C# 9 Records

Шипунов Илья

Fortis.online

+7-911-833-15-34

ishipunov@gmail.com

Mutable models

```
public class PersonName
{
    public string
        FirstName { get; set; }
    public string
        LastName { get; set; }
}

var name = new PersonName
{
    FirstName = "Jack",
    LastName = "Sparrow",
};
```

- ✓ Объявление
- ✓ Инициализация
- ✗ Контроль использования
- ✗ Трудно уловимые ошибки

Immutable models

```
public class PersonName
{
    public PersonName(
        string firstName,
        string lastName)
    {
        FirstName = firstName;
        LastName = lastName;
    }

    public string FirstName { get; }
    public string LastName { get; }
}
```

- ✓ Надежность
- ✓ Многопоточность
- ✗ Шаблонный код
- ✗ Ошибки в конструкторе
- ✗ Стоимость поддержки
- ✗ Нет неразрушающего изменения

Readonly structs

```
public readonly struct PersonName
{
    public PersonName(
        string firstName,
        string lastName)
    {
        FirstName = firstName;
        LastName = lastName;
    }

    public string FirstName { get; }
    public string LastName { get; }
}
```

- ✔ GC
- ✔ Immutable
- ✘ Шаблонный код
- ✘ Подводные камни

Личный опыт

```
private static NameDto[] items =
   { new() { Name = "Sparrow" },
     new() { Name = "Turner" } };

public static
   IEnumerable<NameDto> Sort()
{
    return items.Select(x =>
       {
           x.Name = new
                string(x.Name.Reverse()
                        .ToArray());
           return x;
       })
       .OrderBy(x => x.Name);
}
```

- ✘ IEnumerable<T> + mutable DTO
- ✓ IReadOnlyXXX<T>
- ✓ Immutable DTO
- ✓ Маленькие PR

Чтобы
дважды
не
наступали ...

Records

```
public record PersonName
{
    public string? FirstName { get; init; }
    public string? LastName { get; init; }
}

var name = new PersonName
{
    FirstName = "Jack",
    LastName = "Sparrow",
};
//name.LastName = "Mr. Smith";
```

Records

- ✓ Reference type
- ✓ Минимализм
- ✓ Структурное равенство
- ✓ .NET Standard 2.0
- ✓ Наследование
- ✓ Generics
- ✓ Неразрушающее изменение
- ✓ Pattern matching
- ✓ ToString()

Positional records

```
public record PersonName(
  string FirstName,
  string LastName);

var name = new PersonName("Jack", "Sparrow");
//name.LastName = "Mr. Smith";

WriteLine($"{name.FirstName} {name.LastName}");
```

Positional records

- ✓ Плюсы Records
- ✓ Минимализм
- ✓ Immutable по умолчанию
- ✓ Primary constructor
- ✓ Deconstruct

Init-only setters

```
public class PersonName
{
    public string? FirstName { get; init; }
    public string? LastName { get; init; }
}

var name = new PersonName
{
    FirstName = "Jack",
    LastName = "Sparrow",
};
//name.LastName = "Mr. Smith";
```

Init accessors + Readonly fields

```
public class PersonName
{
    private readonly string _firstName = "<unknown>";

    public string FirstName
    {
        get => _firstName;
        init => _firstName = value
                    ?? throw new ArgumentNullException(nameof(FirstName));
    }
    //...
}
```

Init accessors базового класса

```
public class SecretAgentName :
   PersonName
{
    private readonly string? _id;

    public SecretAgentName()
    {
        // Not allowed with get-only
        // but allowed with init
        FirstName = "";
        LastName = "";
    }
```

```
    public string? Id
    {
        get => _id;
        init
        {
            FirstName = "<Classified>";
            LastName = "<Classified>";
            _id = value;
        }
    }
}
```

Init accessors базового класса

```
var stierlitz
   = new SecretAgentName()
{
    FirstName = "Максим",
    LastName = "Исаев",
    Id = "Штирлиц",
};

WriteLine($"{stierlitz.FirstName}
   {stierlitz.LastName},
   {stierlitz.Id}");
//<Classified> <Classified>,
   Штирлиц
```

```
var bond
   = new SecretAgentName()
{
    Id = "007",
    FirstName = "James",
    LastName = "Bond",
};

WriteLine($"{bond.FirstName}
   {bond.LastName},
   {bond.Id}");
//James Bond, 007
```

А Вас, Штирлиц, я попрошу остаться!

