عنوان مقاله: چگونه از Primary Constructor در سی شارپ ۱۲ استفاده کنیم؟

نویسنده مقاله: تیم فنی نیک‌آموز

تاریخ انتشار: ۱۷ بهمن ۱۴۰۲

منبع: https://nikamooz.com/primary-constructor-in-c-sharp/

Primary Constructor در سی شارپ ۱۲ ، به‌عنوان یک Feature جدید ارائه شده است. این نوع از سازنده، به دلیل دارا بودن نحو (Syntax) منسجم، مزیت‌های مناسبی به شما در برنامه نویسی با سی شارپ ارائه می‌دهد. در مقالات پیشین، به **بررسی ویژگی های جدید .NET 8** و **بررسی بهبودات کارایی آن نسبت به نسخه‌های قبلی** اشاره شد. در این مقاله قصد داریم نحوه کار با Primary Constructor در C# 12 را به‌همراه کدنویسی شرح دهیم.

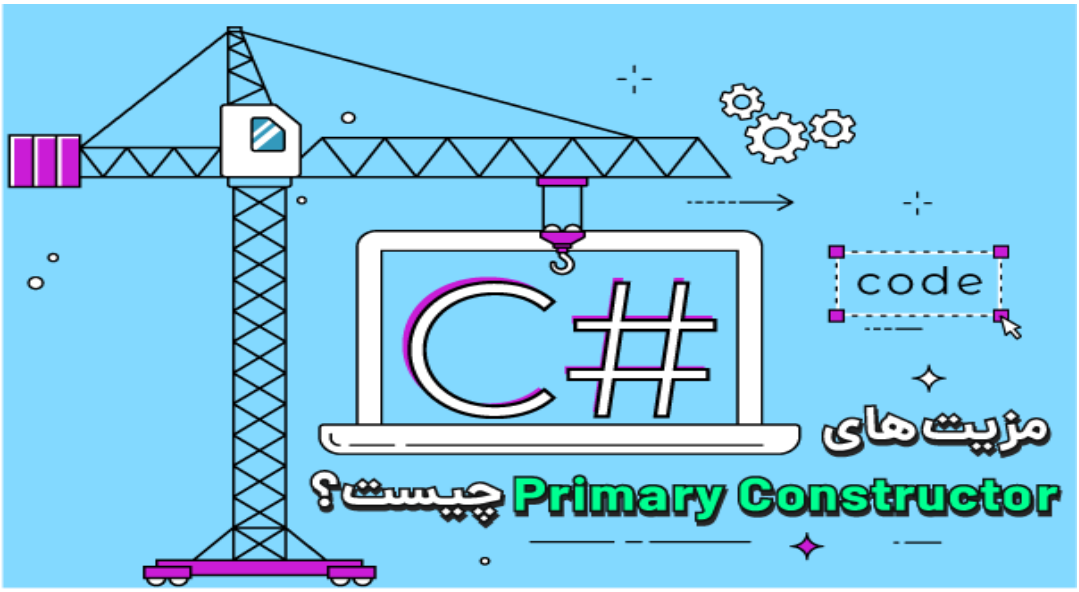
Primary Constructor در C#

با استفاده از Primary Constructor در سی شارپ ، یک رویکرد مناسب و یکپارچه برای ساخت سازنده (Constructor) و راه‌اندازی Property ها در درون کلاس‌ها و Struct ها فراهم می‌شود. Primary Constructor در سی شارپ ۱۲ به هدف کاهش کدهای تکراری، بهبود خوانایی (Readability) کد و تسهیل فرآیند Dependency Injection ارائه شده است.

