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

Делегат представляет **собой тип данных предназначенный для хранения ссылок на методы.**



- `returnType` - тип данных, возвращаемый делегатом.

- `DelegateName` - имя делегата.

- `parameters` - параметры метода, на который будет ссылаться делегат.

***2. Назначение делегатов.***

* **возможность передачи метода как параметра в другой метод**.
* возвращать методы из других методов
* использовать их для определения событий и обработчиков событий.

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

1. \*\*Присвоение **напрямую**:\*\*

Action myDelegate = MyMethod;

Void MyMethod()

{};

2. \*\*Использование ключевого слова `**delegate**`:\*\*

delegate void MyDelegate();

MyDelegate myDelegate = new MyDelegate(MyMethod);

Void MyMethod()

{};

3. \*\*С использованием **анонимных методов**:\*\*

Action myDelegate = delegate ()

{};

4. \*\*Использование **лямбда-выражений** (синтаксис C# 3.0 и выше):\*\*

Action myDelegate = () =>

{};

***4. Поясните назначение метода Invoke.***

В C#, `Invoke` - это метод делегата, **который используется для вызова методов, представленных делегатом.**

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

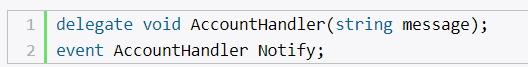
Все экземпляры делегатов обладают возможностью группового вызова (**multicast**). Это означает, что **экземпляр делегата может ссылаться не только на одиночный метод, но и на список методов**. Экземпляры делегатов объединяются **с помощью оператора +=.**

**Делегаты являются неизменяемыми**, так что при использовании оператора += или -= фактически **создается новый экземпляр делегата**, который и присваивается существующей переменной.

***6. Как создать событие?***

Событие (Event) в C# представляет собой **механизм, который позволяет объектам уведомлять другие объекты о том, что что-то произошло**.

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



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

1. Сначала определяется делегат, который будет использоваться для представления обработчиков событий.

2. Затем в классе, генерирующем событие, объявляется событие с использованием ранее определенного делегата.

3. Код, который хочет реагировать на событие, подписывается на него. Подписка происходит путем добавления метода (делегата) в список обработчиков события.

4. Когда происходит событие, код класса, генерирующего событие, вызывает делегат события.

В итоге, делегаты обеспечивают связь между генерирующим событие кодом и обработчиками события. События предоставляют удобный способ для объектов подписываться и реагировать на изменения в других объектах без явного вызова методов.

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

**Лямбда-выражения представляют упрощенную запись анонимных методов**.



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

Ковариантность и контравариантность делегатов - это **концепции, связанные с возможностью использования производных типов вместо базовых типов при работе с делегатами.**

**Ковариантность** применяется к **возвращаемому значению делегата**. Если делегат объявлен как ковариантный, это означает, что **он может использовать производный тип возвращаемого значения вместо базового типа**.

**Контравариантность** применяется к **параметрам метода делегата**. Если делегат объявлен как контравариантный, это означает, что **он может использовать базовый тип вместо производного типа в параметрах метода.**

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

`Action`, `Func`, и `Predicate` - это делегаты в языке программирования C#, предназначенные для представления методов с разным числом параметров и типами возвращаемых значений.

1. \*\*`Action`\*\*:

* предназначенный для представления методов, которые **выполняют какое-то действие, но не возвращают результат.**
* может принимать от 0 до 16 параметров, но не имеет возвращаемого значения.

2. \*\*`Func`\*\*:

* предназначенный для представления методов, **которые возвращают результат**.
* может принимать от 0 до 16 параметров, но последний параметр обязан быть типом возвращаемого значения.

3. \*\*`Predicate`\*\*:

* представляет метод, принимающий один параметр и **возвращающий булево значение.**
* Обычно используется для представления методов, которые выполняют какую-то проверку или условие.