Занятие 7, ч2. 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

Еще немного о делегатах – ковариантность и контрвариантность

- Ковариантность (свободные делегаты) позволяет построить единственный делегат, который может указывать на методы, возвращающие связанные классическим наследованием типы классов
- Контрвариантность позволяет создать единственный делегат, который может указывать на многочисленные методы, принимающие объекты, связанные классическим наследованием

События

- В Visual C# события очень похожи на делегаты. События действительно основаны на делегатах, хотя семантически, отличаются от них в назначении. Тип использует события для свидетельствования о значительном изменении в приложении и организации делегата для вызова обработчика события. В то время как любой объект, имеющий доступ к делегату, может ссылаться на такой делегат, только тип, определяющий событие, может вызвать это событие.
- Когда тип определяет событие, другие типы могут определить метод, соответствующий сигнатуре делегата, связанного с событием. Эти типы могут подписаться на событие и указать, что их метод должен быть запущен при возникновении события.
- События следует использовать только тогда, когда разрабатывается тип, которому необходимо сообщать многим классам об изменении своего состояния.

События

- События основаны на делегатах, поэтому прежде чем определять событие, следует определить делегат, описывающий метод обработки события.
- Согласно стандартному соглашению делегат события не должен возвращать значение и должен принимать два параметра. Первому параметру необходимо иметь тип object и имя sender. Второй параметр должен быть производным от класса EventArgs пространства имен System с именем e. Если событие не требует передачи аргументов события при его вызове, можно использовать класс EventArgs напрямую
- public delegate void MyEventDelegate(object sender, EventArgs e);

События

- Рекомендуется инициализировать событие значением null, поскольку это позволяет типу определить подписался ли какой-нибудь тип на событие до его возникновения (когда метод подписывается на событие, событие больше не будет null)
- При определении события, оно, как правило, имеет уровень доступа public для того, чтобы использующие тип классы могли иметь доступ к нему

Использование события

- После того как событие определено, его можно использовать в приложении. Для этого на событие нужно подписаться в использующих типах и приложениях и генерировать событие в типе.
- Чтобы подписаться на событие используется составная операция присваивания «+=» точно так же, как при добавлении ссылки на метод делегату.
- MyEvent += new MyEventDelegate (myHandlingMethod);
- В дополнение к подписке на событие можно отказаться от подписки на событие. Для этого используется составная операция присваивания «-=».
- MyEvent -= myHandlingMethod;

Генерация события

- Для генерации события используется имя события и указываются параметры в скобках, аналогично вызову метода. При вызове события необходимо сначала проверить, что оно не является нулевым. Если у события нет подписчиков, не следует пытаться его вызвать. При попытке вызвать событие, не имеющее подписчиков, код будет генерировать исключение NullReferenceException.
- if (MyEvent != null)
- {
- EventArgs args = new EventArgs();
- MyEvent(this, args);
- }
- Метод инициализации события следует стараться именовать с префиксом On.

Пример – простейшее событие

- delegate void MyEventHandler();
- // Объявляем класс события,
- class MyEvent {
- public event MyEventHandler SomeEvent;
- // Этот метод вызывается для генерирования события,
- public void OnSomeEvent() {
- if(SomeEvent != null)
- SomeEvent();
- class EventDemo {
- // Обработчик события,
- static void handler() {Console.WriteLine("Произошло событие.");
- public static void Main() {
- MyEvent evt = new MyEvent();
- // Добавляем метод handler() в список события,
- evt.SomeEvent += new MyEventHandler(handler);
- // Генерируем событие,
- evt.OnSomeEvent();
- }
- }

Пример – класс аргументов события

- public class OnPowerOnArgs : EventArgs
- {
- public string DisplayText
- {
- get;
- private set;
- }
- public OnPowerOnArgs(string strText)
- {
- DisplayText = strText;
- }
- }

Пример – класс издатель события

- public class UserInterface
- {
- public event EventHandler<OnPowerOnArgs> OnPowerOn;
- public UserInterface()
- {
- OnPowerOn += HandlePowerOn;
- }
- public void InitiatePowerOn(object sender, OnPowerOnArgs args)
- {
- OnPowerOn.Invoke(sender, args);
- }
- protected void HandlePowerOn(object sender, OnPowerOnArgs args)
- {
- Console.WriteLine(sender.ToString()+": "+args.DisplayText);
- }
- }