مزیت های Primary Constructor چیست؟

مزایای استفاده از Primary Constructor در سی شارپ عبارتند از:

- بهبود خوانایی کدها و درک سریع‌تر آن‌ها
- پذیرش مستقیم Dependency ها توسط پارامترها
- عدم نیاز به استفاده از کلمه کلیدی this برای مقداردهی به Property های سازنده
- تولید کدهای بهینه‌شده و بهبوداتی در کارایی
- دسترسی پیش‌فرضِ شفاف و ارث‌بری‌شده از کلاس



آشنایی با Primary Constructor در سی شارپ ۱۲

در نسخه ۱۲ سی شارپ، فیچرهای مختلفی معرفی شده‌اند که یکی از آن‌ها Primary Constructor نام دارد. از نام این فیچر می‌توان حدس زد که گویا این ویژگی در ساختار سازنده‌های کلاس‌ها ایجاد شده است.

ابتدا به بررسی ساختار کلی سازنده در نسخه‌های پیشین سی شارپ پرداخته و پس از آن، Primary Constructor در سی شارپ ۱۲ را بررسی می‌کنیم.

در تکه کد زیر، کلاسی به اسم Person تعریف شده است که یک سازنده برای دریافت نام و سن کاربر دارد.

```
public class Person
{
    private readonly string _name;
    private readonly int _age;

    public Person(string name, int age)
    {
        _name = name;
        _age = age;
    }
}
```

قطعه کد زیر مربوط به نحوه استفاده از Primary Constructor در سی شارپ ۱۲ است.

```
public class Person(string name, int age)
{
    private readonly string _name = name;
    private readonly int _age = age;
}
```

با مشاهده و مقایسه کد مربوط به Primary Constructor در سی شارپ ۱۲ واضح است که تعداد خط‌های کدنویسی نسبت به حالت سنتی کاهش پیدا کرده‌اند و همچنین، آرگومان‌های سازنده کلاس، در کنار نام کلاس قرار گرفته‌اند. در حالی که در روش سنتی، آرگومان‌ها در یک کد بلاک مجزا قرار داشتند.

در این مثال، name و age پارامترهای Primary Constructor نامیده می‌شود که برای مقداردهی اولیه property با نام name_ و age_ استفاده خواهند شد. سپس، این property در سراسر کلاس Person قابل دسترسی خواهد بود.

فیچر Primary Construct تنها درمورد کدنویسی تمیزتر یا کوتاه‌تر به کار نمی‌رود؛ بلکه مشخصه‌های جذاب دیگری نیز دارد. درحقیقت، این فیچر مشخص می‌کند که مقداردهی پارامترهای آن برای تمام instance هایی لازم است که از کلاس گرفته می‌شوند. در این Feature، هر سازنده دیگری از کلاس، مجبور به استفاده از this خواهد بود.

پیش از بررسی بیش‌تر، بهتر است نگاهی به سازنده‌های Parameterlesss بیاندازیم. اگر برای یک کلاس هیچ سازنده‌ای تعریف نشود، سی شارپ برای آن یک سازنده بدون پارامتر تعریف می‌کند و از آن برای ایجاد نمونه استفاده خواهد شد. فرض کنید کلاس Person به‌صورت زیر درنظر گرفته شود:

```
public class Person
{
    public string Name { get; set; }
    public int Age { get; set; }
}
```

در چنین شرایطی، اگر قصد داشتیم یک نمونه از این کلاس تعریف کنیم، باید از روش زیر استفاده می‌کردیم:

```
Person person = new Person();
```

در عمل، هیچ سازنده‌ای تعریف نشده است که پارامتری به‌عنوان ورودی دریافت نکند؛ اما سی شارپ این عمل را در پس‌زمینه برای ما انجام خواهد داد. بنابراین، مثل آن است که کلاس مورد بحث از ابتدا به‌صورت زیر تعریف شده باشد.

```
public class Person
{
    public string Name { get; set; }
    public int Age { get; set; }

    public Person()
    {
        // some codes can be written here
    }
}
```

به این نوع از سازنده، تحت عنوان Parameterless اشاره می‌شود. با این دیدگاه، در ادامه به بررسی بحث Primary Constructor و الزام استفاده از کلیدواژه this در سازنده‌های parameterless می‌پردازیم. این موضوع، در تکه کد زیر شفاف شده است:

```
public class Person(string name, int age)
{
    private readonly string _name = name;
    private readonly int _age = age;

    public Person(): this("NoName", 0) { }
}
```

