Лабораторная работа №3. Событийно-ориентированное программирование.

1 Цель и порядок работы

Познакомиться с механизмами событийно-ориентированного программирования на языке C#, такими как механизм обработки событий и исключительные ситуации.

Порядок выполнения работы:
- ознакомиться с описанием лабораторной работы;
- получить задание у преподавателя, согласно своему варианту;
- написать программу и отладить ее на ЭВМ.

2 Краткая теория

2.1 Обработка событий

Событие — это автоматическое извещение о каком-либо произошедшем действии. События являются членами класса и объявляются с использованием ключевого слова event. Механизм событий основан на использовании делегатов.

Синтаксис:

event *имя_делегата имя_обьекта*;

Широковещательные события

События могут активизировать несколько обработчиков, в том числе те, что определены в других объектах. Такие события называются широковещательными. Широковещательные события создаются на основе многоадресных делегатов.

Пример:

```
// Объявление делегата, на основе которого будет
// определено событие.
delegate void MyEventHandler () ;
// Объявление класса, в котором инициируется событие.
class MyEvent
{
        public event MyEventHandler activate;
        // В этом методе инициируется событие.
        public void fire()
                { if (activate != null) activate(); }
}
class X
{
        public void Xhandler()
        {
                Console.WriteLine("Событие получено объектом класса X.");
```

```
        }
}
class Y
{
        public void Yhandler()
        {
                Console.WriteLine("Событие получено объектом класса Y.");
        }
}
class EventDemo
{
        static void handler()
        {
                Console.WriteLine("Событие получено объектом класса
        EventDemo.")
        }
        public static void Main()
        {
                MyEvent evt = new MyEvent();
                X xOb = new X();
                Y yOb = new Y();
                // Добавление методов handler (), Xhandler()
                // и Yhandler() в цепочку обработчиков события.
                evt.activate += new MyEventHandler(handler);
                evt.activate += new MyEventHandler(xOb.Xhandler);
                evt.activate += new MyEventHandler(yOb.Yhandler);
                evt.fire();
                Console.WriteLine();
                evt.activate -= new MyEventHandler(xOb.Xhandler);
                evt.fire();
        }
}
```

2.2 Исключительные ситуации

Исключение представляет собой ошибку, происходящую во время выполнения программы. С помощью подсистемы обработки исключений для C# можно обрабатывать такие ошибки, не вызывая краха программы.

Обработка исключений в C# выполняется с применением четырех ключевых слов: try, catch, throw и finally. Эти ключевые слова образуют взаимосвязанную подсистему, в которой использование одного из ключевых слов влечет за собой использование других.

Основа обработки исключений основана на использовании блоков try и catch.

Синтаксис:

try
{*Блок_кода_для_которого_выполняется_мониторинг_ошибок*}
catch (*ExcepType1 exOB1*
{*Обработчик_исключений_ExcepType1*}
catch (*ExcepType2 exOB2*)
{*Обработчик_исключений_ExcepType2*}

Основные системные исключения приведены в следующей таблице:

Исключение	**Значение**
ArrayTypeMismatchException	Тип сохраненного значения несовместим с типом массива
DivideByZeroException	Предпринята попытка деления на ноль.
IndexOutOfRangeException	Индекс массива выходит за пределы диапазона.
InvalidCastException	Некорректное преобразование в процессе выполнения.
OutOfMemoryException	Вызов new был неудачным из-за недостатка памяти.
OverflowException	Переполнение при выполнении арифметической операции.
StackOverflowException	Переполнение стека.

Тип исключения в операторе catch должен соответствовать типу перехватываемого исключения. Неперехваченное исключение непременно приводит к досрочному прекращению выполнения программы.

Для выполнения перехвата исключений вне зависимости от их типа (перехват всех исключений) возможно использование catch без параметров.

Возврат из исключения

Так как оператор catch не вызывается из программы, то после выполнения блока catch управление не передается обратно оператору программы, при выполнении которого возникло исключение. Выполнение программы продолжается с операторов, находящихся после блока catch.

С целью предотвращения этой ситуации возможно указание блока кода, который вызывается после выхода из блока try/catch, с помощью блока finally в конце

последовательности try/catch. Общая форма конструкции try/catch, включающей блок finally, показана ниже:

try
{*Блок кода, выполняющий мониторинг ошибок*}
catch (*ExcepType1 exOB1*)
{*Обработка исключения ExcepType1*}
catch (*ExcepType2 exOB2*)
{*Обработка исключения ExcepType2*}
finally
{*Код блока finally*}

Блок finally будет вызываться независимо от того, появится исключение или нет, и независимо от причин возникновения такового.

Генерация исключений

Исключения автоматически генерируются системой. Однако исключение может быть сгенерировано и посредством оператора throw.

