из методов, поддерживаемых объектом делегата; это означает, что вызывающий код должен ожидать завершения вызова, прежде чем продолжить свою работу. Синхронный метод Invoke обычно явно в коде C# не вызываться, он вызывается «за кулисами» при применении соответствующего синтаксиса C#.
- Методы BeginInvoke и EndInvoke предлагают возможность вызова текущего метода асинхронным образом, в отдельном потоке выполнения. Хотя в библиотеках базовых классов .NET предусмотрено целое пространство имен, посвященное многопоточному программированию (System.Threading), делегаты предлагают эту функциональность в готовом виде.

Инициализация делегата

- Переменные делегата инициализируются конкретными методами при использовании конструктора делегата с одним параметром – именем метода (или именем другого делегата). Если делегат инициализируется статическим методом, требуется указать имя класса и имя метода, для инициализации экземплярным методом указывается объект и имя метода. При этом метод должен обладать подходящей сигнатурой:
- sdStatic = new SomeDelegate(ClassName.SomeStaticFunction);
- sdInstance = new SomeDelegate(obj.SomeInstanceMethod);
- Для инициализации делегата можно использовать упрощенный синтаксис – достаточно указать имя метода без применения new.
- sdStatic = ClassName.SomeStaticFunction;
- sdInstance = obj.SomeInstanceMethod;

Задача

- Создать делегат, инициализровать его и вызвать

Класс Delegate и MulticastDelegate

- Класс System.MulticastDelegate (в сочетании с его базовым классом System.Delegate) обеспечивает своих наследников доступом к списку, содержащему адреса методов, поддерживаемых типом делегата, а также несколькими дополнительными методами (и перегруженными операциями), чтобы взаимодействовать со списком вызовов

Метод	Описание
Combine	Статический метод добавляет метод в список, поддерживаемый делегатом. В C# этот метод вызывается за счет использования перегруженной операции += в качестве сокращенной нотации
GetInvokationListO	Метод возвращает массив типов System.Delegate, каждый из которых представляет определенный метод, доступный для вызова
Remove RemoveAll	Статические методы удаляют метод (или все методы) из списка вызовов делегата. В C# метод Remove может быть вызван неявно, посредством перегруженной операции -=

Свойства Delegate

- классе Delegate определены два неизменяемых открытых экземплярных свойства: Target и Method. При наличии ссылки на объект делегата можно запросить значения этих свойств. Target возвращает ссылку на объект, обрабатываемый при обратном вызове метода. В сущности, Target возвращает значение, хранимое в закрытом поле _target. Если этот метод – статический, Target возвращает null. Свойство Method возвращает объект System.Reflection.MethodInfo, описывающий метод обратного вызова. В сущности, у свойства Method есть внутренний механизм, который преобразует значение из закрытого поля _methodPtr в объект MethodInfo и возвращает его.

Вызов делегата

- После определения делегата и создания его экземпляра делегат можно вызвать в коде. Делегат можно вызвать аналогично методу, используя имя экземпляра делегата со следующими за ним в скобках параметрами. Существует одно важное различие между вызовом делегата и вызовом метода: делегат может не ссылаться на методы и поэтому может быть равен null. Следует всегда проверять делегат на null, прежде чем ссылаться на него.

Пример

- public isValidDelegate isValid = null;
- . . .
- if(isValid != null)
- {
- isValid();
- }
- Обнаружив строку вида
- isValid();
- компилятор генерирует код аналогичный, как если бы он встретил в исходном тексте следующее:
- isValid.Invoke();
- При вызове метода Invoke используются поля _target и _methodPtr для вызова желаемого метода на заданном объекте, при этом, сигнатура метода Invoke соответствует сигнатуре делегата.

Задача

- Создать 3 класса, не связанных между собой. В каждом классе создать по 3 метода: 1 параметр string, 2 параметра (string, int), без параметров. В 3-ем классе все методы должны быть статическими.
- Создать делегат, и записать в него методы. Делегатов может быть несколько. Вызвать каждый метод с помощью делегата.

Групповые делегаты

- Цепочка делегатов (chaining) – это подмножество или набор объектов-делегатов, которое позволяет вызывать все методы, представленные делегатами набора.

- DelegateIntro di;
- . . .
- Feedback fb1 = new Feedback(FeedbackToConsole);
- Feedback fb2 = new Feedback(FeedbackToMsgBox);
- Feedback fb3 = new Feedback(di.FeedbackToFile);

Групповые делегаты

- Переменная-ссылка на объект-делегат Feedback должна ссылаться на цепочку или набор делегатов, служащих оболочками методам обратного вызова. Инициализация fbChain значением null говорит об отсутствии методов обратного вызова. Открытый статический метод Combine класса Delegate используется для добавления делегата в цепочку:
- Feedback fbChain = null;
- fbChain = (Feedback)Delegate.Combine(fbChain, fb1);
- При выполнении последней строки кода метод Combine «видит» попытку объединить null и fb1. Код метода Combine просто возвращает значение fb1, а в переменной fbChain размещается ссылка на тот же объект-делегат, на который ссылается fb1.

