OOP ATÖLYESİ-1 ÇÖZÜMLERİ

1: Akıllı Ev Cihazı (Smart Home Device)

Bu çözüm, `SmartLamp` sınıfını kapsülleme ilkelerine uygun olarak oluşturur. Private alanlar `_brightness` ve `_isOn`, public property ve metotlar aracılığıyla kontrollü bir şekilde yönetilir. `Brightness` property'sinin `set` bloğu, gelen veriyi doğrular.

```
using System;

// Program.cs
public class SmartLamp
{
    // Private alanlar (fields)
    private int _brightness;
    private bool _isOn;

    // Public Property: Parlaklık değerini kontrollü bir şekilde ayarlar.
    public int Brightness
    {
        get { return _brightness; }
        set
        {
            if (value >= 0 && value <= 100)
            {
                _brightness = value;
            }
            else
            {
                Console.WriteLine("Hata: Parlaklık değeri 0-100 arasında
olmalıdır.");
            }
        }
    }

    // Lambayı açan metot
    public void TurnOn()
    {
        _isOn = true;
```

```
        Console.WriteLine("Lamba açıldı.");
    }

    // Lambayı kapatan metot
    public void TurnOff()
    {
        _isOn = false;
        Console.WriteLine("Lamba kapatıldı.");
    }

    // Lambanın mevcut durumunu gösteren metot
    public void ShowStatus()
    {
        if (_isOn)
        {
            Console.WriteLine($"Lamba açık, parlaklık: %{_brightness}");
        }
        else
        {
            Console.WriteLine("Lamba kapalı.");
        }
    }
}

public class Program
{
    public static void Main(string[] args)
    {
        // 1. Nesne oluşturma
        SmartLamp livingRoomLamp = new SmartLamp();
        livingRoomLamp.ShowStatus(); // Başlangıç durumu

        // 2. Lambayı açma ve parlaklık ayarlama
        livingRoomLamp.TurnOn();
        livingRoomLamp.Brightness = 75;
        livingRoomLamp.ShowStatus();

        // 3. Parlaklığı artırma
        livingRoomLamp.Brightness = 100;
        livingRoomLamp.ShowStatus();

        // 4. Hatalı değer ataması testi
        livingRoomLamp.Brightness = 150; // Hata mesajı vermeli
```

```
        livingRoomLamp.ShowStatus(); // Değerin değişmediğini kontrol et

        // 5. Lambayı kapatma
        livingRoomLamp.TurnOff();
        livingRoomLamp.ShowStatus();
    }
}
```

2: RPG Oyunu Karakteri (RPG Character)

Bu örnekte, `Player` sınıfı bir yapıcı metot (constructor) ile başlangıç değerlerine sahip olur. `TakeDamage` metodu canın 0'ın altına düşmesini engeller ve `LevelUp` metodu karakterin özelliklerini günceller.

```
using System;

// Program.cs
public class Player
{
    // Private alanlar
    private int _health;
    private int _attackPower;
    private int _level;

    // Yapıcı Metot (Constructor)
    public Player()
    {
        _health = 100;
        _attackPower = 10;
        _level = 1;
        Console.WriteLine("Yeni bir oyuncu oluşturuldu!");
        ShowStatus();
    }

    // Hasar alma metodu
    public void TakeDamage(int damage)
    {
        _health -= damage;
```

```
        if (_health <= 0)
        {
            _health = 0;
            Console.WriteLine("Oyuncu hayatını kaybetti.");
        }
        else
        {
            Console.WriteLine($"Oyuncu {damage} hasar aldı.");
        }
        ShowStatus();
    }

    // Seviye atlama metodu
    public void LevelUp()
    {
        _level++;
        _health = 100; // Canı full'le
        _attackPower += 5;
        Console.WriteLine("Tebrikler, seviye atladınız!");
        ShowStatus();
    }

    // Yardımcı durum gösterme metodu
    public void ShowStatus()
    {
        Console.WriteLine($"-- Seviye: {_level} | Can: {_health} |
Saldırı Gücü: {_attackPower} --");
    }
}

public class Program
{
    public static void Main(string[] args)
    {
        // 1. Oyuncu oluşturma
        Player player1 = new Player();

        // 2. Hasar alma senaryoları
        player1.TakeDamage(30);
        player1.TakeDamage(50);

        // 3. Seviye atlama
        Console.WriteLine("\nOyuncu seviye atlıyor...");
```

```
        player1.LevelUp();

        // 4. Seviye atladıktan sonra hasar alma
        player1.TakeDamage(40);

        // 5. Oyuncuyu yenilgiye uğratma
        Console.WriteLine("\nOyuncu büyük bir darbe alıyor...");
        player1.TakeDamage(120);
    }
}
```

