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**
2. **Какие есть способы присваивания делегату адреса метода?**

class Program

{

    delegate int Operation(int x, int y);

    static void Main(string[] args)

    {

        Operation del = Add;

        Operation del2 = new Operation(Add);

        Console.Read();

    }

    private static int Add(int x, int y) { return x + y; }

}

1. **Поясните назначение метода Invoke**

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

Message mes = Hello;

        mes.Invoke();

Message mes = null;

mes?.Invoke();        // ошибки нет, делегат просто не вызывается

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

|  |
| --- |
| Message mes1 = Hello;  mes1 += HowAreYou;  mes1 += Hello;  mes1 += Hello;    mes1(); |

Консольный вывод:

Hello

How are you?

Hello

Hello

1. **Как создать событие?**

События объявляются в классе с помощью ключевого слова **event**, после которого идет название делегата:

|  |  |
| --- | --- |
|  | // Объявляем делегат  public delegate void AccountStateHandler(string message);  // Событие, возникающее при выводе денег  public event AccountStateHandler Withdrawn; |

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. Например:

delegate int Operation(int x, int y);

     static void Main(string[] args)

     {

        Operation operation = (x, y) => x + y;

        Console.WriteLine(operation(10, 20));       // 30

        Console.WriteLine(operation(40, 20));       // 60

        Console.Read();

     }

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

Ковариантность позволяет возвращать из метода объект, тип которого является производным от типа, возвращаемого делегатом.

class Person

{

    public string Name { get; set; }

}

class Client : Person { }

delegate Person PersonFactory(string name);

static void Main(string[] args)

{

    PersonFactory personDel;

    personDel = BuildClient; // ковариантность

    Person p = personDel("Tom");

    Console.WriteLine(p.Name);

    Console.Read();

}

private static Client BuildClient(string name)

{

    return new Client(name);

}

Контрвариантность предполагает возможность передавать в метод объект, тип которого является более универсальным по отношению к типу параметра делегата.

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)

{

    Console.WriteLine(p.Name);

}

1. **Поясните разницу между встроенными делагатами Func и Action.**

Делегат Action является обобщенным, принимает параметры и возвращает значение void. Как правило, этот делегат передается в качестве параметра метода и предусматривает вызов определенных действий в ответ на произошедшие действия.

Еще одним распространенным делегатом является **Func**. Он возвращает результат действия и может принимать параметры.