Делегаты, события и лямбды

Делегаты – это указатели на методы и с помощью делегатов мы можем вызвать данные методы.

Для объявления делегата используется ключевое слово delegate, после которого идет возвращаемый тип, название и параметры.

Например

```
1  delegate void Message();
```

Например

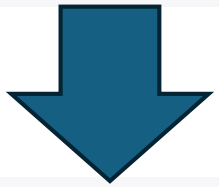
Делегат Message в качестве возвращаемого типа имеет тип **void (то есть ничего не возвращает)** и не принимает никаких параметров.

Рассмотрим применение этого делегата:

```
1  Message mes;                // 2. Создаем переменную делегата
2  mes = Hello;                // 3. Присваиваем этой переменной адрес метода
3  mes();                      // 4. Вызываем метод
4
5  void Hello() => Console.WriteLine("Hello                ");
6
7  delegate void Message(); // 1. Объявляем делегат
```

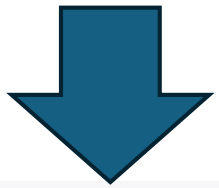
Прежде всего сначала необходимо определить сам делегат:

```
1  delegate void Message(); // 1. Объявляем делегат
```

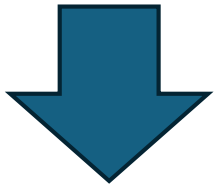


Для использования делегата объявляется переменная этого делегата:

```
1  Message mes; // 2. Создаем переменную делегата
```



```
1  mes = Hello; // 3. Присваиваем этой переменной адрес метода
```



Затем через делегат вызываем метод, на который ссылается данный делегат:

```
1  mes(); // 4. Вызываем метод
```

Место определения Делегата

Как и другие типы, делегат определяется **в конце кода**. Но в принципе делегат можно определять внутри класса

Место определения Делегата

```
1  class Program
2  {
3      delegate void Message(); // 1. Объявляем делегат
4      static void Main()
5      {
6          Message mes;            // 2. Создаем переменную делегата
7          mes = Hello;            // 3. Присваиваем этой переменной адрес метода
8          mes();                  // 4. Вызываем метод
9
10         void Hello() => Console.WriteLine("Hello");
11     }
12 }
```

Место определения Делегата

Либо вне класса:

```
1  delegate void Message(); // 1. Объявляем делегат
2  class Program
3  {
4      static void Main()
5      {
6          Message mes;             // 2. Создаем переменную делегата
7          mes = Hello;             // 3. Присваиваем этой переменной адрес метода
8          mes();                   // 4. Вызываем метод
9
10         void Hello() => Console.WriteLine("Hello");
11     }
12 }
```

Параметры и результат делегата

```
1  Operation operation = Add;       // делегат указывает на метод Add
2  int result = operation(4, 5);    // фактически Add(4, 5)
3  Console.WriteLine(result);       // 9
4
5  operation = Multiply;            // теперь делегат указывает на метод Multiply
6  result = operation(4, 5);        // фактически Multiply(4, 5)
7  Console.WriteLine(result);       // 20
8
9  int Add(int x, int y) => x + y;
10
11 int Multiply(int x, int y) => x * y;
12
13 delegate int Operation(int x, int y);
```

Присвоение ссылки на метод

Выше переменной делегата напрямую присваивался метод. Есть еще один способ - создание объекта делегата с помощью конструктора, в который передается нужный метод:

```
1  Operation operation1 = Add;
2  Operation operation2 = new Operation(Add);
3
4  int Add(int x, int y) => x + y;
5
6  delegate int Operation(int x, int y);
```

Присвоение ссылки на метод

Но надо учитывать, что во внимание также принимаются модификаторы ref, in и out.
Например, пусть у нас есть делегат

Например, пусть у нас есть делегат:

```
1  delegate void SomeDel(int a, double b);
```

Этому делегату соответствует, например, следующий метод:

```
1  void SomeMethod1(int g, double n) { }
```

А следующие методы НЕ соответствуют:

```
1  double SomeMethod2(int g, double n) { return g + n; }
2  void SomeMethod3(double n, int g) { }
3  void SomeMethod4(ref int g, double n) { }
4  void SomeMethod5(out int g, double n) { g = 6; }
```