Групповые делегаты

- Чтобы добавить в цепочку еще один делегат, снова вызывается метод Combine.
- fbChain = (Feedback)Delegate.Combine(fbChain, fb2);
- Код метода Combine «видит», что fbChain уже ссылается на объект-делегат, и поэтому создает новый объект-делегат. Новый объект-делегат инициализирует свои закрытые поля _target и _methodPtr. Присваиваемые полям значения для данного обсуждения не важны, но важно то, что поле _invocationList инициализируется ссылкой на массив объектов-делегатов. Первый элемент массива (с индексом 0) инициализируется ссылкой на делегат, служащий оберткой метода FeedbackToConsole (это делегат, на который сейчас ссылается fbChain). Второй элемент массива (индекс 1) инициализируется ссылкой на делегат, служащий оберткой метода FeedbackToMsgBox (на этот делегат ссылается fb2). Наконец, переменной fbChain присваивается ссылка на вновь созданный объект-делегат.

Групповые делегаты

- Для создания третьего делегата снова вызывается метод Combine.
- fbChain = (Feedback)Delegate.Combine(fbChain, fb3);
- И снова, видя, что fbChain уже ссылается на объект-делегат, Combine создает новый объект-делегат. Как и раньше, новый объект-делегат инициализирует свои закрытые поля _target и _methodPtr какими-то значениями, а поле _invocationList инициализируется ссылкой на массив объектов-делегатов. Первый и второй элементы массива (индексы 0 и 1) инициализируются ссылками на те же делегаты, на которые ссылался предыдущий объект-делегат в массиве. Третий элемент массива (индекс 2) инициализируется ссылкой на делегат, служащий оберткой метода FeedbackToFile (на этот делегат ссылается fb3). В заключение, переменной fbChain присваивается ссылка на вновь созданный объект-делегат. Ранее созданный делегат и массив, на который ссылается его же поле _invocationList, теперь подлежат обработке механизмом сборки мусора.

Групповые делегаты

- После выполнения всего кода создания цепочки, переменная fbChain передается методу Counter.
- Counter(1, 2, fbChain);
- . . .
- private static void Counter(int from, int to, Feedback fb)
- {
- for (int val = from; val <= to; val++)
- {
- // If any callbacks are specified, call them
- if (fb != null)
- fb(val);
- }
- }
- . . .
- Внутри Counter содержится код, неявно вызывающий метод Invoke по отношению к объекту-делегату Feedback. Когда метод Invoke вызывается по отношению к делегату, на который ссылается fbCbain, делегат обнаруживает, что поле _invocationList не равно null, и инициируется выполнение цикла, итерационно обрабатывающего все элементы массива путем вызова метода, оболочкой которого служит указанный делегат. Методы вызываются в следующей последовательности: FeedbackToConsole, FeedbackToMsgBox и FeedbackToFile

Удаление делегата из цепочки

- Делегаты можно удалять из цепочки, вызывая статический метод Remove объекта Delegate. Например:
- fbChain = (Feedback)Delegate.Remove(fbChain, new Feedback(FeedbackToMsgBox));
- Метод Remove просматривает массив делегатов (с конца и до члена с индексом 0), поддерживаемых объектом-делегатом, на который ссылается первый параметр (в примере fbChain). Remove ищет делегат, поля _target и _methodPtr которого совпадают с соответствующими полями второго параметра (в примере нового делегата Feedback). Если Remove находит совпадение и в массиве остается более одного элемента, создается новый объект-делегат – создается массив _involocationList и инициализируется ссылкой на все элементы исходного массива за исключением удаляемого элемента – и возвращается ссылка на новый объект-делегат. При удалении последнего элемента в цепочке Remove возвращает null. Следует отметить, что за один раз Remove удаляет из цепочки лишь один делегат, а не все делегаты с заданными значениями в полях _target и _methodPtr.

Упрощенный подход

- DelegateIntro di;
- . . .
- Feedback fb1 = new Feedback(di.FeedbackToConsole);
- Feedback fb2 = new Feedback(di.FeedbackToMsgBox);
- Feedback fb3 = new Feedback(di.FeedbackToFile);
- . . .
- Feedback fbChain = null;
- fbChain += fb1;
- fbChain += fb2;
- fbChain += fb3;
- . . .
- fbChain -= new Feedback(FeedbackToMsgBox);

Задача

- Продемонстрировать использование групповых делегатов на примере предыдущей задачи (вызвать цепочку методов с помощью одного делегата)

Итерация по цепочкам делегатов

- Если требуется вызвать конкретный делегат из цепочки, или вызвать все делегаты последовательно, то для этого класс Delegate представляет метод GetInvocationList(), который возвращает массив делегатов из цепочки.
- Feedback fbChain = null;
- fbChain = fb1 + fb2 + fb3;
- Delegate[] delegateList = fbChain.GetInvocationList();
- for (int i = delegateList.Length - 1; i >= 0; i--) {
- ((Feedback)delegateList[i])();
- }