3: Farklı Medya Oynatıcı Sistemi (Media Player System)

Bu çözüm, kalıtım ve polimorfizmi gösterir. `MediaFile` temel sınıfından türeyen `MusicFile` ve `VideoFile` sınıfları, `Play` metodunu kendilerine özgü şekilde `override` eder. Ana programda hepsi bir `MediaFile` dizisinde toplanır ve tek bir döngü ile farklı davranışlar sergiler.

```
using System;
using System.Collections.Generic;

// Program.cs
// Temel Sınıf
public class MediaFile
{
    public string FileName { get; set; }
    public int Duration { get; set; }

    public virtual void Play()
    {
        Console.WriteLine("Medya dosyası oynatılıyor.");
    }
}

// Türetilmiş Müzik Dosyası Sınıfı
public class MusicFile : MediaFile
{
    public string Artist { get; set; }
```

```
    public override void Play()
    {
        Console.WriteLine($"{Artist} - {FileName} çalınıyor...");
    }
}

// Türetilmiş Video Dosyası Sınıfı
public class VideoFile : MediaFile
{
    public string Resolution { get; set; }

    public override void Play()
    {
        Console.WriteLine($"{FileName} adlı video {Resolution}
çözünürlükte oynatılıyor...");
    }
}

public class Program
{
    public static void Main(string[] args)
    {
        // Farklı tiplerde medya dosyaları oluşturma
        MediaFile[] mediaPlaylist = new MediaFile[]
        {
            new MusicFile { FileName = "Bohemian Rhapsody", Artist =
"Queen", Duration = 355 },
            new VideoFile { FileName = "Inception_trailer.mp4",
Resolution = "1080p", Duration = 150 },
            new MusicFile { FileName = "Smells Like Teen Spirit", Artist
= "Nirvana", Duration = 301 },
            new VideoFile { FileName = "TheMatrix_reloaded_scene.mkv",
Resolution = "4K", Duration = 245 },
            new MediaFile { FileName = "generic_sound.wav", Duration = 5
} // Temel sınıf nesnesi
        };

        // Polimorfizm: Tüm medya dosyalarını tek bir döngüde oynatma
        Console.WriteLine("--- Oynatma Listesi Başlatılıyor ---");
        foreach (var file in mediaPlaylist)
        {
            file.Play(); // Her nesne kendi Play() metodunu çağırır.
```

```
        }
    }
}
```

4: Ödeme Sistemi Tasarımı (Payment System Design)

Sanal metotlar ve kalıtım kullanılarak esnek bir ödeme altyapısı kurulmuştur. `PaymentMethod` temel sınıfındaki `MakePayment` metodu `virtual` olarak tanımlanmış, türetilmiş sınıflar bu metodu `override` ederek kendi ödeme mantıklarını uygulamıştır.

```
using System;

// Program.cs
// Temel Ödeme Yöntemi Sınıfı
public class PaymentMethod
{
    public decimal Amount { get; set; }

    public virtual bool MakePayment()
    {
        Console.WriteLine("Uyarı: Geçerli bir ödeme yöntemi seçilmedi.
Ödeme başarısız.");
        return false;
    }
}

// Kredi Kartı ile Ödeme Sınıfı
public class CreditCardPayment : PaymentMethod
{
    public string CardNumber { get; set; }

    public override bool MakePayment()
    {
        // Gerçekte burada kart doğrulama, banka ile iletişim vb. olurdu.
        Console.WriteLine($"{CardNumber} numaralı kart ile {Amount:C}
tutarında ödeme yapıldı.");
        return true;
    }
```

```
}

// Banka Transferi ile Ödeme Sınıfı
public class BankTransferPayment : PaymentMethod
{
    public string BankName { get; set; }

    public override bool MakePayment()
    {
        // Gerçekte burada banka API'si ile işlem onayı alınırdı.
        Console.WriteLine($"{BankName} bankası üzerinden {Amount:C}
tutarında transfer gerçekleştirildi.");
        return true;
    }
}


public class Program
{
    public static void Main(string[] args)
    {
        decimal cartTotal = 250.75m;
        Console.WriteLine($"Sepet tutarınız: {cartTotal:C}\n");

        // 1. Kredi Kartı ile Ödeme Denemesi
        CreditCardPayment ccPayment = new CreditCardPayment
        {
            Amount = cartTotal,
            CardNumber = "1234-5678-9876-5432"
        };

        bool isCreditCardPaymentSuccessful = ccPayment.MakePayment();
        if (isCreditCardPaymentSuccessful)
        {
            Console.WriteLine("Kredi kartı ödemesi başarılı!\n");
        }
        else
        {
            Console.WriteLine("Kredi kartı ödemesi başarısız!\n");
        }

        // 2. Banka Transferi ile Ödeme Denemesi
        BankTransferPayment transferPayment = new BankTransferPayment
```

```
        {
            Amount = cartTotal,
            BankName = "Garanti Bankası"
        };

        bool isTransferPaymentSuccessful = transferPayment.MakePayment();
        if (isTransferPaymentSuccessful)
        {
            Console.WriteLine("Banka transferi başarılı!");
        }
        else
        {
            Console.WriteLine("Banka transferi başarısız!");
        }
    }
}
```