Init-only setters в интерфейсах

```
public interface IPersonName
{
    public string FirstName { get; init; }
    public string LastName { get; init; }
}
```

Init-only setters в интерфейсах

```
public static class NameFactory
{
    public static T CreateJackSparrow<T>()
        where T : IPersonName, new()
    {
        var name = new T()
        {
            FirstName = "Jack",
            LastName = "Sparrow",
        };
        return name;
    }
}
```

Readonly structs

```
public readonly struct PersonName
{
    public string FirstName { get; init; }
    public string LastName { get; init; }
}

var name = new PersonName
{
    FirstName = "Jack",
    LastName = "Sparrow",
};
WriteLine($"{name.FirstName} {name.LastName}");
//Jack Sparrow
```

Варианты использования

- Object initializer
- with expression initializer
- Constructor через this или base
- Init accessor через this или base
- Named parameters атрибутов
- Кроме local function или lambda

Breaking changes

- Совместимость с get-only properties
- Обычный setter
- Модификатор CIL – modreq(IsExternalInit)
- Нарушается binary compatibility

.NET Standard 2.0

```
#if !NET5_0

// ReSharper disable once CheckNamespace
namespace System.Runtime.CompilerServices
{
    public sealed class IsExternalInit
    {
    }
}

#endif
```

Структурное равенство

```
var captain = new PersonName()
{
    FirstName = "Jack",
    LastName = "Sparrow",
};
var monkey = new PersonName()
{
    FirstName = "Jack",
    LastName = "Sparrow",
};
```

```
captain.Equals(monkey)
   .Should().BeTrue();
(captain == monkey)
   .Should().BeTrue();

ReferenceEquals(
   captain, monkey)
   .Should().BeFalse();
```

Александр Дюма

Отец

Сын

IEquatable<T> + GetHashCode()

```
var list = new
  List<PersonName>
{
    captain
};

list.Contains(monkey)
  .Should().BeTrue();
```

```
var set = new
  HashSet<PersonName>
{
    captain
};

set.Contains(monkey)
  .Should().BeTrue();
```

Equals и наследование

```
public record PirateName
   : PersonName
{}
public record AnimalName
   : PersonName
{}

var pirate = new PirateName()
{
    FirstName = "Jack",
    LastName = "Sparrow",
};
```

```
var animal = new AnimalName()
{
    FirstName = "Jack",
    LastName = "Sparrow",
};

pirate.Equals(animal)
  .Should().BeFalse();
```

Equals и коллекции

```
public class SequenceEqual<T> :
   IImmutableList<T>
{
    private IImmutableList<T>
        Collection { get; }
    public virtual bool Equals(
        SequenceEqual<T>? other)
    {
        return other != null &&
       Collection.SequenceEqual(other);
    }
    //...
}
```

```
var firstList = ImmutableList.Create(
   "Sparrow", "Turner");
var secondList = ImmutableList.Create(
   "Sparrow", "Turner");
firstList.Equals(secondList)
   .Should().BeFalse();


var firstSeq = new
   SequenceEqual<string>(firstList);
var secondSeq = new
   SequenceEqual<string>(secondList);
firstSeq.Equals(secondSeq)
   .Should().BeTrue();
```

Pattern matching

```
public static string GetFullName(PersonName name)
{
    return name switch
    {
        AnimalName { FirstName: "Jack", LastName: _ } => "Jack",
        PersonName { FirstName: "Jack", LastName: "Sparrow" }
            => "Captain Jack Sparrow",
        { FirstName: var firstName, LastName: var lastName }
            => $"{firstName} {lastName}",
    };
}

var captainJackSparrow = GetFullName(pirate);
WriteLine(captainJackSparrow);
//Captain Jack Sparrow
```

Неразрушающее изменение

```
var name = new PersonName
{
    FirstName = "Jack",
    LastName = "Sparrow",
};
var anotherName = name with { FirstName = "Captain Jack" };

WriteLine($"{name.FirstName} {name.LastName}");
WriteLine($"{anotherName.FirstName} {anotherName.LastName}");
//Jack Sparrow
//Captain Jack Sparrow
```

Копирование

```
var copyOfPirate =
  ((PersonName)pirate) with { };

pirate.Equals(copyOfPirate)
  .Should().BeTrue();

WriteLine(copyOfPirate);
//PirateName { FirstName = Jack, LastName =
  Sparrow }
```

Неглубокое копирование

```
public record FamousPirateName : PirateName
{
    public List<string> Nicknames { get; } = new List<string>();
}

var son = new FamousPirateName
{
    FirstName = "William",
    LastName = "Turner",
};
var father = son with { };
father.Nicknames.Add("Bootstrap Bill");

son.Nicknames.Count.Should().Be(1);
father.Nicknames.Count.Should().Be(1);
```

with + Immutable collections

```
public record FamousPirateName : PirateName
{
    public ImmutableList<string> Nicknames { get; init; }
        = ImmutableList.Create<string>();
}

var son = new FamousPirateName
{
    FirstName = "William",
    LastName = "Turner",
};
var father = son with { Nicknames = son.Nicknames.Add("Bootstrap Bill") };

son.Nicknames.Count.Should().Be(0);
father.Nicknames.Count.Should().Be(1);
```