در مثال فوق، برای کلاس Person یک سازنده از نوع Parameterless اضافه کرده‌ایم. در این سازنده برای مقداردهی پارامتر Primary Constructor ، لازم است از this استفاده شود. اگر این موضوع نادیده گرفته شود، برنامه در هنگام کامپایل با خطا همراه خواهد بود. بدین طریق، پارامترهای سازنده اولیه یا Primary Constructor به‌طور حتمی توسط تمامی سازنده‌ها مقداردهی می‌شوند.

ذکر این نکته ضروری است که Scope پارامترهای Primary Constructor شامل کل کلاس می‌شود؛ این یعنی، هر عضوی از کلاس، می‌تواند به این پارامترها دسترسی داشته باشد، ولی احتمالاً این موضوع برای خارج از کلاس امکان‌پذیر نخواهد بود.

در ادامه نگاهی به Dependency Injection در Primary Constructor بیاندازیم. فرض کنید یک Controller داریم که قرار هست از یک سرویس برای نمایش دمای هوا استفاده کند. این سرویس باید در آن کنترلر، Inject شود. این سرویس به گونه‌ای پیاده‌سازی شده است که از یک اینترفیس به اسم IWeatherService و یک کلاس به اسم WeatherService تشکیل شده باشد. بدیهی است که اینترفیس مذکور از کلاس Weather ارث‌بری کرده و در محلی با استفاده از تکه کد زیر رجیستر شده است.

```
services.AddTransient<IWeather, Weather>();
```

جزئیات پیاده‌سازی سرویس موردنظر، در اینجا مطرح نیست و تنها می‌خواهیم روشی که در سی شارپ ۱۲ معرفی شده را بررسی کنیم. ابتدا، نگاهی به تکه کد زیر بیاندازید:

```
[ApiController]
public class WeatherController(IWeatherService weatherService) :
ControllerBase
{
    [HttpGet]
    public ActionResult<double> GetTemperature()
    {
        return weatherService.GetTemperature();
    }
}
```

با استفاده از Primary Constructor ، سرویس weather را در داخل کنترلر WeatherController اینجکت (Inject) کرده و سپس، در متد GetTemerature از آن استفاده کرده‌ایم.

تا این مرحله مشخص شد که سازنده اولیه یا Primary Constructor برای چه هدفی به وجود آمده و به چه صورتی قابل استفاده است. در قدم بعدی، توجه کنید که به‌طور کلی، یک کلاس به‌همراه سازنده اولیه چه تفاوتی با یک شی از نوع Record دارد.

دو تا شی به نام PersonClass و PersonRecord تعریف می‌کنیم. احتمالاً از روی نوع نامگذاری آن‌ها می‌توان متوجه شد که شی اول، یک کلاس با سازنده اولیه و شی دوم، یک Record است.

نگاهی به تعریف هردو بیاندازیم:

```
public class PersonClass(string name, int age);
public record PersonRecord(string Name, int Age);
```

تعریف هردو شبیه به یکدیگر است. هردوی آن‌ها، دو پارامتر نام و سن رو دریافت می‌کنند. پارامترهای Record را Positional Parameter می‌نامند که از شیوه PascalCase برای نام‌گذاری آن‌ها استفاده می‌شود. پس تا این مرحله، مشخص شد که از حیث ظاهر تفاوت اندکی با یکدیگر دارند. با این وجود، در عملکرد کاملاً متفاوت رفتار می‌کنند. برای اینکه دقیق‌تر متوجه این موضوع شوید، نگاهی به IL Code های تولیدشده برای هر شی بیاندازید.

```
public class PersonRecord
{
    [CompilerGenerated]
    [DebuggerBrowsable(DebuggerBrowsableState.Never)]
    private readonly string <Name>k__BackingField;
    [CompilerGenerated]
    [DebuggerBrowsable(DebuggerBrowsableState.Never)]
    private readonly int <Age>k__BackingField;
```

```
    public PersonRecord(string Name, int Age)
    {
        this.<Name>k__BackingField = Name;
        this.<Age>k__BackingField = Age;
    }

    public string Name
    {
        [CompilerGenerated] get
        {
            return this.<Name>k__BackingField;
        }
        [CompilerGenerated] init
        {
            this.<Name>k__BackingField = value;
        }
    }

    public int Age
    {
        [CompilerGenerated] get
        {
            return this.<Age>k__BackingField;
        }
        [CompilerGenerated] init
        {
            this.<Age>k__BackingField = value;
        }
    }
}
```

