1. **Что такое делегат? Как определить делегат?**

Делегаты представляют такие объекты, которые указывают на методы. То есть делегаты - это указатели на методы и с помощью делегатов мы можем вызвать данные методы.

1. **Назначение делегатов.**

<https://habr.com/post/143925/>

делегат - это ссылка на метод или группу методов. Если вам нужно абстрагировать вызов метода от его реализации - вы используете делегаты.

1. **Какие есть способы присваивания делегату адреса метода?**
2. **Поясните назначение метода Invoke.**

Чтобы вызвать метод, на который указывает делегат, надо использовать его метод **Invoke**.

1. **Что такое групповая адресация делегата?**

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.

1. **Как создать событие?**

Объявление события осуществляется на основе ранее объявленного типа делегата. Общая форма объявления события:

event ***делегат\_события* *имя\_события***;

где

* *делегат\_события* – имя типа делегата, который используется для объявления события;
* *имя\_события* – конкретное имя объекта (переменной) типа «событие».

**Пример.** Объявление события с именем MyEvent на основе типа делегата MyDelType.

event MyDelType MyEvent;

1. **Как события связаны с делегатами? Опишите и поясните схему взаимодействия.**

Связь с делегатом означает, что метод, обрабатывающий данное событие, должен принимать те же параметры, что и делегат, и возвращать тот же тип, что и делегат.

1. **Что такое лямбда-выражения? Приведите пример лямбда-выражения с несколькими параметрами.**

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.

Ламбда-выражения имеют следующий синтаксис: слева от лямбда-оператора => определяется список параметров, а справа блок выражений, использующий эти параметры: (список\_параметров) => выражение

1. **Что такое ковариантность и контравариантность делегатов? Что это дает?**

Делегаты также, как и классы и интерфейсы, могут быть обобщенными. И также, как и обобщенные интерфейсы, они могут быть ковариантными и контравариантными. Благодаря ковариантности мы можем присвоить делегату метод, возвращаемым типом которого является тип, производный от того типа, который возвращается делегатом.

В этом случае ковариантность делегата может выглядеть следующим образом:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | delegate Person PersonFactory(string name);  static void Main(string[] args)  {      PersonFactory personDel;      personDel = BuildClient; // ковариантность      Person p = personDel("Tom");      p.Display();      Console.Read();  }  private static Client BuildClient(string name)  {      return new Client(name);  } |

Благодаря контравариантности можно присвоить делегату метод, тип параметра которого представляет базовый класс по отношению к классу параметра, который используется делегатом. Например:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | delegate void ClientInfo(Client client);  static void Main(string[] args)  {      ClientInfo clientInfo = GetPersonInfo; // контравариантность      Client client = new Client("Alice");      clientInfo(client);      Console.Read();  }  private static void GetPersonInfo(Person p)  {      p.Display();  } |

Несмотря на то, что делегат в качестве параметра принимает объект Client, ему можно присвоить метод, принимающий в качестве параметра объект базового типа Person.

**10.Поясните разницу между встроенными делегатами Action и Func.**

Action имеет возвращаемый тип void, Func – возвращаемый тип указывается последним в скобках. Func<in string, out string >