Bu dokümanda, C# uygulamalarında karmaşık veri yapılarının ve nesnelerin kalıcı olarak nasıl saklanacağı ve taşınacağı üzerine odaklanıyoruz. Basit metin dosyalarının ötesine geçerek, bir nesnenin tüm durumunu (state) koruyarak onu bir dosyaya yazma işlemi olan Serileştirme (Serialization) konsepti derinlemesine incelenmektedir. Bu çalışma, modern ve endüstri standardı haline gelmiş JSON, yapılandırılmış veri alışverişinin temel taşı olan XML ve performans odaklı ancak artık kullanımı tavsiye edilmeyen Binary formatları üzerinden nesnelerin nasıl serileştirileceğini ve dosyalardan okunarak tekrar nasıl hayata döndürüleceğini (deserialization) adım adım açıklamaktadır. Her bir format, kendi kütüphaneleri ve pratik kod örnekleriyle sunularak, C# geliştiricilerinin veri kalıcılığı ve iletişimi konusunda yetkinlik kazanması hedeflenmektedir.

NESNE SERİLEŞTİRME (SERİALİZATİON) İLE DOSYA İŞLEMLERİ

Serileştirme ve Tersine Serileştirme

Oğuzhan Karagüzel

İçindekiler

GİRİŞ

Önceki derslerimizde, File.WriteAllText() gibi metotlarla dosyalara basit metinler yazdık. Peki, bir öğrencinin ID'si, adı, soyadı ve aldığı derslerin listesini içeren karmaşık bir Ogrenci nesnesini tek bir seferde nasıl dosyaya kaydedebiliriz? Bu bilgileri manuel olarak bir string'e çevirip daha sonra bu string'i tekrar ayrıştırmaya çalışmak hem zahmetli hem de hataya çok açık bir yöntemdir.

İşte bu noktada **Serileştirme (Serialization)** devreye girer. Serileştirme, bir nesnenin o anki durumunu (içerdiği verileri) bir bayt dizisine veya bir metin formatına (JSON, XML vb.) dönüştürerek saklanabilir veya ağ üzerinden transfer edilebilir hale getirme işlemidir. Bu doküman, C#'ta bu işlemin nasıl yapıldığını üç temel format üzerinden, sıfırdan başlayarak ve her adımı açıklayarak öğretecektir.

SERİLEŞTİRME (SERIALIZATION) NEDİR?

En temel tanımıyla **Serileştirme**, bir nesnenin canlı, bellek üzerindeki halini, onu daha sonra yeniden oluşturabileceğimiz (deserialization) donuk, kalıcı bir formata dönüştürme sürecidir.

- **Canlı Nesne (Bellekte):** Ogrenci ogr = new Ogrenci(); koduyla oluşturduğumuz, metotları ve özellikleri olan nesne.
- **Donuk/Serileştirilmiş Format (Disk veya Ağda):** Bu nesnenin DNA'sının bir kopyası. Sadece veriyi içerir, davranışları (metotları) değil. Bu format JSON, XML veya ikilik (binary) bir veri akışı olabilir.

Sürecin İki Yönü Vardır:

1. **Serialization (Serileştirme):** Nesne → Dosya Formatı (JSON, XML, Binary)
2. **Deserialization (Deserileştirme):** Dosya Formatı → Nesne

Neden İhtiyaç Duyarız?

- **Veri Kalıcılığı (Persistence):** Program kapandığında nesnenin durumunu bir dosyaya kaydedip, program yeniden açıldığında bu dosyadan okuyarak nesneyi kaldığı yerden hayata döndürmek.
- **Veri İletişimi:** Bir web API'sine istek gönderirken veya bir API'den veri alırken, nesneler JSON veya XML formatında serileştirilerek ağ üzerinden gönderilir.
- **Önbellekleme (Caching):** Sıkça erişilen ve oluşturulması maliyetli olan nesneleri serileştirilmiş formatta bellekte veya diskte tutarak performansı artırmak.

C#'TA SERİLEŞTİRME FORMATLARI VE KÜTÜPHANELERİ

C#'ta bu işlem için kullanılan üç ana format ve ilgili kütüphaneler şunlardır:

- **JSON (JavaScript Object Notation): (İleride, Web programlama sıkça karşınıza çıkacak)**

    - **Özellikleri:** İnsan tarafından kolayca okunabilen, hafif, metin tabanlı bir formattır. anahtar:değer çiftlerinden oluşur. Modern web uygulamalarının ve API'lerin standart veri alışveriş formatıdır.
    - **Kütüphane:** System.Text.Json - .NET Core 3.0 ile birlikte gelen, Microsoft'un modern, yüksek performanslı ve standart JSON kütüphanesidir. (Daha önce popüler olan Newtonsoft.Json kütüphanesinin yerini almaktadır.)