همانطور که در کد بالا مشاهده می‌کنید، دو متغیر Public برای نام و سن توسط کامپایلر ایجاد می‌شوند که setter آن به‌صورت init خواهد بود؛ این یعنی، مقداردهی متغیرها فقط ازطریق تنها سازنده کلاس امکان‌پذیر است و از هیچ راه دیگری نمی‌توان آن را ویرایش کرد.

برخلاف record ها، در کلاس‌هایی که سازنده اولیه دارند، پارامترهای آن توسط کامپایلر به‌صورت public ایجاد نمی‌شوند. به همین خاطر، از بیرون از کلاس قابل دستیابی نیستند و اینکه برای کلاس‌های با سازنده اولیه، کلیدواژه with قابل استفاده نیست. برای بررسی این موضوع، به مثال زیر توجه کنید:

```
var person1 = new PersonRecord("Pooya", 37);
var person2 = person1 with { Name = "Nastaran" };

var person3 = new PersonClass("Pooya", 37);

// This line has compile error
var person4 = person3 with { name = "Nastaran" };
```

یک نمونه از شی PersonRecord ایجاد و پارامترهای آن مقداردهی اولیه شدند؛ پس از آن، نتیجه در متغیر person1 ذخیره شد. در قدم بعدی، یک شی دیگر ایجاد شده، با این تفاوت که تنها نام آن متفاوت از نمونه قبلی باشد. این کار را با استفاده از کلمه کلیدی with انجام داده‌ایم. شی دوم را person2 نامیده و سن را مشابه شی person1، سی و هفت سال قرار داده‌ایم؛ با این تفاوت که نام آن به Nastaran تغییر پیدا کرده است.

حال باید بررسی کرد آیا می‌توان همین کار را با کلاس و همراه با سازنده اولیه انجام داد یا خیر. یک نمونه از PersonClass ساخته و پارامترهای آن را مقداردهی اولیه کرده‌ایم؛ نتیجه حاصل در متغیر person3 ذخیره شده است. در قدم بعدی، لازم است یک کلاس نمونه دیگر ایجاد شود؛ توجه کنید که می‌خواهیم نام دیگری برای متغیر name درنظر گرفته شود، اما این کار موفق‌آمیز انجام نشد؛ زیرا در چنین شرایطی، در زمان کامپایل با ارور Syntax مواجه خواهیم شد.

این موضوع با نگاه به IL Code کلاس PersonClass قابل درک خواهد بود.

```
public class PersonClass
{
  [NullableContext(1)]
  public PersonClass(string name, int age)
  {
    base..ctor();
  }
}
```

در قطعه کد فوق، هیچ متغیر دیگری برای کلاسی با سازنده اولیه تعریف نمی‌شود. به این ترتیب، می‌توان درک کرد که چرا امکان استفاده از with وجود ندارد.

جمع بندی

Primary Constructor در سی شارپ ۱۲ ارائه شده است تا با استفاده از آن، میزان کدنویسی کاهش بیابد. در این مقاله به بررسی این نوع از سازنده در C# پرداخته شد. به‌طور کلی، این ویژگی، رویکرد مناسبی برای پیاده‌سازی پروژه به‌حساب می‌آید و شما با مزیت‌های آن در این مطلب آشنا شدید. اگر در بخشی از مطلب دچار ابهام هستید، پیشنهاد می‌شود پرسش‌های خود را مطرح کنید.