5: Otomat Simülasyonu (Vending Machine Simulation)

Bu çözümde `Product` ve `VendingMachine` sınıfları arasındaki ilişki gösterilmektedir. `VendingMachine`, içinde `Product` nesnelerinden oluşan bir liste barındırır ve bakiye ile stok yönetimini kapsülleme ilkelerine uygun olarak yapar.

```
using System;
using System.Collections.Generic;
using System.Linq;

// Program.cs
public class Product
{
    public string Name { get; set; }
    public decimal Price { get; set; }
    public int Stock { get; set; }
}

public class VendingMachine
{
    private List<Product> _products;
```

```
    private decimal _currentBalance;

    public VendingMachine()
    {
        _products = new List<Product>();
        _currentBalance = 0;
    }

    // Makineye yeni ürün ekler
    public void AddProduct(Product product)
    {
        _products.Add(product);
        Console.WriteLine($"{product.Name} ürünü makineye eklendi.");
    }

    // Kullanıcının para atmasını sağlar
    public void InsertCoin(decimal amount)
    {
        _currentBalance += amount;
        Console.WriteLine($"{amount:C} para atıldı. Mevcut Bakiye:
{_currentBalance:C}");
    }

    // Ürün satın alma işlemi
    public void PurchaseProduct(string productName)
    {
        Product productToBuy = _products.FirstOrDefault(p =>
p.Name.Equals(productName, StringComparison.OrdinalIgnoreCase));

        if (productToBuy == null)
        {
            Console.WriteLine("Hata: Böyle bir ürün bulunamadı.");
            return;
        }

        if (productToBuy.Stock <= 0)
        {
            Console.WriteLine($"Hata: {productToBuy.Name} ürününün stoğu
tükenmiştir.");
            return;
        }

        if (_currentBalance < productToBuy.Price)
```

```
        {
            Console.WriteLine($"Hata: Yetersiz bakiye. Ürün fiyatı:
{productToBuy.Price:C}, Bakiyeniz: {_currentBalance:C}");
            return;
        }

        // Satın alma başarılı
        productToBuy.Stock--;
        decimal change = _currentBalance - productToBuy.Price;
        _currentBalance = 0; // Bakiye sıfırlanır (para üstü verildikten
sonra)

        Console.WriteLine($"{productToBuy.Name} satın alındı. Afiyet
olsun! Para üstünüz: {change:C}");
    }
}

public class Program
{
    public static void Main(string[] args)
    {
        // 1. Otomat ve ürünleri oluşturma
        VendingMachine machine = new VendingMachine();
        machine.AddProduct(new Product { Name = "Su", Price = 5.0m, Stock
= 10 });
        machine.AddProduct(new Product { Name = "Kola", Price = 15.0m,
Stock = 5 });
        machine.AddProduct(new Product { Name = "Bisküvi", Price = 8.5m,
Stock = 0 }); // Stoğu bitmiş ürün
        Console.WriteLine("\n----------------------------\n");

        // 2. Başarılı senaryo
        Console.WriteLine("--- Başarılı Satın Alma Senaryosu ---");
        machine.InsertCoin(10m);
        machine.InsertCoin(10m);
        machine.PurchaseProduct("Kola");
        Console.WriteLine("\n----------------------------\n");

        // 3. Yetersiz bakiye senaryosu
        Console.WriteLine("--- Yetersiz Bakiye Senaryosu ---");
        machine.InsertCoin(5m);
        machine.PurchaseProduct("Kola");
        Console.WriteLine("\n----------------------------\n");
```

```
        // 4. Stokta olmayan ürün senaryosu
        Console.WriteLine("--- Stokta Olmayan Ürün Senaryosu ---");
        machine.InsertCoin(20m);
        machine.PurchaseProduct("Bisküvi");

        // 5. Varolmayan ürün senaryosu
         Console.WriteLine("\n--- Varolmayan Ürün Senaryosu ---");
        machine.PurchaseProduct("Çikolata");
    }
}
```

6: Kargo Teslimat Stratejisi (Cargo Delivery Strategy)