Объединение делегатов

Делегаты можно объединять в другие делегаты. Например:

```
1  Message mes1 = Hello;
2  Message mes2 = HowAreYou;
3  Message mes3 = mes1 + mes2; // объединяем делегаты
4  mes3(); // вызываются все методы из mes1 и mes2
5
6  void Hello() => Console.WriteLine("Hello");
7  void HowAreYou() => Console.WriteLine("How are you?");
8
9  delegate void Message();
```

Сильная сторона делегатов состоит в том, что они позволяют делегировать выполнение некоторому коду извне. И на момент написания программы мы можем не знать, что за код будет выполняться. Мы просто вызываем делегат. А какой метод будет непосредственно выполняться **при вызове делегата, будет решаться потом**.

АНОНИМНЫЕ МЕТОДЫ

Анонимные методы используются для создания экземпляров делегатов.

АНОНИМНЫЕ МЕТОДЫ

Анонимные методы используются для создания экземпляров делегатов.

АНОНИМНЫЕ МЕТОДЫ

```
1  delegate(параметры)
2  {
3      // инструкции
4  }
```

АНОНИМНЫЕ МЕТОДЫ

Анонимный метод не может существовать сам по себе, он используется для инициализации экземпляра делегата

АНОНИМНЫЕ МЕТОДЫ

```
1  ShowMessage("hello!", delegate (string mes)
2  {
3      Console.WriteLine(mes);
4  });
5
6  static void ShowMessage(string message, MessageHandler handler)
7  {
8      handler(message);
9  }
10
11 delegate void MessageHandler(string message);
```

ЛЯМБДЫ

Лямбда-выражения представляют упрощенную запись анонимных методов.

ЛЯМБДЫ

Лямбда-выражения имеют следующий синтаксис: слева от лямбда-оператора => определяется список параметров, а справа блок выражений, использующий эти параметры:

```
1 (список_параметров) => выражение
```

ЛЯМБДЫ

```
1  Message hello = () => Console.WriteLine("Hello");
2  hello();          // Hello
3  hello();          // Hello
4  hello();          // Hello
5
6  delegate void Message();
```

ПАРАМЕТРЫ ЛЯМБДЫ

```
1  Operation sum = (x, y) => Console.WriteLine($"{x} + {y} = {x + y}");
2  sum(1, 2);         // 1 + 2 = 3
3  sum(22, 14);     // 22 + 14 = 36
4
5  delegate void Operation(int x, int y);
```

ПАРАМЕТРЫ ЛЯМБДЫ

```
1  var sum = (x, y) => Console.WriteLine($"{x} + {y} = {x + y}");   // ! Ошибка
```

```
1  var sum = (int x, int y) => Console.WriteLine($"{x} + {y} = {x + y}");
2  sum(1, 2);         // 1 + 2 = 3
3  sum(22, 14);     // 22 + 14 = 36
```

СОБЫТИЯ

События сигнализируют системе о том, что произошло определенное действие. И если нам надо отследить эти действия, то как раз мы можем применять события.

СОБЫТИЯ

События объявляются в классе с помощью ключевого слова **event**, после которого указывается тип делегата, который представляет событие:

```
1  delegate void AccountHandler(string message);
2  event AccountHandler Notify;
```

СОБЫТИЯ

Определив событие, мы можем его вызвать в программе как метод, используя имя события:

```
1  Notify("Произошло действие");
```

```
1  if(Notify !=null) Notify("Произошло действие");
```

КОНСТРУКЦИЯ TRY..CATCH..FINALLY

При выполнении программы возникают ошибки, которые трудно предусмотреть или предвидеть такие ситуации называются **исключениями**

```
try
{

}
catch
{

}
finally
{

}
```

При использовании блока **try...catch..finally** вначале выполняются все инструкции **в блоке try**. Если в этом блоке не возникло исключений, то после его выполнения начинает выполняться **блок finally**