ToString()

```
var name = new PersonName
{
    FirstName = "Jack",
    LastName = "Sparrow",
};

WriteLine(name);
//PersonName { FirstName = Jack, LastName = Sparrow }
```

Перегрузка ToString()

```
protected virtual bool PrintMembers(
  StringBuilder builder)
{
    builder.Append(
       $"Name = {FirstName}, Surname = {LastName}");
    return true;
}

//PersonName { Name = Jack, Surname = Sparrow }
```

Реализация интерфейсов

```
public interface IPersonName
{
    public string FirstName { get; init; }
    public string LastName { get; init; }
}

public record PersonName(
  string FirstName, string LastName)
       : IPersonName;
```

Наследование

```
public record SuperheroName(
  string FirstName,
  string LastName,
  string Nickname)
    : PersonName(FirstName, LastName);
```

Наследование

```
var ironMan = new SuperheroName(
  "Tony", "Stark", "Iron Man");


WriteLine(ironMan);
//SuperheroName { FirstName = Tony, LastName
  = Stark, Nickname = Iron Man }
```

Deconstruct

```
var (firstName, lastName, nickname) =
  ironMan;


WriteLine(
  $"{firstName} {lastName}, {nickname}");
//Tony Stark, Iron Man
```

Pattern matching

```
public static string GetFullName(PersonName name)
{
    return name switch
    {
        AnimalName("Jack", _) => "Jack",
        PersonName("Jack", "Sparrow") => "Captain Jack Sparrow",
        var (firstName, lastName) => $"{firstName} {lastName}",
    };
}

var captainJackSparrow = GetFullName(name);
WriteLine(captainJackSparrow);
//Captain Jack Sparrow
```

Explicit new() declaration

```
public record SupervillainName(
   string FirstName,
   string LastName,
   string Nickname)
     : PersonName(
          FirstName, LastName)
{
    public SupervillainName()
         : this("<unknown>",
   "<unknown>", "<unknown>")
    {
    }
}
```

```
var doctorOctopus =
   new SupervillainName()
{
     FirstName = "Otto Günther",
     LastName = "Octavius",
     Nickname = "Doctor Octopus",
};

WriteLine(doctorOctopus);
//SupervillainName {
   FirstName = Otto Gunther,
   LastName = Octavius,
   Nickname = Doctor Octopus }
```

Explicit Property Declaration

```
public record SecretAgentName(
   string FirstName, string LastName, string Id)
     : PersonName(FirstName, LastName)
{
     private string Id { get; init; } = Id;
}

var stierlitz = new SecretAgentName(
   "Макс Отто", "фон Штирлиц", "Максим Исаев");

WriteLine($"{stierlitz}");
//SecretAgentName { FirstName = Макс Отто, LastName = фон Штирлиц }
```

Добавление properties

```
public record PersonName(string FirstName, string LastName)
{
    public string? Summary { get; init; }
}

var jackSparrow = new PersonName("Jack", "Sparrow")
{
    Summary = "The Pirate Baron",
};
WriteLine(jackSparrow);
//PersonName { FirstName = Jack, LastName = Sparrow, Summary =
  The Pirate Baron }
```

System.Text.Json

```
var jackSparrowJson = JsonSerializer.Serialize(jackSparrow);

WriteLine(jackSparrowJson);
//{"FirstName":"Jack","LastName":"Sparrow","Summary":"The Pirate
  Baron"}

var jackSparrowObj =
  JsonSerializer.Deserialize<PersonName>(jackSparrowJson);

jackSparrow.Equals(jackSparrowObj)
  .Should().BeTrue();
```

Сравнение объема кода

```
public record PersonName(
  string FirstName, string LastName);
```