Задача

- Вызвать методы с помощью итерации по цепочке делегатов

Делегаты открытого экземпляра

- Делегаты открытого экземпляра (несвязанные делегаты) используются, когда необходимо, чтобы делегат представлял метод определенного экземпляра, но вызывать этот метод можно на целой коллекции экземпляров.
- Класс MethodInfo выявляет атрибуты метода и обеспечивает доступ к его метаданным

Пример

- DelegateIntro di1;
- DelegateIntro di2;
- DelegateIntro di3;

- List< DelegateIntro > diChain = new List< DelegateIntro >();
- diChain.Add(fb1);
- diChain.Add(fb2);
- diChain.Add(fb3);
- MethodInfo mi =
- typeof(DelegateIntro).GetMethod("Print",
- BindingFlags.Public | BindingFlags.Instance);
- Feedback fbChain =
- (Feedback)Delegate.CreateDelegate(typeof(Feedback), mi);
- foreach (var c in cl) {
- fbChain(c);
- }

Анонимные методы

- Для использования делегата, он должен ссылаться на метод. Однако, часто бывает необходимо только отослать делегат к блоку кода, при этом создание метода, реализующего логику этого блока кода, добавляет накладные расходы. Для решения таких ситуаций Visual C# позволяет определить анонимные методы.
- Анонимным является методом, не имеющий имени, а имеющий только тип, список параметров и тело метода. Анонимный метод можно использовать с делегатом, вызывая его через делегат.

Пример

```
delegate int myDelegate(int number);
// Create an instance of the delegate type.
myDelegate myDelegateInstance = null;
…
    // Add an anonymous method to the delegate.
    // Do not specify any parameters.
    myDelegateInstance += new myDelegate(delegate
    {
        // Perform operation.
        return 5;
    });
    // Add an anonymous method to the delegate.
    // Specify parameters; the parameters must match
    // the signature of the delegate.
    myDelegateInstance += new myDelegate(delegate(int parameter)
    {
        return 10;
    });
    // Invoke the delegate.
    int returnedValue = myDelegateInstance(2);
    // returnedValue = 10 (as second method is called last).
```

Задача

- Добавить анонимный метод в цепочку предыдущей задачи

Анонимные методы

- Анонимные методы могут обращаться к локальным переменным метода, в котором они определены. Формально такие переменные называются внешними (outer) переменными анонимного метода.

Особенности анонимных методов

- Анонимный метод не имеет доступа к ref и out параметрам определяющего их метода.
- Анонимный метод не может иметь локальных переменных, имена которых совпадают с именами локальных переменных объемлющего метода.
- Анонимный метод может обращаться к переменным экземпляра (или статическим переменным) из контекста объемлющего класса.
- Анонимный метод может объявлять локальные переменные с теми же именами, что и у членов объемлющего класса (локальные переменные имеют отдельный контекст и скрывают внешние переменные-члены).

Захваченные переменные

- При объявлении анонимных методов любые переменные, объявленные вне контекста анонимного метода, но доступные внутри его, включая ссылку this, рассматриваются как внешние переменные. И если тело анонимного метода ссылается на эти переменные, считают, что анонимный метод «захватил» переменную.
- Возможность обращения из тел анонимных методов к переменным, принадлежащим ко контексту, в котором эти методы были определены, очень удобна. Это называется замыкание.

Анонимные методы как привязки параметров делегатов (карирование)

- Привязка параметров – это техника, при которой необходимо вызвать делегат, имеющий обычно более одного параметра, таким образом, чтобы один или более параметров были зафиксированы, а другие в разных вызовах вариировались

Расширяющие методы

- Расширяющие методы позволяют «добавлять» методы в существующие типы без создания нового производного типа, перекомпиляции или иного изменения исходного кода.
- public static class MyExtensions
- {
- public static int WordCount(this String str)
- {
- return str.Split(new char[] { ' ', '.', '?' },
- StringSplitOptions.RemoveEmptyEntries).Length;
- }
- }

Пример

```
public static class ExtensionMethods
{
public static double Angle(this Complex compl)
{
if ((compl.Re == 0) && (compl.Im >= 0)) {
return Math.PI / 2;
}
if ((compl.Re == 0) && (compl.Im < 0)) {
return 3 * Math.PI / 2;
}
return Math.Atan(compl.Im / compl.Re);
}
}
class Program
{
static void Main(string[] args)
{
Complex c1 = new Complex(1,2);
Console.WriteLine(c1);
double angle = c1.Angle();
Console.WriteLine(angle);
Console.ReadKey();
}
}
```

Задача

- Создать расширенный метод для одного из 3-х классов, созданных в начале занятия

Домашнее задание

- На основании методов из класса калькулятор (созданного во время изучения модульного тестирования) создать делегат. Добавить в делегат несколько методов, и вызвать.
- Добавить в делегат и вызвать анонимный метод, который возвращает дробную часть при делении.
- Создать расширенные методы для поиска e-mail адресов в текстовых файлах (для класса из домашнего задания 5-2)