C# 12 İle Gelen Yenilikler

C# 12 ve .NET 8 geliştiricilere performans ve verimlilik açısından önemli avantajlar sunuyor. Bu yeni özellikler, gelecekteki performans iyileştirmelerinin temellerini oluşturuyor ve yazılım geliştirmeyi daha kolay ve verimli hale getiriyor.

Primary Constructors

- C#'ın 12. versiyonu, sınıf ve yapılar içinde temel yapılandırıcılar (primary constructors) oluşturmayı mümkün kılıyor. Artık temel yapılandırıcılar sadece kayıt tiplerle sınırlı değil. Temel yapılandırıcı parametreleri, sınıfın tüm gövdesi boyunca erişilebilir hale geliyor. Tüm açıkça bildirilen yapılandırıcılar, this() sözdizimini kullanarak temel yapılandırıcıyı çağırmalıdır, böylece tüm temel yapılandırıcı parametreleri kesin olarak atanır. Bir sınıfa temel yapılandırıcı eklemek, derleyicinin otomatik olarak parametresiz bir yapılandırıcı oluşturmasını engeller. Yapıda, otomatik olarak oluşturulan parametresiz yapılandırıcı, tüm alanları, temel yapılandırıcı parametrelerini de içerecek şekilde o-bit deseniyle başlatır.

- Birincil yapılandırıcı parametreleri için sadece kayıt tiplerde (record class veya record struct tipleri) derleyici tarafından otomatik olarak genel özellikler (public properties) oluşturulur. Kayıt olmayan sınıf ve yapılar, birincil yapılandırıcı parametreleri için her zaman bu davranışı istemeyebilir.

Inline Diziler: Performansı Artıran Yenilik

- InlineArrayAttribute, daha önceki bir .NET 8 önizlemede tanıtılan önemli bir özelliktir. Bu, C#'ın 12. versiyonunda daha da geliştirilerek performansı artıran bir araç haline gelmiştir. Inline diziler, bir türün sürekli bir dizi temsilini tanımlamak için kullanılır ve aşırı taşmaları engellemek için indekslenmesi ve kesilmesi kolaydır. Bu tür veriler, etkin ve hızlı bir şekilde işlenir ve .NET kitaplıkları ve bazı diğer kütüphaneler tarafından yaygın olarak kullanılır.
- Örnek olarak, inline dizilerin kullanımına bakalım

```
private static void InlineArrayAccess(Buffer10<int> inlineArray)
{
for (int i = 0; i < 10; i++)
{
inlineArray[i] = i * i;
}

foreach (int i in inlineArray)
{
Console.WriteLine(i);
}
}
```

- Burada Buffer10 adında bir yapı (struct) kullanılarak inline bir dizi oluşturuyoruz. Bu yapı, InlineArrayAttribute ile işaretlenmiştir ve 10 elemanlı bir dizi temsil eder. İlk 10 sayının karesini hesaplayarak ve yazdırarak bu inline diziyi kullanıyoruz.

Interceptors: Yenilikçi ve Deneysel Bir Yaklaşım

C#'ın 12. versiyonuyla birlikte deneysel bir özellik olan interceptors tanıtıldı. Bu özellik, derleme öncesi (AOT) derleme için daha iyi destek sağlayan gelişmiş senaryolar için tasarlanmıştır. Interceptors, özel kodun belirli yöntem çağrılarını yönlendirmesine olanak tanır. Özellikle kaynak üreteçleri için uygundur ve belirli kodları yeniden yönlendirme yeteneği sunar.

Interceptors'ın nasıl kullanıldığına dair örnek

```
[Features("InterceptorsPreview")]
public class MyService
{
[Interceptor(typeof(MyInterceptor))]
public void DoSomething()
{
// Burada yapılacak işlem
}
}
```

Yukarıdaki örnek, MyService sınıfının DoSomething() yönteminin MyInterceptor adlı özel bir interceptor ile yeniden yönlendirilmesini göstermektedir.

Nameof ile Erişilebilen Öğeler

- C#'ın 12. versiyonuyla birlikte nameof anahtar kelimesi, artık üye adları, başlatıcılar, statik üyeler ve niteliklerle çalışır. Bu özellik, C# dilinin daha esnek ve kullanıcı dostu olmasını sağlar. Özellikle, adları bilinen üyeleri ve nitelikleri kullanırken, daha az hata yapma olasılığını artırır.

```
internal class NameOf
 {
 public string S { get; } = "";
 public static int StaticField;
 public string NameOfLength { get; } = nameof(S.Length);
 public static void NameOfExamples()
 {
 Console.WriteLine(nameof(S.Length));
 Console.WriteLine(nameof(StaticField.MinValue));
 }
 [Description($"String {nameof(S.Length)}")]
 public int StringLength(string s)
 {
 return s.Length;
 }
 }
```

- Yukarıdaki örnekte, nameof anahtar kelimesi, sınıfın içindeki öğelerin adlarına ve niteliklerine erişim sağlamak için kullanılır.

Varsayılan Lambda Parametreleri ve Tüm Tiplere Takma Ad Verme

- Bu yeni versiyon ile gelen iki önemli özellik, yazılım geliştirme deneyimini daha zenginleştiriyor.
- **Varsayılan Lambda Parametreleri:** Artık lambda ifadelerinde parametreler için varsayılan değerler belirleyebilirsiniz. Sözdizimi ve kurallar, herhangi bir yöntem veya yerel fonksiyon için argümanlara varsayılan değer eklemekle aynıdır. Bu, kodunuzu daha esnek ve okunabilir hale getirirken, lambda ifadeleri ile çalışmayı daha kolay ve verimli hale getirir.
- **Tüm Tiplere Takma Ad Verme:** using alias direktifi, artık yalnızca adlandırılmış tipler için değil, tüm tipler için takma adlar oluşturmanıza olanak tanır. Bu, tuple tipleri, dizi tipleri, işaretçi tipleri veya diğer güvensiz tipler için anlamsal takma adlar oluşturmanızı sağlar. Bu sayede, kodunuzun daha anlaşılır ve temiz olmasını sağlayabilirsiniz.