- **XML (eXtensible Markup Language):**
    - **Özellikleri:** İnsan tarafından okunabilen, etiket (<tag>) tabanlı, hiyerarşik ve yapısal bir formattır. JSON'a göre daha ayrıntılı (verbose) ve dolayısıyla daha büyük dosya boyutlarına sahiptir. Özellikle kurumsal uygulamalarda ve eski sistemlerle entegrasyonda hala yaygındır.
    - **Kütüphane:** System.Xml.Serialization - .NET'in standart XML serileştirme kütüphanesidir.
- **Binary (İkilik):**
    - **Özellikleri:** İnsan tarafından okunamaz. Veriyi doğrudan bayt olarak sakladığı için en kompakt (en küçük dosya boyutu) ve en performanslı formatlardan biridir.
    - **Kütüphane:** System.Runtime.Serialization.Formatters.Binary

ÇOK ÖNEMLİ UYARI: BinaryFormatter'dan Kaçının!
BinaryFormatter, ciddi güvenlik açıkları nedeniyle **obsolete (kullanımdan kaldırılmış)** olarak işaretlenmiştir ve .NET 5 ve sonrası sürümlerde kullanımı varsayılan olarak engellenmiştir. Güvenilmeyen bir kaynaktan gelen binary veriyi deserileştirmek, saldırganın sunucunuzda rastgele kod çalıştırmasına olanak tanıyabilir. Bu dokümanda **yalnızca eğitim ve eski sistemleri anlama amacıyla** gösterilmiştir. **Yeni projelerde kesinlikle KULLANMAYIN!** Performans ve kompaktlık gerekiyorsa protobuf-net veya MessagePack gibi modern binary serileştirme kütüphanelerini araştırın.

ÖRNEK Ogrenci SINIFININ TANIMLANMASI

Tüm örneklerimizde aşağıdaki Ogrenci sınıfını kullanacağız. Serileştirme kütüphanelerinin düzgün çalışabilmesi için özelliklerin (property) public olması ve hem get hem de set erişimcilerine sahip olması (veya parametresiz bir constructor'ının olması) genellikle gereklidir.

Bu aşamada Nesne tabanlı programlamaya hala daha geçmedik. Ancak bu konuyu öğrenmek için oop bilmeye gerek yoktur. Structları kullanarak hala daha basit programlar oluşturabilir. Şu ana kadar öğrendikleriniz ile çalışan düzgün bir program yazabilirsiniz. Bundan dolayı şu anlık OOP'nin konusu olan işlemleri anlamasanız bile uygulayın ve ileride anlayacağınızı bilerek eğitiminize devam edin.

```
public class Ogrenci
{
    public int OgrenciNo { get; set; }
    public string Ad { get; set; }
```

```
    public string Soyad { get; set; }
    public List<string> Dersler { get; set; }
}
```

JSON SERİLEŞTİRME VE DESERİLEŞTİRME (System.Text.Json)

Modern ve en çok tavsiye edilen yöntemdir.

ADIM 1: NESNEYİ JSON STRING'İNE SERİLEŞTİRME

JsonSerializer.Serialize() metodu, verilen nesneyi bir JSON string'ine dönüştürür.

```
using System.Text.Json;

// 1. Örnek bir Ogrenci nesnesi oluşturalım.
Ogrenci ogr1 = new Ogrenci
{
    OgrenciNo = 101,
    Ad = "Ahmet",
    Soyad = "Yılmaz",
    Dersler = new List<string> { "Matematik", "Fizik", "Tarih" }
};

// 2. JsonSerializerOptions ile daha okunaklı bir çıktı alalım.
JsonSerializerOptions options = new JsonSerializerOptions
{
    WriteIndented = true // JSON çıktısını girintili ve düzenli formatta
yazar.
};

// 3. Nesneyi serileştir.
string jsonString = JsonSerializer.Serialize(ogr1, options);

Console.WriteLine("--- Oluşturulan JSON String ---");
```

```
Console.WriteLine(jsonString);
```

