Init only setters

DTO

```
public class SpeakerDTO
{
    public string FirstName { get; }
    public string LastName { get; }
    public string Company { get; }
    public bool IsMicrosoftMVP { get; }
    public string Bio { get; }
    public string GithubNickname { get; }
    public string TwitterNickname { get; }

    public SpeakerDTO(string firstName, string lastName, string company) { [...] }
    public SpeakerDTO(string firstName, string lastName, string company, bool isMicrosoftMvp, string bio, string git
    {
        [..]
    }
}
```

Object initializer

Object initializer has some pros and cons:

- reduces the amount of code that we need to write, for example constructor
- gives freedom to the developer who wants to create the DTO

but...

- requires adding setters once again, what break immutability
- requires default, parameterless constructor

Init only setters

With C# 9 we can use a new **`init only setter`** for the properties. To do this, we need to use keyword **`init`** instead of **`set`**

```
public class SpeakerDTO
{
    public string FirstName { get; init; }
    public string LastName { get; init; }
    public string Company { get; init;}

    public SpeakerDTO() {}
    [...]
}
```

It is a **`special variation of the setter`** that works only when the object is initialized. The **`value can be set when a object is initialized`** (with an initializer or constructor), but **`not after the object initialization is completed`**.

Modifying init only properties

The compiler raises a compilation error when an attempt is made to modify the value of such property.

```
var speaker = new SpeakerDTO
{
    FirstName = "Konrad",
    LastName = "Kokosa",
    Company = "Dotnetos"
};

speaker.Company = "Dotnetos Limited"; // Compilation Error

// Init-only property or indexer 'SpeakerDTO.Company' can only be assigned in an object initializer,
// or on 'this' or 'base' in an instance constructor or an 'init' accessor.
```

Reflection

`init` accessor prevents from modifying the value of the property during the compilation time, but it doesn't guarantee readonly functionality. The value still can be changed using Reflection.

```
var speaker = new SpeakerDTO
{
  FirstName = "Konrad",
  LastName = "Kokosa",
  Company = "Dotnetos"
};

var type = speaker.GetType();
var property = type.GetProperty("Company", BindingFlags.Public | BindingFlags.Instance);

if (property is not null && property.CanWrite)
  property.SetValue(speaker, "DotnetosViaReflection");

Console.WriteLine(speaker.Company); // DotnetosViaReflection
```

Other init actions

When an init accessor is invoked, the instance of the object is known to be in the open construction phase. At that time, the init accessor is allowed to take the same actions as a normal setter can do, plus new additions:

- Call other init accessors, in **`this`** or **`base`** class
- Assign readonly fields declared in **`this`** class

```
class AttendeeDTO
{
    private readonly int _x;
    private string _y;
    public int Z { get; init ; }
    public int XProperty
    {
        get => _x;
        init
        {
            _x = value;
            _y = "";
            Z = 0;
        }
    }
}
```

Mutable readonly fields

`Init` accessor can be also used to set the value of the readonly field because it can happen only during object initialization. It provides similar functionality to the constructor, which is also able to set the value of the readonly field.

```
private readonly string _company;

public string Company
{
    get => _company;
    init => _company = value ?? throw new ArgumentNullException(nameof(Company));
}

public SpeakerDTO(string company)
{
    _company = company ?? throw new ArgumentNullException(nameof(Company));
}
```

Inheritance

If the member (base property) is accessible and the object is known to be in the initialization phase, the property is settable. For example, the following code allows to initialize base property with **`init-only`** accessor, but still fails when trying to initialize base **`get-only`** property.

```
class SpeakerDTO
{
    public int AccommodationCost { get; init; }
    public string FirstName { get; }
}

class OnlineSpeakerDTO : SpeakerDTO
{
    OnlineSpeakerDTO()
    {
        AccommodationCost = 0; // initialization of the `base` is allowed
        FirstName = ""; // initialization of the `base get-only` property is not allowed, property has no setter
    }
}
```

Inheritance and readonly properties

The ability to assign **`readonly`** fields from an **`init`** accessor is limited to fields declared in the **`same type as the accessor`**.

```
class SpeakerDTO
{
    protected readonly bool SpeakerContactedField;
    public bool SpeakerContacted
    {
        get => SpeakerContactedField;
        init => SpeakerContactedField = value;
    }
}
```

Inheritance and readonly properties

`Init` accessor cannot be used to assign readonly fields in a base type.

```
class OnlineSpeakerDTO : SpeakerDTO
{
    private readonly bool _onlineTestPerformed;
    public bool OnlineTestPerformed
    {
        get => _onlineTestPerformed;
        init
        {
            _onlineTestPerformed = value;

            SpeakerContacted = value;      // Ok, `init` accessor can initialize other init accessors
            SpeakerContactedField = value; // Error, field readonly, `init` accessor cannot initialize base members
        }
    }
}
```

> "This rule ensures that type authors remain in control over the mutability behavior of their type. Developers who do not wish to utilize init cannot be impacted from other types choosing to do so" - a note from C# 9.0 proposal

Overriding

When **`init`** accessor is used in a virtual property definition, all overrides must be also marked with the **`init`**. It is not possible to overwrite a simple **`set`** with **`init`**

```
class SpeakerDTO
{
    public virtual int AccommodationCost { get; init; }
}

class OnlineSpeakerDTO : SpeakerDTO
{
    public override int AccommodationCost { get; init; }
}

class OfflineSpeakerDTO : SpeakerDTO
{
// Error: 'OfflineSpeakerDTO.AccommodationCost' must match by init-only of overridden member 'SpeakerDTO.Accommodati
    public override int AccommodationCost { get; set; }
}
```

Other restrictions

- The init accessor can only be used on instance properties
- A property cannot contain both an init and set accessor
- All overrides of a property must have init if the base had init. This rule also applies to interface implementation.

Structs

`Init` accessors are allowed on properties of **`readonly struct`** and **`readonly properties`** of the struct.

```
public readonly struct ReadonlyStructSpeakerDTO
{
    public string FirstName { get; init; }
}

public struct StructSpeakerDTO
{
    public readonly string FirstName { get; init; }
}
```