Пример – класс-подписчик

- public class User
- {
- static void Main(string[] args)
- {
- UserInterface ui = new UserInterface();
- ui.InitiatePowerOn
- ("Vasiliy Pupkin", new OnPowerOnArgs("Power On"));
- Console.ReadKey();
- }
- }

Задача

- Создать класс, который моделирует движение автомобиля: начало движения, остановка, движение (используется топливо), заправка.
- Создать событие «все топливо использовано», в результате которого автомобиль остановиться.

Управление удалением и добавлением метода

- event событийный_делегат имя__события {
- add {
- // Код добавления события в цепочку событий.
- remove {
- // Код удаления события из цепочки событий.
- }
- }

Как работают события

- Событие в C# развертывается в два скрытых метода, один из которых имеет префикс _add, а другой - _remove. За этим префиксом следует имя события.
- В методе _add осуществляется вызов метода Delegate.Combine()
- _remove вызывает Delegate.Remove()

Рекомендации по использованию событий

- При определении делегата для события следует использовать стандартную сигнатуру события
- Для вызова события следует использовать защищенный виртуальный метод
- Не следует отправлять нулевые значения событиям

Упрощенная форма регистрации событий в Visual Studio

- После ввода += Visual Studio предлагает нажать Tab для автоматического заполнения экземпляра делегата. После чего предлагается имя обработчика события по умолчанию (соответственно его можно редактировать). После повторного нажатия Tab создается заготовка метода, который будет вызываться при генерации события.

Использование событий в графических приложениях

- Графические приложения используют большое количество событий и обычно включают много взаимодействий с пользователем. Например, при нажатии кнопки пользователь ожидает незамедлительной реакции приложения. При разработке графических приложений используются события, немедленно реагирующие на взаимодействие пользователя с интерфейсом. Например, кнопка предоставляет событие Click, и, подписавшись на событие Click, можно добавить обработчик события, в котором поместить код для обработки события нажатия кнопки, выполняющий соответствующую логику.

Потоки для событий

- При написании простого приложения, как правило, используется один поток, в котором последовательно выполняется весь код. Если процесс занимает длительное время, приложение будет приостановлено, ожидая завершения процесса. Часто разрабатываются приложения «зависание» которых в ожидании завершения длительно-работающего процесса не желательно. В .NET Framework эту проблему можно решить путем использования нескольких потоков. Тогда каждый поток может работать одновременно с другими потоками, что означает, что приложению больше не нуждается в ожидании завершения длительного процесса. Длительный процесс может быть выполнен в другом потоке, в то время как приложение может продолжать работу с другими задачами.

Свойства для событий

- Издатель определяет момент вызова события, подписчики определяют предпринятое ответное действие
- У события может быть несколько подписчиков. Подписчик может обрабатывать несколько событий от нескольких издателей
- События, не имеющие подписчиков, никогда не возникают
- События используются для оповещения о действиях пользователя, таких как нажатие кнопок или выбор меню и их пунктов в графическом пользовательском интерфейсе
- Если событие имеет несколько подписчиков, то при его возникновении происходит синхронный вызов обработчиков событий
- В библиотеке классов .Net Framework в основе событий лежит делегат EventHandler и базовый класс EventArgs

Пример

- Изменить пример с UserInterface

Слабые события

- С помощью событий издатель и слушатель (подписчик) соединяются напрямую. По этой причине может возникнуть проблема при сборке мусора. Например, если на слушателя больше никто не ссылается, все-таки остается ссылка от издателя. Сборщик мусора не может очистить память от слушателя, поскольку издатель удерживает ссылку и генерирует события для слушателя.
- Это жесткое соединение может быть разрешено с использованием шаблона слабого события (weak event pattern) и применения WeakEventManager в качестве посредника между издателями и слушателями
- Определен в System.Windows сборки WindowsBase