ADIM 2: JSON STRING'İNİ DOSYAYA YAZMA

File.WriteAllText() metodu ile oluşturulan bu string'i dosyaya yazabiliriz.

```
string dosyaYolu = @"C:\CSharpDersleri\ogrenci.json";

File.WriteAllText(dosyaYolu, jsonString);

Console.WriteLine($"\nJSON verisi başarıyla '{dosyaYolu}' dosyasına
yazıldı.");
```

ogrenci.json Dosyasının İçeriği:

```
{
  "OgrenciNo": 101,
  "Ad": "Ahmet",
  "Soyad": "Yılmaz",
  "Dersler": [
    "Matematik",
    "Fizik",
    "Tarih"
  ]
}
```

ADIM 3: DOSYADAN OKUMA VE DESERİLEŞTİRME

Şimdi bu dosyayı okuyup, içeriğini tekrar bir Ogrenci nesnesine dönüştürelim.

```
// 1. Dosyadaki JSON metnini oku.
string okunanJsonString = File.ReadAllText(dosyaYolu);


// 2. JSON metnini deserialize ederek yeni bir Ogrenci nesnesi oluştur.
//    <Ogrenci> ile hangi tipe dönüştürüleceğini belirtiyoruz.
Ogrenci okunanOgrenci =
JsonSerializer.Deserialize<Ogrenci>(okunanJsonString);


Console.WriteLine("\n--- Dosyadan Okunan ve Deserileştirilen Öğrenci
Bilgileri ---");
Console.WriteLine($"No: {okunanOgrenci.OgrenciNo}");
Console.WriteLine($"Ad Soyad: {okunanOgrenci.Ad} {okunanOgrenci.Soyad}");
```

```
Console.WriteLine($"İlk Ders: {okunanOgrenci.Dersler[0]}");
```

XML SERİLEŞTİRME VE DESERİLEŞTİRME (System.Xml.Serialization)

XML serileştirme, genellikle bir Stream (veri akışı) üzerinde çalışır.

ADIM 1 & 2: NESNEYİ SERİLEŞTİRME VE DOSYAYA YAZMA

Bu iki adım genellikle birlikte yapılır. FileStream kullanarak bir dosya akışı açar ve serileştiriciyi bu akışa yazması için yönlendiririz.

```
using System.Xml.Serialization;


Ogrenci ogr1 = new Ogrenci { /* ... aynı öğrenci verisi ... */ };


// 1. XmlSerializer'ı, hangi tipin serileştirileceğini belirterek oluştur.
XmlSerializer serializer = new XmlSerializer(typeof(Ogrenci));


string dosyaYolu = @"C:\CSharpDersleri\ogrenci.xml";


// 2. Bir dosya akışı (stream) oluştur. 'using' bloğu, işlem bitince akışın
güvenli
//    bir şekilde kapatılmasını garanti eder.
using (FileStream stream = new FileStream(dosyaYolu, FileMode.Create))
{
    // 3. Nesneyi akışa serileştir.
    serializer.Serialize(stream, ogr1);
}
Console.WriteLine($"XML verisi başarıyla '{dosyaYolu}' dosyasına
yazıldı.");
```

ogrenci.xml Dosyasının İçeriği:

```
<?xml version="1.0"?>
<Ogrenci xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <OgrenciNo>101</OgrenciNo>
```

```
  <Ad>Ahmet</Ad>
  <Soyad>Yılmaz</Soyad>
  <Dersler>
    <string>Matematik</string>
    <string>Fizik</string>
    <string>Tarih</string>
  </Dersler>
</Ogrenci>
```

ADIM 3 & 4: DOSYADAN OKUMA VE DESERİLEŞTİRME

```
Ogrenci okunanOgrenci = null;
using (FileStream stream = new FileStream(dosyaYolu, FileMode.Open))
{
    // Deserialize metodu bir 'object' döndürdüğü için,
    // onu kendi tipimize cast etmemiz (dönüştürmemiz) gerekir.
    okunanOgrenci = (Ogrenci)serializer.Deserialize(stream);
}


Console.WriteLine("\n--- XML'den Okunan ve Deserileştirilen Öğrenci
Bilgileri ---");
// ... öğrenci bilgilerini yazdırma kodu ...
```

BINARY SERİLEŞTİRME VE DESERİLEŞTİRME (BinaryFormatter)

