# C# 12
# Primary constructors

Никита Маслов

# Основные конструкторы

https://learn.microsoft.com/ru-ru/dotnet/csharp/whats-new/csharp-12

https://learn.microsoft.com/ru-ru/dotnet/csharp/whats-new/tutorials/primary-constructors

# Record primary constructors

```
record Person(int Age);
```

```
class Person
{
    private readonly int <Age>k__BackingField;

    public int Age
    {
        get => <Age>k__BackingField;
        init => Age>k__BackingField = value;
    }

    public Person(int Age)
    {
        <Age>k__BackingField = Age;
    }
}
```

# Class primary constructors. Ожидание

```
class Person(int age);
```

```
class Person
{
    private readonly int _age;

    public Person(int age)
    {
        _age = age;
    }
}
```

# Class primary constructors. Реальность

```
class Person(int age);
```

```
class Person
{
    public Person(int age)
    {
    }
}
```

# Что же это такое?

Classes and structs can have a parameter list, and their base class specification can have an argument list. Primary constructor parameters are in scope throughout the class or struct declaration, and if they are captured by a function member or anonymous function, they are appropriately stored (e.g. as unspeakable private fields of the declared class or struct).

https://github.com/dotnet/csharplang/blob/main/proposals/csharp-12.0/primary-constructors.md

```
public class Person(int age)
{
    public int GetAge()
    {
        return age;
    }
}
```

```
public class Person
{
    private int <age>P;

    public Person(int age)
    {
        <age>P = age;
    }

    public int GetAge()
    {
        return <age>P;
    }
}
```

# Структуры

```
struct Person(int age)
{
    public int GetAge()
    {
        return age;
    }
}
```

```
internal struct Person
{
    private int <age>P;

    public Person(int age)
    {
        <age>P = age;
    }

    public int GetAge()
    {
        return <age>P;
    }
}
```

# Readonly-структуры

```
readonly struct Person(int age)
{
    public int GetAge()
    {
        return age;
    }
}
```

```
internal struct Person
{
    private readonly int <age>P;

    public Person(int age)
    {
        <age>P = age;
    }

    public int GetAge()
    {
        return <age>P;
    }
}
```

# Использование. Поля, свойства и методы

```
class Person(int age)
{
    private readonly int age = age + 1;

    public int Age { get; } = age;

    public int GetAge() => age;
}
```

```
var person = new Person(1);
WriteLine(person.Age); // 1
WriteLine(person.GetAge()); // 2
```

# Наследование

```
class Base(int age);

class Derived(int derivedAge, string name)
  : Base(derivedAge)
{
    // error CS0103: The name 'age' does not exist in the current context
    public int GetBaseAge() => age;
}
```

# Явные конструкторы

```
public class Person(int age)
{
    public Person(int age, string name)
        : this(age)
    {
    }
}
```

# Проблемы

# Изменяемое состояние

```
public class Service(IDependency dependency)
{
    public string Method()
    {
        dependency = null; // !
        return dependency.ToString();
    }
}
```

# Нет проверки аргументов

```
public class Service(IDependency dependency)
{
    private readonly IDependency _dependency =
        dependency ?? throw new ArgumentNullException(nameof(dependency));

    public string Method()
    {
        return _dependency.ToString();
    }
}
```

# Неконсистентное именование

```
public class Person(int age)
{
    public int GetAge()
    {
        // Error CS1061: "Person" не содержит определения "age", и не удалось
найти доступный метод расширения "age", принимающий тип "Person" в качестве
первого аргумента
        return this.age;
    }
}
```

# Неконсистентное именование

```
class Person(int _age);

var person = new Person(_age: 1);
```

# Двойное использование (double storage)

```
class Person(int age)
{
    // CS9124: Parameter is captured into the state
of the enclosing type and its value is also used to
initialize a field, property, or event.
    public int InitializedAge { get; set; } = age;

    public string CapturedAge => $"{age}!";
}
```

```
var p = new Person(1);
WriteLine(p.InitializedAge); // 1
WriteLine(p.CapturedAge);    // 1!

p.InitializedAge++;
WriteLine(p.InitializedAge); // 2
WriteLine(p.CapturedAge);    // 1!
```

https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/proposals/csharp-12.0/primary-constructors#double-storage-warnings

# Почему так случилось?

… But it would also lead to surprising differences from how parameters and locals are captured elsewhere in C#. …

Another problem often raised with this approach is that many developers have different naming conventions for parameters and fields. Which should be used for the primary constructor parameter? Either choice would lead to inconsistency with the rest of the code.

https://github.com/dotnet/csharplang/blob/main/proposals/csharp-12.0/primary-constructors.md#explicit-generated-fields

# Что с этим делать?

# Смириться

- Кортежи (tuple) изменяемые – но это никого не смущает

# Ждать

```
class Person(readonly int age);
```

https://github.com/dotnet/csharplang/blob/main/proposals/readonly-parameters.md

https://github.com/dotnet/csharplang/blob/main/meetings/2023/LDM-2023-07-26.md

https://github.com/dotnet/csharplang/blob/main/meetings/2023/LDM-2023-07-31.md

# Source-генератор

```
using PrimaryParameter.SG;

var p = new Person(1);

partial class Person([Field] int age)
{
    public int GetAgeOk() => _age;

    // Error PC01 : Can't access a
primary parameter ('age') with a
[Field] [RefField], [Property] or
[DoNotUse] attribute, use '_age'
    public int GetAgeError() => age;
}
```

```
// <auto-generated/>
partial class Person
{
    private readonly int _age = age;
}
```

https://github.com/FaustVX/PrimaryParameter

# А как в других языках?

# F#

```
type Person(age: int, name) =
    let ageField = age + 1
    do printfn "the ageField is now %i" ageField
    member this.Age = age
    member this.GetAge() = ageField
    member this.GetName() = name

let person = new Person(1, "3")
printfn "%i"  <| person.Age // 1
printfn "%i"  <| person.GetAge() // 2
printfn "%s"  <| person.GetName() // 3
```

https://learn.microsoft.com/en-us/dotnet/fsharp/language-reference/members/constructors
https://fsharpforfunandprofit.com/posts/classes/

# Kotlin

```
class Person(firstName: String,
             val lastName: String,
             var age: Int) {
    init {
        println("initializer $firstName")
    }
}
```

```
fun main() {
     // initializer FN
    var person = Person("FN", "LN", 1)

    println(person.lastName) // LN
    person.age = 2
    println(person.age) // 2
}
```

https://kotlinlang.org/docs/classes.html#constructors

# Выводы

- Фича – спорная, каждый решает сам, использовать или нет

# Ссылки

1. https://github.com/dotnet/csharplang/blob/main/proposals/csharp-12.0/primary-constructors.md
2. https://github.com/dotnet/csharplang/issues/2691 Primary Constructors
3. https://github.com/dotnet/csharplang/discussions/7109 Preview Feedback: C# 12 Primary constructors
4. https://github.com/dotnet/csharplang/discussions/7667 Feedback on primary constructors
5. https://learn.microsoft.com/en-us/dotnet/csharp/whats-new/tutorials/primary-constructors
6. https://learn.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/instance-constructors#primary-constructors

# Вопросы и ответы

https://t.me/mister_m0j0

https://github.com/m0j0