Лабораторная работа № 13
Разработка программы, управляемой событиями

Цель. Получить практические навыки разработки программы, управляемой событиями, использования делегатов и событий..

Теоретические сведения.

1. Делегаты

Делегат (delegate) — это тип, который позволяет хранить ссылки на функции. Объявляются делегаты практически также, как и функции, но только безо всякого тела функции и с ключевым словом **delegate.**

В объявлении любого делегата указывается возвращаемый тип и список параметров.

```
[спецификаторы ] delegate тип имя_делегата ( [ параметры ] )
```

Спецификаторы делегата имеют тот же смысл, что и для класса, причем допускаются только спецификаторы `new, public, protected, internal` и `private.`

`Тип` – тип функции, `параметры` – параметры функции.

```
public delegate void D ( int i );
```

После определения делегата можно объявлять переменную с типом этого делегата. Далее эту переменную можно инициализировать как ссылку на любую функцию, которая имеет точно такой же возвращаемый тип и список параметров, как и у делегата. После этого функцию можно вызывать с использованием переменной делегата так, будто бы это и есть сама функция.

Многоадресатная передача — это способность создавать список вызовов (или цепочку вызовов) методов, которые должны автоматически вызываться при вызове делегата. Для этого достаточно создать экземпляр делегата, а затем для добавления методов в эту цепочку использовать оператор "+=". Для удаления метода из цепочки используется оператор " - = " (можно + и – соответственно). Делегат с многоадресатной передачей имеет одно ограничение: он должен возвращать тип void.

2. События

События похожи на исключения тем, что они тоже генерируются, т.е. выдаются объектами, и тем, что для них тоже можно предоставлять реагирующий на них выполнением какого-нибудь действия код. Однако существует и несколько отличий, наиболее важное из которых состоит в отсутствии для обработки событий структуры, эквивалентной try. . . catch. Вместо применения этой структуры на события нужно подписываться (subscribe).

Под подпиской на событие подразумевается предоставление кода, который должен выполняться при генерации данного события, в виде обработчика событий (event handler).

На событие можно подписывать несколько обработчиков, которые тогда все будут вызываться при генерации этого события. Эти обработчики могут являться как частью того класса объекта, который генерирует данное событие, так и частью других классов.

Сами обработчики событий представляют собой просто функции. Единственным ограничением для такой функции является то, что ее возвращаемый тип и параметры должны обязательно соответствовать тем, которых требует событие. Это ограничение входит в состав определения события и задается **делегатом**.

Базовая последовательность обработки выглядит следующим образом:

1. Приложение создает объект, который может генерировать событие.
2. Приложение подписывается на событие.
3. При генерации события подписчику отправляется соответствующее уведомление.

Перед определением события требуется обязательно определить используемый вместе с событием тип делегата, т.е. тип делегата, типу и параметрам которого должен соответствовать метод обработки событий. Для выполнения этого используется стандартный синтаксис делегатов, с помощью которого необходимый делегат определяется как общедоступный.

Для обработки события на него нужно подписываться, предоставляя функцию — обработчик событий, возвращаемый тип и параметры которой должны совпадать с возвращаемым типом и параметрами делегата, закрепленного для применения с этим событием.

3. Пример программы, обрабатывающей события

В C# каждое событие определяется делегатом, описывающим сигнатуру сообщения. Объявление события - это двухэтапный процесс:

- Объявляется делегат - функциональный класс, задающий сигнатуру.
- В классе, создающем события, объявляется событие как экземпляр соответствующего делегата**.**

```
delegate void CollectionHandler(object source, CollectionHandlerEventArgs
args);//делегат
    class MyNewCollection:MyCollection
    {
//происходит при добавлении нового элемента или при удалении элемента из
//коллекции
 public event CollectionHandler CollectionCountChanged;
//объекту коллекции присваивается новое значение
public event CollectionHandler CollectionReferenceChanged;
```