Tekrar Uyarı: Bu yöntem güvensizdir ve yeni projelerde kullanılmamalıdır!
BinaryFormatter'ı kullanabilmek için, serileştirilecek sınıfı [Serializable] attribute'u ile işaretlememiz gerekir.

```
[Serializable] // Bu işaretleyici olmadan BinaryFormatter hata verir.
public class Ogrenci
{
    // ... sınıfın içeriği aynı ...
}
```

ADIM 1 & 2: NESNEYİ SERİLEŞTİRME VE DOSYAYA YAZMA

XML'e benzer şekilde Stream kullanırız.

```
// using System.Runtime.Serialization.Formatters.Binary;

// Proje dosyanıza
<EnableUnsafeBinaryFormatterSerialization>true</EnableUnsafeBinaryFormatter
Serialization> eklemeniz gerekebilir.

Ogrenci ogr1 = new Ogrenci { /* ... aynı öğrenci verisi ... */ };

BinaryFormatter formatter = new BinaryFormatter();

string dosyaYolu = @"C:\CSharpDersleri\ogrenci.dat"; // Binary dosyalar
için .dat, .bin gibi uzantılar kullanılır.


using (FileStream stream = new FileStream(dosyaYolu, FileMode.Create))

{

    formatter.Serialize(stream, ogr1);

}

Console.WriteLine($"Binary veri başarıyla '{dosyaYolu}' dosyasına
yazıldı.");
```

ogrenci.dat Dosyasının İçeriği:
Bu dosya bir metin düzenleyici ile açıldığında anlamsız karakterler ve semboller yığını olarak görünür çünkü insan tarafından okunabilir değildir.

ADIM 3 & 4: DOSYADAN OKUMA VE DESERİLEŞTİRME

```
Ogrenci okunanOgrenci = null;

using (FileStream stream = new FileStream(dosyaYolu, FileMode.Open))

{

    okunanOgrenci = (Ogrenci)formatter.Deserialize(stream);

}


Console.WriteLine("\n--- Binary'den Okunan ve Deserileştirilen Öğrenci
Bilgileri ---");

// ... öğrenci bilgilerini yazdırma kodu ...
```

FORMATLARIN KARŞILAŞTIRMASI VE SEÇİM KRİTERLERİ

Kriter	JSON (System.Text.Json)	XML (System.Xml.Serialization)	Binary (BinaryFormatter) - *TAVSİYE EDİLMEZ*
Okunabilirlik	**Çok Yüksek.** İnsan tarafından kolayca okunur/yazılır.	**Yüksek.** İnsan tarafından okunabilir ama daha karmaşık.	**Yok.** Makine tarafından okunur.
Dosya Boyutu	**Düşük.** Oldukça kompakttır.	**Yüksek.** Etiketler nedeniyle dosya boyutu büyüktür.	**En Düşük.** En kompakt formattır.
Performans	**Çok Yüksek.** Modern ve hızlıdır.	**Orta.** Ayrıştırması (parsing) JSON'a göre yavaştır.	**Çok Yüksek** (tarihsel olarak).
Kullanım Alanı	Web API'leri, konfigürasyon dosyaları, mobil uygulamalar.	Kurumsal sistemler, SOAP web servisleri, eski standartlar.	Yüksek performans gerektiren kapalı sistemler.
Esneklik / Standart	**Endüstri Standardı.** Dil bağımsız, çok yaygın.	Geniş bir standart, şema (XSD) desteği var.	.NET'e özgü, platformlar arası uyumu zayıf.
Güvenlik	**Güvenli.**	**Güvenli.**	**GÜVENSİZ!** Kullanımı risklidir.

SONUÇ VE TAVSİYELER

Serileştirme, nesnelerimizi kalıcı hale getirerek ve sistemler arasında taşıyarak uygulamalarımıza güç katan temel bir tekniktir.

- **Genel bir kural olarak, tüm yeni projelerinizde JSON serileştirmeyi (System.Text.Json) tercih edin.** Hem modern, hem performanslı, hem de endüstri standardıdır.
- Mevcut bir sistemle entegre olmanız gerekiyorsa veya verinin belirli bir şemaya uyması zorunlu ise **XML** hala geçerli bir seçenektir.
- **Binary** serileştirmeden, özellikle BinaryFormatter'dan, güvenlik riskleri nedeniyle **uzak durun**. Performans ve boyut kritik ise, Google Protocol Buffers veya MessagePack gibi modern, platformlar arası ve güvenli binary serileştirme kütüphanelerini öğrenmeyi düşünün.