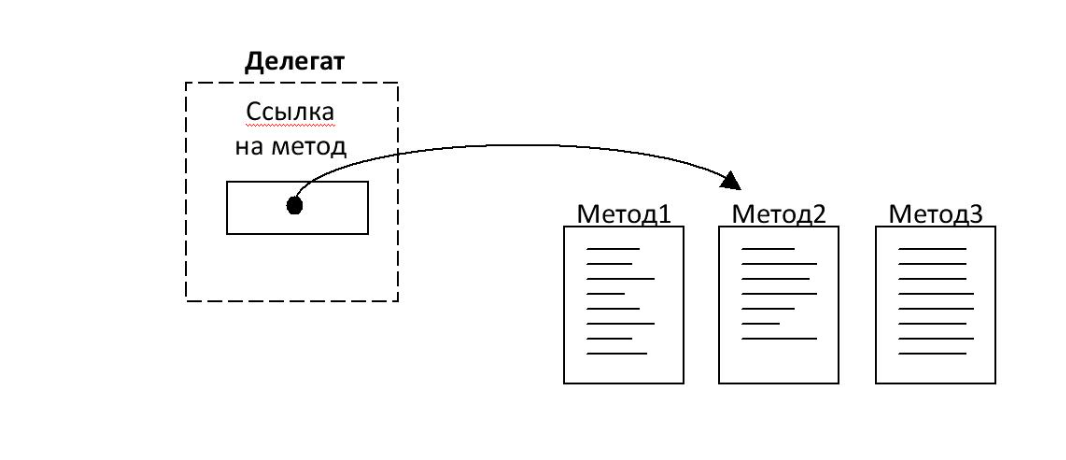
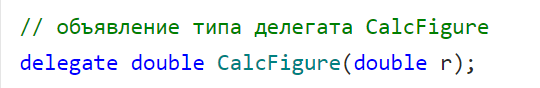
1. Что такое делегат? Как определить делегат?

Делегат в языке C# - представляет или может представлять много методов. Этакое отношение один-ко-многим.

Делегат похож на переменную, которая может ссылаться на один или больше методов с одинаковой сигнатурой и возвращаемым типом.





2. Назначение делегатов.

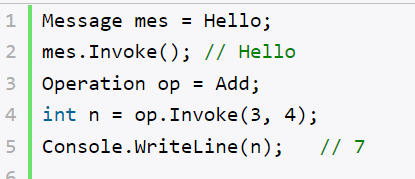
Делегат в C# — это тип, который представляет ссылки на методы со списком параметров и типом возвращаемого значения. Делегаты используются для передачи методов в качестве аргументов к другим методам.

3. Какие есть способы присваивания делегату адреса метода?

Различными способами. Можно непосредственно к делегату добавлять методы при помощи знака ‘+’ (в конструкторе или просто в любом месте программы). А также при объявлении указать название метода в скобках после делегата, при условии, что метод и делегат объявлены в одном классе.

4. Поясните назначение метода Invoke.

Суть метода Invoke довольно проста - он принимает делегат и выполняет его в том потоке, в котором был создан элемент управления, у которого вызывается Invoke.



5. Что такое групповая адресация делегата?

Групповая адресация. Делегаты предоставляют возможность создать список, состоящий из делегата с переданными ему последовательно несколькими методами. Такой список называется групповой адресацией и позволяет создать цепочку вызовов методов, которые вызываются автоматически при обращении к делегату.

6. Как создать событие?

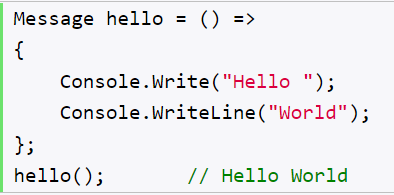


7. Как события связаны с делегатами? Опишите и поясните схему взаимодействия.

События строятся на сигнатуре делегатов => определяют его сигнатуру (параметры и возвращаемый типы). Точно так же, как и делегаты, события содержат “ссылки” на методы.

8. 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.



9. Что такое ковариантность и контравариантность делегатов? Что это дает?

Ковариантность позволяет присвоить делегату метод, возвращаемым типом которого служит класс, производный от класса, указываемого в возвращаемом типе делегата. А контравариантность позволяет присвоить делегату метод, типом параметра которого служит класс, являющийся базовым для класса, указываемого в объявлении делегата.

Каждый из параметров-типов обобщенного делегата или интерфейса должен быть помечен как ковариантный или контравариантный. Это не приводит ни к каким нежелательным последствиям, но позволит применять ваших делегатов в большем количестве сценариев и позволит вам осуществлять приведение типа переменной обобщенного делегата к тому же типу делегата с другим параметром-типом.

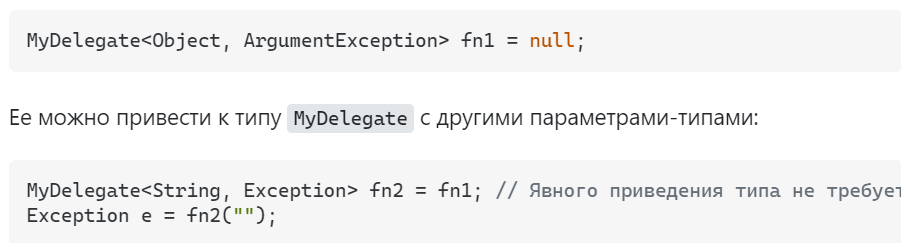
Параметры-типы могут быть:

* **Инвариантными**. Параметр-тип не может изменяться.
* **Контравариантными**. Параметр-тип может быть преобразован от класса к  
  классу, производному от него. В языке C# контравариантный тип  
  обозначается ключевым словом **in**. Контравариантный параметр-тип  
  может появляться только во входной позиции, например, в качестве  
  аргументов метода.
* **Ковариантными**. Аргумент-тип может быть преобразован от класса к одному из его базовых классов. В языке С# ковариантный тип обозначается ключевым словом **out**. Ковариантный параметр обобщенного типа может появляться только в выходной позиции, например, в качестве возвращаемого значения метода.

Предположим, что существует следующий тип делегата:



Здесь параметр-тип T помечен словом **in**, делающим его контравариантным, а параметр-тип TResult помечен словом **out**, делающим его ковариантным. Пусть объявлена следующая переменная:



Это говорит о том, что fn1 ссылается на функцию, которая получает Object и возвращает ArgumentException. Переменная fn2 пытается сослаться на метод, который получает String и возвращает Exception. Так как мы можем передать String методу, которому требуется тип Object (тип String является производным от Object), а результат метода, возвращающего ArgumentException, может интерпретироваться как Exception (тип ArgumentException является производным от Exception), представленный здесь программный код откомпилируется, а **на этапе компиляции будет сохранена безопасность типов**.

10. Поясните разницу между встроенными делегатами Action и Func.

Делегат Func предназначен для инкапсуляции метода, который принимает в качестве параметров от нуля для 16 аргументов и возвращает значение. Что же касается делегата Action, то единственное отличие его от Func – это то, что Action возвращает процедуру.