Диспетчер слабых событий

- Создается класс, унаследованный от WeakEventManager (System.Windows сборки WindowsBase)
- Шаблон слабого события подразумевает наличие в классе диспетчера слабых событий методов AddListener(), Removelistener(). С помощью этих методов слушатель подключается и одключается от событий издателя вместо использования событий издателя непосредственно.
- Слушатель также нуждается в реализации интерфейса IWeakEventListener

- (более подробно – Нейгел, Ивьен…)

Задача

- Изменить задание с автомобилем

lambda

- Лямбда-выражением является выражение, возвращающее метод.
- Полезны для определения анонимных, но строго типизированных методов. Широко используются в Visual C# особенно при определении Language-Integrated Query (LINQ) выражений.
- Лямбда-выражение состоит из набора параметров и тела. Тело определяет функцию, которая может вернуть значение , в этом случае Visual C# компилятор выводит возвращаемый тип из определения метода.
- Лямбда-выражение определяется с помощью операции =>.
- x => x * x
- Прочитать это лямбда-выражение можно так «Для данного x, вычислить x * x». Тип x не определен, он будет выведен компилятором при использовании лямбда-выражения. Кроме того будет выведен и тип возвращаемого значения.

lambda

- На лямбда-выражение можно ссылаться из делегата.
- delegate int MyDelegate(int a);
- ...
- MyDelegate myDelegateInstance = null;
- myDelegateInstance += x => x * x;
- При вызове делегата необходимо указать значения для всех параметров, которые принимает лямбда-выражение, и тело лямбда-выражения выполнится. Из тела лямбда-выражения можно вернуть любое возвращаемое значение так же, как при вызове обычного метода. В следующем примере показан вызов делегата на основе простого лямбда-выражения и делегата, описанного выше.
- Console.WriteLine(myDelegateInstance (10)); // Displays the value 100

Определение lambda

- Лямбда-выражения были первоначально частью математической нотации, называемой лямбда-вычисления (Lambda Calculus), которая обеспечивала обозначения для описания функций.
- Тело лямбда-выражения может быть простым выражением, или это может быть блок кода Visual C#, который определяет тело метода, заключенное в фигурные скобки. При определении тела метода, можно использовать любые программные конструкции Visual C#.
- Лямбда-выражение может принимать более одного параметра, в этом случае указывается список параметров, заключенных в круглые скобки.
- delegate int AddDelegate(int a, int b)
- ...
- AddDelegate myAddDelegate = null;
- myAddDelegate += (x, y) => x + y;
- Лямбда-выражения могут также принимать нулевые параметры. В этом случае указывается пустой список параметров

Пример

- x => x * x
- x => { return x * x ; }
- (int x) => x / 2
- () => myObject.MyMethod(0)
- (x, y) => { x++; return x / y; }
- (ref int x, int y) { x++; return x / y; } // лучше не использовать

Задача

- Создать делегат с параметрами string, int, возвращает string.
- Добавить лямбда-выражение для вызова с помощью делегата. Выполняет добавление числа к тексту.
- Вызвать делегат.

Область действия переменной в lambda

- При определении лямбда-выражения в его теле можно определить переменные. Областью видимости этих переменных является лямбда-выражение. Когда лямбда-выражение завершается, переменная выходит из области видимости.
- В лямбда-выражении можно получить доступ к переменным, определенным вне этого выражения. Однако время жизни этих переменных будет расширено до тех пор, пока лямбда-выражение само не выйдет из области видимости, что может иметь влияние на сбор мусора

Замыкание

- Анонимные методы и лямбда-операторы способны захватывать внешний контекст вычисления. Если при описании тела анонимного метода применялась внешняя переменная, вызов метода будет использовать текущее значение переменной. Захват внешнего контекста иначе называют замыканием (closure).

Домашнее задание

- Создать событие превышение допустимого размера данных при записи в файл. Если размер данных больше допустимого, реализовать вывод пользователю сообщения с возможностью выбора завершения операции: записать только те данные, которые помещаются в файл или не записывать нечего.
- Оформить подписку на событие.
- Создать методы записи и считывания данных из файла. И записать данные разных типов (в том числе, объектов собственного класса).
- Функционал обработки события реализовать в виде классов аргументов события (с наследованием от EventArgs), издателя события и подписчика.