При использовании блока **try...catch..finally** вначале выполняются все инструкции **в блоке try**. Если в этом блоке не возникло исключений, то после его выполнения начинает выполняться **блок finally**

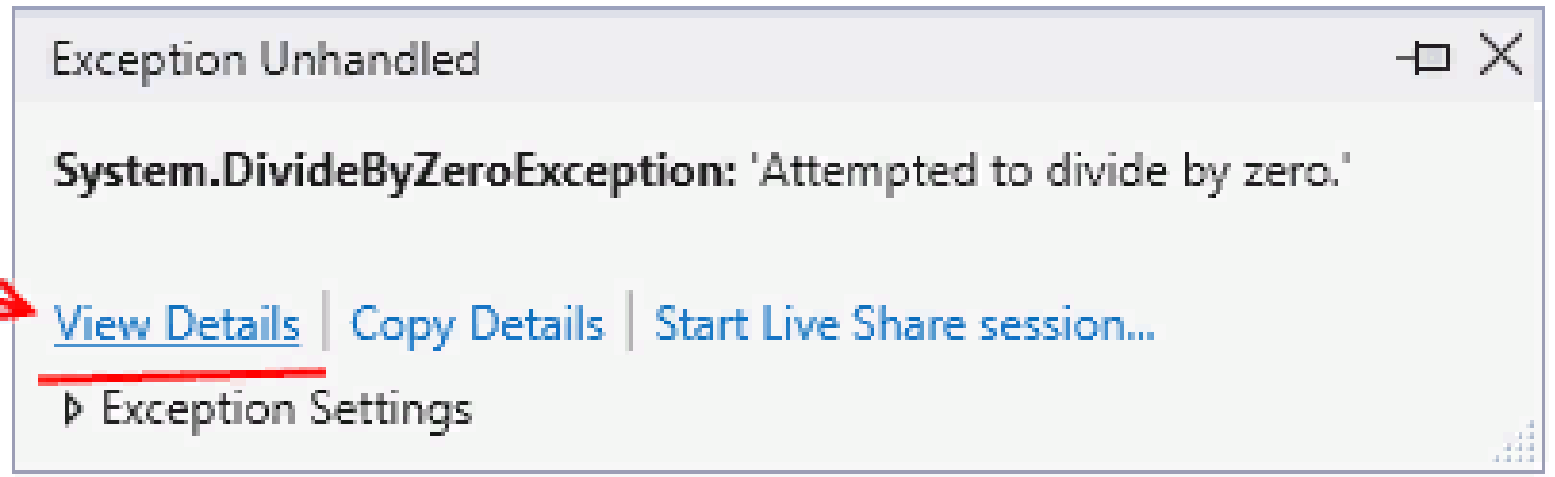
```
1  int x = 5;
2  int y = x / 0;
3  Console.WriteLine($"Результат: {y}");
4  Console.WriteLine("Конец программы");
```

