1)Делегат представляет собой объект, который может ссылаться на метод.

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

3)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 |

Далее в делегат передается адрес определенного метода (в нашем случае метода Hello). Обратите внимание, что данный метод имеет тот же возвращаемый тип и тот же набор параметров (в данном случае отсутствие параметров), что и делегат.

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

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

|  |  |
| --- | --- |
| 1 | mes(); // 4. Вызываем метод |

Вызов делегата производится подобно вызову метода.

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

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

Оба способа равноценны.

4)В примерах выше делегат вызывался как обычный метод. Если делегат принимал параметры, то при его вызове для параметров передавались необходимые значения:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | Message mes = Hello;  mes();  Operation op = Add;  int n = op(3, 4);  Console.WriteLine(n);    void Hello() => Console.WriteLine("Hello");  int Add(int x, int y) => x + y;    delegate int Operation(int x, int y);  delegate void Message(); |

Другой способ вызова делегата представляет метод **Invoke()**:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | Message mes = Hello;  mes.Invoke(); // Hello  Operation op = Add;  int n = op.Invoke(3, 4);  Console.WriteLine(n);   // 7    void Hello() => Console.WriteLine("Hello");  int Add(int x, int y) => x + y;    delegate int Operation(int x, int y);  delegate void Message(); |

Если делегат принимает параметры, то в метод Invoke передаются значения для этих параметров.

Следует учитывать, что если делегат пуст, то есть в его списке вызова нет ссылок ни на один из методов (то есть делегат равен Null), то при вызове такого делегата мы получим исключение, как, например, в следующем случае:

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | Message? mes;  //mes();        // ! Ошибка: делегат равен null    Operation? op = Add;  op -= Add;      // делегат op пуст  int n = op(3, 4);       // !Ошибка: делегат равен null |

Поэтому при вызове делегата всегда лучше проверять, не равен ли он null. Либо можно использовать метод Invoke и оператор условного null:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1  2  3  4  5  6 | Message? mes = null;  mes?.Invoke();        // ошибки нет, делегат просто не вызывается    Operation? op = Add;  op -= Add;          // делегат op пуст  int? n = op?.Invoke(3, 4);   // ошибки нет, делегат просто не вызывается, а n = null  5) Попросту говоря, групповая адресация — это возможность создать список, или цепочку вызовов, для методов, которые вызываются автоматически при обращении к делегату. Создать такую цепочку нетрудно. Для этого достаточно получить экземпляр делегата, а затем добавить методы в цепочку с помощью оператора + или +=. Для удаления метода из цепочки служит оператор - или -=. Если делегат возвращает значение, то им становится значение, возвращаемое последним методом в списке вызовов. Поэтому делегат, в котором используется групповая адресация, обычно имеет возвращаемый тип void.  6) События объявляются в классе с помощью ключевого слова event, после которого указывается тип делегата, который представляет событие:  delegate void AccountHandler(string message);  event AccountHandler Notify;  В данном случае вначале определяется делегат AccountHandler, который принимает один параметр типа string. Затем с помощью ключевого слова event определяется событие с именем Notify, которое представляет делегат AccountHandler. Название для события может быть произвольным, но в любом случае оно должно представлять некоторый делегат.  8) Лямбда-выражения представляют упрощенную запись анонимных методов. Лямбда-выражения позволяют создать емкие лаконичные методы, которые могут возвращать некоторое значение и которые можно передать в качестве параметров в другие методы. 9)Ковариантность Ковариантность позволяет передать делегату метод, возвращаемый тип которого является производный от возвращаемого типа делегат. То есть если возвращаемый тип делегата Message, то у метод может иметь в качестве возвращаемого типа класс EmailMessage:   |  |  | | --- | --- | | 1  2  3  4  5  6  7  8 | // делегату с базовым типом передаем метод с производным типом  MessageBuilder messageBuilder = WriteEmailMessage; // ковариантность  Message message = messageBuilder("Hello");  message.Print();    // Email: Hello    EmailMessage WriteEmailMessage(string text) => new EmailMessage(text);    delegate Message MessageBuilder(string text); |   Здесь делегат MessageBuilder возвращает объект Message. Однако благодаря ковариантности данный делегат может указывать на метод, который возвращает объект производного типа, например, на метод WriteEmailMessage.  **Контрвариантность**  Контрвариантность позволяет присваить делегату метод, тип параметра которого является более универсальным по отношению к типу параметра делегата. Например, возьмем выше определенные классы Message и EmailMessage и используем их в следующем примере:   |  |  | | --- | --- | | 1  2  3  4  5  6  7 | // делегату с производным типом передаем метод с базовым типом  EmailReceiver emailBox = ReceiveMessage; // контравариантность  emailBox(new EmailMessage("Welcome"));  // Email: Welcome    void ReceiveMessage(Message message) => message.Print();    delegate void EmailReceiver(EmailMessage message); |   Несмотря на то, что делегат в качестве параметра принимает объект EmailMessage, ему можно присвоить метод, у которого параметр представляет базовый тип Message. Может показаться на первый взгляд, что здесь есть некоторое противоречие, то есть использование более универсального тип вместо более производного. Однако в реальности в делегат при его вызове мы все равно можем передать только объекты типа EmailMessage, а любой объект типа EmailMessage является объектом типа Message, который используется в методе.  10) Делегат Func предназначен для инкапсуляции метода, который принимает в качестве параметров от нуля для четырех аргументов и возвращает значение. Что же касается делегата Action, то единственное отличие его от Func – это то, что Action возвращает процедуру. |

Событию нужна ссылка на метод, эту ссылку как раз и обеспечивает делегат.

public delegate void Hand();

public static event Hand D;

private static void Morning()

{

Console.WriteLine("Доброе утро");

}

D += Morning;

D();

Hand hand = new(Morning);

hand();

**Событие** представляет собой автоматическое уведомление о том, что произошло некоторое действие. События действуют по следующему принципу: объект, проявляющий интерес к событию, регистрирует обработчик этого события. Когда же событие происходит, вызываются все зарегистрированные обработчики этого события. Обработчики событий обычно представлены делегатами.

Отличтаются возвр значением и набором типов аргумента

Групповая

адресация

Cоздание списка, или цепочки вызовов , для методов,

которые вызываются автоматически при обращении к

делегату

delegate

void OperationWithArray

Cобытие - это эле мент класса, позволяющий ему

посылать другим объектам уведомления

об изменении своего состояния