Синтаксис:
throw exceptOb;

Исключение, перехваченное одним оператором catch, может генерироваться повторно, благодаря чему оно может перехватываться внешним оператором catch. Для этого указывается ключевое слово throw без имени исключения.

Наследование классов исключений

Можно создавать заказные исключения, выполняющие обработку ошибок в пользовательском коде. Генерирование исключений не представляет особых сложностей. Просто определите класс, наследуемый из класса Exception. В качестве общего правила руководствуйтесь тем, что определенные пользователем исключения наследуются из класса ApplicationException, так как они представляют собой иерархию зарезервированных исключений, связанных с приложениями. Наследуемые классы не нуждаются в фактической реализации в каком-либо виде, поскольку само их существование в системе типов данных позволяет воспользоваться ими в качестве исключений.

Создаваемые пользователем классы исключений автоматически получают доступные для них свойства и методы, определенные в классе Exception.

3 Контрольные вопросы

1) Что понимается под термином «событие»?

2) Являются ли события членами классов?

3) Какое ключевое слово языка C# используется для описания событий?

4) На каком механизме языка C# основана поддержка событий?

5) Приведите синтаксис описания события в общем виде. Проиллюстрируйте его фрагментом программы на языке C#.

6) Что понимается под термином «широковещательное событие»?

7) На основе какого механизма языка C# строятся широковещательные события?

8) Приведите синтаксис описания широковещательного события в общем виде. Проиллюстрируйте его фрагментом программы на языке C#.

9) Что понимается под термином «исключительная ситуация (исключение)»?

10) В чем состоит значение механизма исключений в языке C#?

11) Какие операторы языка C# используются для обработки исключений?

12) Какие операторы языка C# являются важнейшими для обработки исключений?

13) Приведите синтаксис блока try…catch в общем виде. Проиллюстрируйте его фрагментом программы на языке C#.

14) Приведите пять видов основных системных исключений.

15) Необходимо ли обеспечивать соответствие типов исключения в операторе catch типу перехватываемого исключения?

16) Что происходит в случае неудачного перехвата исключения?

17) В каком случае возможно использование оператора языка C# catch без параметров?

18) Каким образом осуществляется возврат в программу после обработки исключительной ситуации?

19) Какой оператор языка C# используется для обеспечения возврата в программу после обработки исключения?

20) Приведите синтаксис блока finally (в составе оператора try…catch) в общем виде. Проиллюстрируйте его фрагментом программы на языке C#.

21) Зависит ли вызов блока finally от наличия исключения?

22) Какие способы генерации исключений Вам известны?

23) Что является источником автоматически генерируемых (неявных) исключений?

24) Каким образом возможно осуществить явную генерацию исключений?

25) Какой оператор языка C# используется для явной генерации исключений?

26) Приведите синтаксис оператора throw в общем виде. Проиллюстрируйте его фрагментом программы на языке C#.

27) Каким образом осуществляется повторный перехват исключений в языке C#?

28) Возможно ли создавать специализированные исключения для обработки ошибок в коде пользователя?

29) Какой системный класс является базовым для создания исключений?

30) На основе какого системного класса осуществляется генерация пользовательских исключений?

31) Необходима ли явная реализация классов, наследуемых от системных исключений?

32) Каким образом обеспечивается обращение к свойствам и методам системных исключений?

4 Задание

4.1 Базовый уровень

1. Реализовать класс «Точка», содержащий в качестве полей название точки (A, B, C и т.д.) и значение координат точки (X, Y). В классе предусмотреть:

- конструкторы по умолчанию и конструкторы с параметрами;
- свойства.

2. Реализовать класс геометрическая фигура на плоскости (геометрическая фигура выбирается согласно варианта).

В классе предусмотреть:

- поле – массив класса «Точка»;
- индексаторы, свойства;
- конструкторы;
- метод вычисления площади фигуры;
- событие (площадь фигуры равна 1). Обработчик события должен выводить старое и новое значение площади на экран (старое значение площади передавать в качестве параметра события);
- предусмотреть возможность обработки исключений при вводе данных (стандартные виды исключений).

Варианты заданий:

0) треугольник;

1) квадрат;

2) ромб;

3) трапеция;

4) прямоугольник;

5) параллелограмм;

6) окружность (вместо координат вершины – координаты центра круга (класс «Точка») и радиус);

7) ломанная (вместо площади – длина ломанной).

Вариант определяется следующим образом: номер студента по списку mod 8).

4.2 Повышенный уровень

В дополнение к базовому уровню реализовать свой класс исключение, основанный на классе Exception, которое возникает при невозможности построения данной фигуры исходя из введенных координат вершин фигуры. Реализовать реакцию на данное исключение на основе события и обработчик этого события. В качестве параметров передавать объект класса «Точка» вызвавший исключение.