Program.cs

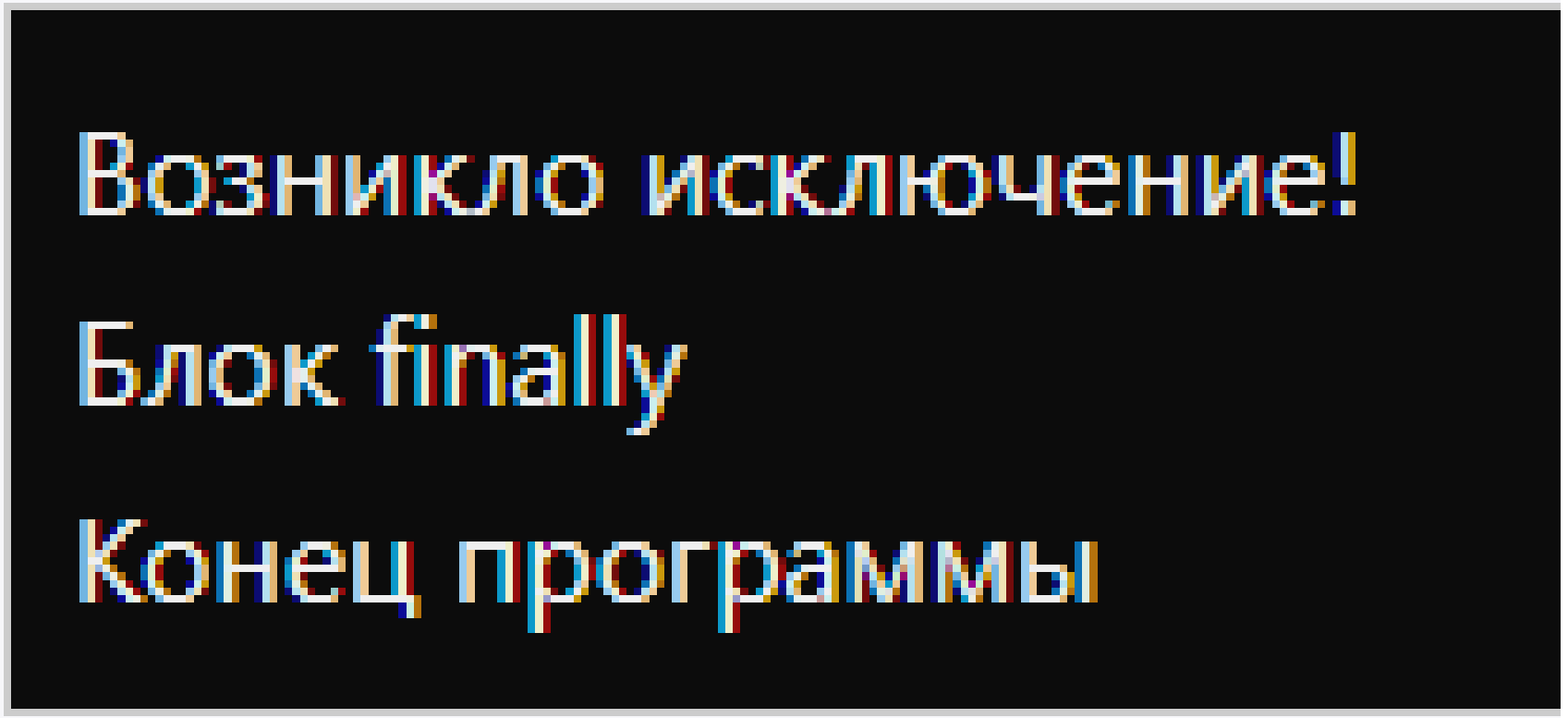
HelloApp

```
1   int x = 5;
2   int y = x / 0;
3   Console.WriteLine($"Результат: {y}");
4   Console.WriteLine("Конец программы");
```

посмотреть информацию об исключении



```
try
{
    int x = 5;
    int y = x / 0;
    Console.WriteLine($"Результат: {y}");
}
catch
{
    Console.WriteLine("Возникло исключение!");
}
finally
{
    Console.WriteLine("Блок finally");
}
Console.WriteLine("Конец программы");
```


Возникло исключение!
Блок finally
Конец программы

Определение блока catch

```
1  catch
2  {
3      // выполняемые инструкции
4  }
```

```
1  catch (тип_исключения)
2  {
3      // выполняемые инструкции
4  }
```

```
try
{
    int x = 5;
    int y = x / 0;
    Console.WriteLine($"Результат: {y}");
}
catch(DivideByZeroException)
{
    Console.WriteLine("Возникло исключение DivideByZeroException");
}
```

Базовым для всех типов исключений является **тип Exception.**

InnerException, Message, Source, StackTrace, Ta

ГЕНЕРАЦИЯ ИСКЛЮЧЕНИЯ И ОПЕРАТОР THROW

C# также позволяет генерировать исключения вручную с помощью оператора throw

```
try
{
    Console.Write("Введите имя: ");
    string? name = Console.ReadLine();
    if (name== null || name.Length < 2)
    {
        throw new Exception("Длина имени меньше 2 символов");
    }
    else
    {
        Console.WriteLine($"Ваше имя: {name}");
    }
}
catch (Exception e)
{
    Console.WriteLine($"Ошибка: {e.Message}");
}
```