Сравнение объема кода

```
public class PersonName : IEquatable<PersonName>
{
    protected virtual Type EqualityContract => typeof(PersonName);

    public string FirstName { get; init; }
    public string LastName { get; init; }

    public PersonName(string FirstName, string LastName)
    {
        this.FirstName = FirstName;
        this.LastName = LastName;
    }

    public override string ToString()
    {
        StringBuilder stringBuilder = new StringBuilder();
        stringBuilder.Append("PersonName");
        stringBuilder.Append(" { ");
        if (PrintMembers(stringBuilder))
        {
            stringBuilder.Append(" ");
        }

        stringBuilder.Append("}");
        return stringBuilder.ToString();
    }

    protected virtual bool PrintMembers(StringBuilder builder)
    {
        builder.Append("FirstName");
        builder.Append(" = ");
        builder.Append((object) FirstName);
        builder.Append(", ");
        builder.Append("LastName");
        builder.Append(" = ");
```

```
    builder.Append((object) LastName);
    return true;
}

public static bool operator !=(PersonName r1, PersonName r2)
{
    return !(r1 == r2);
}

public static bool operator ==(PersonName r1, PersonName r2)
{
    if ((object) r1 != r2)
    {
        if ((object) r1 != null)
        {
            return r1.Equals(r2);
        }

        return false;
    }

    return true;
}

public override int GetHashCode()
{
    return EqualityComparer<Type>.Default.GetHashCode(EqualityContract)
    * -1521134295
                    +
    EqualityComparer<string>.Default.GetHashCode(FirstName) * -
    1521134295
                    +
    EqualityComparer<string>.Default.GetHashCode(LastName);
}

public override bool Equals(object? Obj)
```

```
    {
        return Equals(obj as PersonName);
    }

    public virtual bool Equals(PersonName? Other)
    {
        if ((object?) other != null && EqualityContract ==
        other.EqualityContract &&
                    EqualityComparer<string>.Default.Equals(FirstName,
        other.FirstName))
        {
            return EqualityComparer<string>.Default.Equals(LastName,
        other.LastName);
        }
        return false;
    }

    public virtual PersonName Clone()
    {
        return new PersonName(this);
    }

    protected PersonName(PersonName original)
    {
        FirstName = original.FirstName;
        LastName = original.LastName;
    }

    public void Deconstruct(out string FirstName, out string LastName)
    {
        FirstName = this.FirstName;
        LastName = this.LastName;
    }
}
```

Личное мнение

- ✓ Удобство
- ✓ Минимализм
- ✓ Синтаксис инициализации
- ✓ Init-only setters
- ✓ Структурное равенство
- ✓ Возможность Immutability
- ✓ System.Text.Json

- ✗ Доступ к base
- ✗ Mutability
- ✗ Class primary constructor
- ✗ Primary constructor
- ✗ Nominal records

Primary constructor

```
var jack = new PersonData(
    42,
    "Jack",
    "Sparrow",
    40,
    true,
    true,
    false,
    true,
    true);
```

```
var jack = new PersonData
{
    Id = 42,
    Name = "Jack",
    Surname = "Sparrow",
    Age = 40,
    IsMale = true,
    IsPirate = true,
    IsEunuch = false,
    IsDrinker = true,
    IsDeadman = true,
};
```

Nominal records

```
public record PersonName
{
    string FirstName;
    string LastName;
}
public record PirateName : PersonName
{
    ImmutableList<string> Nicknames;
}
```

Выводы

- Immutable Records – Basic Value Objects/DTO
- Class – Value Objects/Entities/ООП
- Immutable collections – пользуйтесь
- Structs – подводные камни

Литература по Records

- What's new in C# 9.0 - Record types https://docs.microsoft.com/en-us/dotnet/csharp/whats-new/csharp-9#record-types
- Explore record types tutorial https://docs.microsoft.com/en-us/dotnet/csharp/tutorials/exploration/records
- C# 9.0 on the record https://devblogs.microsoft.com/dotnet/c-9-0-on-the-record/
- Records specification https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/proposals/csharp-9.0/records

Литература по Records

- 6 less popular facts about C# 9 records https://tooslowexception.com/6-less-popular-facts-about-c-9-records/
- Avoid C# 9 Record Gotchas https://khalidabuhakmeh.com/avoid-csharp-9-record-gotchas
- Using C# 9 outside .NET 5 https://github.com/dotnet/roslyn/discussions/47701
- modreq(IsExternalInit) https://github.com/dotnet/runtime/issues/34978
- CIL modreq and volatile https://www.red-gate.com/simple-talk/blogs/subterranean-il-custom-modifiers/

Литература по Init-only setters

- Init-only setters specification https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/proposals/csharp-9.0/init
- System.Text.Json https://docs.microsoft.com/en-us/dotnet/standard/serialization/system-text-json-immutability?pivots=dotnet-5-0

Литература по DDD

- C# 9 Records as DDD Value Objects https://enterprisecraftsmanship.com/posts/csharp-records-value-objects/
- Entity vs Value Object: the ultimate list of differences https://enterprisecraftsmanship.com/posts/entity-vs-value-object-the-ultimate-list-of-differences/
- DTO vs Value Object vs POCO https://enterprisecraftsmanship.com/posts/dto-vs-value-object-vs-poco/
- Эванс Э. Предметно-ориентированное проектирование. — М.: Вильямс, 2003

Спасибо за внимание

Вопросы?