- Поскольку действия по включению могут повторяться, полезно в состав методов класса добавить защищенную процедуру, включающую событие. Даже если событие генерируется только в одной точке, написание такой процедуры считается признаком хорошего стиля. Этой процедуре обычно дается имя, начинающееся со слова On, после которого следует имя события. Будем называть такую процедуру On-процедурой. Она проста и состоит из вызова объявленного события, включенного в тест, который проверяет перед вызовом, а есть ли хоть один обработчик события, способный принять соответствующее сообщение.

```
//обработчик события CollectionCountChanged
public virtual void OnCollectionCountChanged(object source,
CollectionHandlerEventArgs args)
        {
            if (CollectionCountChanged != null)
               CollectionCountChanged(source, args);
        }
//обработчик события OnCollectionReferenceChanged
public virtual void OnCollectionReferenceChanged(object source,
CollectionHandlerEventArgs args)
       {
            if (CollectionReferenceChanged != null)
                CollectionReferenceChanged(source, args);
        }
```

Объекты, которые принимают сообщение о событии, должны заранее присоединить обработчики событий к объекту EventHandler evnt, задающему событие.

- Последний шаг, который необходимо выполнить в классе создающем события - это в нужных методах класса вызвать процедуру On. Естественно, что перед вызовом нужно определить значения входных аргументов события. После вызова может быть выполнен анализ выходных аргументов, определенных обработчиками события.

```
    public override bool Remove(int position)
```

```
{
 OnCollectionCountChanged(this, new CollectionHandlerEventArgs(this.Name,
"delete", list[position]));
 return base.Remove(position);
 }
     public override int Add(Person p)
 {
 OnCollectionCountChanged(this, new CollectionHandlerEventArgs(this.Name,
"add", p));
return base.Add(p);
 }
     public override Person this[int index]
        {
          get
            {
               return base[index];
            }
          set
            {
OnCollectionReferenceChanged(this, new CollectionHandlerEventArgs(this.Name,
"changed", list[index]));
base[index] = value;
            }
         }
```

Объекты класса-отправителя создают события и уведомляют о них объекты класса (классов)-получателя событий.

Класс-получатель должен иметь обработчик события – процедуру, согласованную по сигнатуре с функциональным типом делегата, который задает событие;

```
  public void CollectionCountChanged(object sourse,
CollectionHandlerEventArgs e)
        {
            JournalEntry je = new JournalEntry(e.NameCollection,
e.ChangeCollection, e.Obj.ToString());
            journal.Add(je);

        }
       public void CollectionReferenceChanged(object sourse,
CollectionHandlerEventArgs e)
        {
            JournalEntry je = new JournalEntry(e.NameCollection,
e.ChangeCollection, e.Obj.ToString());
            journal.Add(je);
        }
```

- Подписка на события заключается в присоединении обработчика события к event-объекту:

```
MyNewCollection mc1 = new MyNewCollection("FIRST");
//один объект Journal подписать на события CollectionCountChanged и
CollectionReferenceChanged из первой коллекции
            Journal joun1 = new Journal();
            mc1.CollectionCountChanged += new
CollectionHandler(joun1.CollectionCountChanged);
            mc1.CollectionReferenceChanged += new
CollectionHandler(joun1.CollectionReferenceChanged);
```

Постановка задачи

Часть 1.

1. Создать иерархию классов (см. лаб. 9). Для каждого класса реализовать конструктор без параметров, с параметрами, свойства для доступа к полям объектов, метод для автоматического формирования объектов. Перегрузить метод ToString() для формирования строки со значениями всех полей класса.
2. Создать класс MyCollection как производный класс от класса Collection<MyClass>. Реализовать в классе методы для заполнения коллекции (элементы коллекции формируются автоматически), добавления элементов коллекции, удаления элементов коллекции, сортировки элементов коллекции по заданному полю, очистки коллекции. Реализовать итератор для доступа к элементам коллекции. Реализовать свойство Length (только для чтения), содержащее текущее количество элементов коллекции.
3. Написать демонстрационную программу иллюстрирующую работу всех методов класса MyCollection. Использовать исключительные ситуации для обработки ошибочных ситуаций.

Часть 2.