C# 12 Özellikleri ve Preview Sürümleri:

Özellik	Preview Sürümü
Primary constructors	Visual Studio 17.6 preview 2
Lambda ifadelerinde opsiyonel parametreler	Visual Studio 17.5 preview 2
Tüm tiplere takma ad verme	Visual Studio 17.6 preview 3
Inline diziler	Visual Studio 17.7 preview 3
Interceptorlar (Preview)	Visual Studio 17.7 preview 3

Sonuç

- C#'ın 12. versiyonu ve .NET 8'in önizlemesi, yazılım geliştirmeye önemli katkılarda bulunuyor. Inline diziler sayesinde performans artışı elde ederken, interceptors ile daha yenilikçi kod yapıları oluşturmak mümkün oluyor. Ayrıca nameof ile daha güvenli ve esnek bir kodlama deneyimi yaşanıyor. Tüm bu özellikler, gelecekteki performans iyileştirmelerine zemin hazırlıyor ve geliştiricilere daha etkili araçlar sunuyor.
- Bu yenilikçi özelliklerle C# 12 ve .NET 8, yazılım dünyasında yeni bir döneme işaret ediyor. Daha hızlı ve verimli uygulamalar geliştirmek için Visual Studio önizleme sürümünü veya .NET SDK'nın en son sürümünü indirebilir ve projenizin dil sürümünü önizlemeye ayarlayabilirsiniz.
- Daha fazla bilgi için Microsoft Learn'daki "C# 12'de Neler Yeni" sayfasına ve Roslyn Özellik Durumu sayfasına göz atabilirsiniz. Siz de bu yenilikçi özellikleri deneyerek yazılım geliştirme deneyiminizi geliştirebilirsiniz.

Kaynak

Öncesi ve Sonrası

1.Primary Collection C# 12 Den Önce

```
public class UserService:IUserService
   {
   private readonly IUserRepository _userRepository;
   private readonly ILogger _logger;
public UserService(IUserRepository userRepository,
    ILogger logger;
{
  _userRepository=userRepository;
  _logger=logger;
}
   }
```

Öncesi ve Sonrası

1.Primary Collection C# 12 Sonrası

```
public class UserService(IUserRepository
   userRepository,ILoger logger):IUserService
{
private readonly IUserRepository
   _userRepository=userRepository;
private readonly ILogger _logger=logger;
}
```

Öncesi ve Sonrası

2.Collection Expressions C# 12 Den Önce

```
List<User> users=new List<User>();
// veya
var users=new List<User>();
// veya
List<User> user=new();
```

Öncesi ve Sonrası

2.Collection Expressions C# 12 Den Sonra

```
List<User> users=
[
new User {UserName="user1"},
new User {UserName="user2"}
];
```

Öncesi ve Sonrası

3.Default Lambda Paramaters C# 12 Den Önce

```
var getFullName=(string firstName,string
   lastName)=>string.Join("",firstName,lastNam
   e);
var fullName1=getFullName("Ali Can Yücel","");
var fullName2=getFullName("Yücel","Ali  Can");
```

Öncesi ve Sonrası

3.Default Lambda Parameters C# 12 Den Sonra

```
var getFullName=(string firstName,string
   lastName="")=>string.Join("",firstName,lastN
   ame);
var fullName1=getFullName("Yücel");
```

Öncesi ve Sonrası

4.Alias Any Type C# 12 Den Önce

```
public Person Get()
{
using
Person=System.ValueTuple<string,string>;
}
```

Öncesi ve Sonrası

4.Alias Any Type C# 12 Den Sonra

```
public  Person Get()
  {
   return ("Ali Can Yücel");
  }
public List<Person> GetAll()
{
return
[
  {"Ali Can Yücel"},
  {"Sefer Algan"},
  {"Ozan İz"},
  {"Uğur Kara"},
  {"Yağmur Mert"}
];
}
```

Öncesi ve Sonrası

5.Experimental Attribute C# 12 Den Önce

```
[Experimental("Feature07")]
public void DeleteAllProducts()
{
_context.Database.ExecuteSqlRaw("Delete
   from Products");
}
```

Öncesi ve Sonrası

5.Experimental Attribute C# 12 Den Sonra

```
#pragma warning disable Feature07
_context.Database.ExecuteSqlRaw("Delete
   from Products");
#pragma warning restore Feature07
```

Öncesi ve Sonrası

6.Inline Arrays C# 12 Den Önce

```
var users=new  MyArray<User>();
for(int i=0;i<5;i++)
{
users[i]=new User();
}
foreach(var user in users)
{
Console.WriteLine(user.FirstName + " " +
    user.LastName);
}
```

Öncesi ve Sonrası

6.Inline Arrays C# 12 Den Sonra

```
Using System.Runtime.CompilerServices;
[InlineArray(5)]
public struct MyArray<T>
{
private T _element
}
```

Öncesi ve Sonrası

7.Raw String Literals C# 12 Den Önce

```
var minPrice=100;
var maxPrice=1000;
var query=$""" Select * from Products Where
Price=>(minPrice) and Price<{maxPrice} Order
By Price Desc """;
```

Öncesi ve Sonrası

7.Raw String Literals C# 12 Den Sonra

```
var id=1;
var firstName="Ali Can";
var lastName="Yücel";
var json=$$"""
{
"id":"{{id}}",
"firstName":"{{firstName}}",
"lastName":"{{lastName}}"
}
""";
```