1. Определить класс MyNewCollection производный от класса MyCollection, который с помощью событий извещает об изменениях в коллекции.
   - Коллекция состоит из объектов ссылочных типов. Коллекция изменяется при удалении/добавлении элементов или при изменении одной из входящих в коллекцию ссылок, например, когда одной из ссылок присваивается новое значение. В этом случае в соответствующих методах или свойствах класса бросаются события.
   - При изменении данных объектов, ссылки на которые входят в коллекцию, значения самих ссылок не изменяются. Этот тип изменений не порождает событий.
   - Для событий, извещающих об изменениях в коллекции, определяется свой делегат. События регистрируются в специальных классах-слушателях.
2. Для событий определить делегат CollectionHandler с сигнатурой:
   void CollectionHandler (object source, CollectionHandlerEventArgs args);
3. Определить класс CollectionHandlerEventArgs, производный от класса System.EventArgs, который содержит
   - открытое автореализуемое свойство типа string с названием коллекции, в которой произошло событие;
   - открытое автореализуемое свойство типа string с информацией о типе изменений в коллекции;
   - открытое автореализуемое свойство для ссылки на объект, с которым связаны изменения;
   - конструкторы для инициализации класса;
   - перегруженную версию метода string ToString() для формирования строки с информацией обо всех полях класса.
4. В новую версию класса MyNewCollection добавить
   - открытое автореализуемое свойство типа string с названием коллекции;
   - метод bool Remove (int j) для удаления элемента с номером j ; если в списке нет элемента с номером j, метод возвращает значение false;
   - индексатор (с методами get и set) с целочисленным индексом для доступа к элементу с заданным номером.
5. В класс MyNewCollection добавить два события типа CollectionHandler.
   - CollectionCountChanged, которое происходит при добавлении нового элемента в коллекцию или при удалении элемента из коллекции; через объект CollectionHandlerEventArgs событие передает имя коллекции, строку с

информацией о том, что в коллекцию был добавлен новый элемент или из нее был удален элемент, ссылку на добавленный или удаленный элемент;
    - CollectionReferenceChanged, которое происходит, когда одной из ссылок, входящих в коллекцию, присваивается новое значение; через объект CollectionHandlerEventArgs событие передает имя коллекции, строку с информацией о том, что был заменен элемент в коллекции, и ссылку на новый элемент.
6. Событие CollectionCountChanged бросают следующие методы класса MyNewCollection
    - AddDefaults();
    - Add (object[]) ;
    - Remove (int index).
7. Событие CollectionReferenceChanged бросает метод set индексатора, определенного в классе MyNewCollection.
8. Определить класс Journal, который можно использовать для накопления информации об изменениях в коллекциях типа MyNewCollection. Класс Journal хранит информацию в списке объектов типа JournalEntry. Каждый элемент списка содержит информацию об отдельном изменении, которое произошло в коллекции. Класс JournalEntry содержит
    - открытое автореализуемое свойство типа string с названием коллекции, в которой произошло событие;
    - открытое автореализуемое свойство типа string с информацией о типе изменений в коллекции;
    - открытое автореализуемое свойство типа string с данными объекта, с которым связаны изменения в коллекции;
    - конструктор для инициализации полей класса;
    - перегруженную версию метода string ToString().

9. Класс Journal содержит
    - коллекцию элементов типа JournalEntry (закрытое поле);
    - обработчики событий CollectionCountChanged и CollectionReferenceChanged, которые добавляют новый элемент JournalEntry в коллекцию; для инициализации JournalEntry используется информация из объекта CollectionHandlerEventArgs, который передается вместе с событием;
    - перегруженную версию метода string ToString() для формирования строки с информацией обо всех элементах массива.

10. Написать демонстрационную программу, в которой:
    - создать две коллекции MyNewCollection.
    - Создать два объекта типа Journal, один объект Journal подписать на события CollectionCountChanged и CollectionReferenceChanged из первой коллекции, другой объект Journal подписать на события CollectionReferenceChanged из обеих коллекций.
11. Внести изменения в коллекции MyNewCollection
    - добавить элементы в коллекции;
    - удалить некоторые элементы из коллекций;
    - присвоить некоторым элементам коллекций новые значения.
12. Вывести данные обоих объектов Journal.

При работе в среде VisualStudio необходимо уметь

- средствами Solution Explorer добавить в проект новый класс;

- с помощью диаграммы классов(Class Diagram) добавить в класс методы, поля и свойства;
- с помощью диаграммы классов добавить в класс перегруженную (override) версию виртуального метода.