`abstract` sınıf ve metotların kullanımını gösteren bir örnektir. `DeliveryVehicle` sınıfı, bir teslimat aracının sahip olması gereken temel özellikleri ve `CalculateDeliveryTime` gibi bir davranışı zorunlu kılar, ancak bu davranışın nasıl uygulanacağını (`abstract`) türetilmiş sınıflara bırakır.

```
using System;

// Program.cs
// Abstract Temel Sınıf
public abstract class DeliveryVehicle
{
    public double Speed { get; protected set; } // km/saat
    public double Capacity { get; protected set; } // kg

    // Soyut metot: Türetilen her sınıf bu metodu uygulamak zorundadır.
    public abstract double CalculateDeliveryTime(double distance);
}

// Kamyon Sınıfı
public class Truck : DeliveryVehicle
{
    public Truck()
    {
        Speed = 80; // km/saat
```

```
        Capacity = 5000; // kg
    }

    public override double CalculateDeliveryTime(double distance)
    {
        double travelTime = distance / Speed;
        int numberOfBreaks = (int)(distance / 50); // Her 50 km'de bir
mola
        double breakTime = numberOfBreaks * 0.25; // 15 dakika = 0.25
saat
        return travelTime + breakTime;
    }
}

// Drone Sınıfı
public class Drone : DeliveryVehicle
{
    public Drone()
    {
        Speed = 100; // km/saat
        Capacity = 5; // kg
    }

    public override double CalculateDeliveryTime(double distance)
    {
        // Gidiş-dönüş süresi (batarya değişimi için)
        double oneWayTime = distance / Speed;
        return oneWayTime * 2;
    }
}

public class Program
{
    public static void Main(string[] args)
    {
        double deliveryDistance = 160; // km

        DeliveryVehicle truck = new Truck();
        DeliveryVehicle drone = new Drone();

        double truckDeliveryTime =
truck.CalculateDeliveryTime(deliveryDistance);
```

```
        double droneDeliveryTime =
drone.CalculateDeliveryTime(deliveryDistance);

        Console.WriteLine($"{deliveryDistance} km mesafe için teslimat
süreleri:");
        Console.WriteLine($"Kamyon: {truckDeliveryTime:F2} saat");
        Console.WriteLine($"Drone: {droneDeliveryTime:F2} saat");

        if (truckDeliveryTime < droneDeliveryTime)
        {
            Console.WriteLine("\nKamyon daha hızlı teslimat
yapacaktır.");
        }
        else
        {
            Console.WriteLine("\nDrone daha hızlı teslimat yapacaktır.");
        }
    }
}
```

7: Metin Editörü Eklenti Sistemi (Text Editor Plugin System)

Bu örnek, `interface` kullanarak birbirinden tamamen bağımsız ve modüler bileşenler (eklentiler) oluşturmayı gösterir. `TextEditor`, `IPlugin` arayüzünü uygulayan herhangi bir sınıfı kabul edebilir. Bu, sisteme yeni özellikler eklemeyi çok kolaylaştırır.

```
using System;
using System.Collections.Generic;

// Program.cs
// Arayüz (Interface): Tüm eklentilerin uyması gereken sözleşme
public interface IPlugin
{
    string Execute(string text);
}

// Eklenti 1: Metni büyük harfe çevirir
public class UpperCasePlugin : IPlugin
```

```
{
    public string Execute(string text)
    {
        return text.ToUpper();
    }
}

// Eklenti 2: Kelime sayısını ekler
public class WordCountPlugin : IPlugin
{
    public string Execute(string text)
    {
        int wordCount = text.Split(new char[] { ' ', '\r', '\n' },
StringSplitOptions.RemoveEmptyEntries).Length;
        return $"{text} (Kelime Sayısı: {wordCount})";
    }
}

// Eklenti 3: Zaman damgası ekler
public class TimestampPlugin : IPlugin
{
    public string Execute(string text)
    {
        string timestamp = DateTime.Now.ToString("[dd.MM.yyyy HH:mm]");
        return $"{timestamp} {text}";
    }
}

// Eklentileri yöneten ana sınıf
public class TextEditor
{
    private readonly List<IPlugin> _plugins = new List<IPlugin>();

    public void AddPlugin(IPlugin plugin)
    {
        _plugins.Add(plugin);
        Console.WriteLine($"Eklenti eklendi: {plugin.GetType().Name}");
    }

    public string RunPlugins(string initialText)
    {
        string processedText = initialText;
        foreach (var plugin in _plugins)
```

```
        {
            processedText = plugin.Execute(processedText);
        }
        return processedText;
    }
}


public class Program
{
    public static void Main(string[] args)
    {
        TextEditor editor = new TextEditor();
        string myText = "nesne yönelimli programlama harika bir
paradigma.";

        Console.WriteLine("--- Senaryo 1: Zaman -> Büyük Harf -> Kelime
Sayısı ---\n");
        editor.AddPlugin(new TimestampPlugin());
        editor.AddPlugin(new UpperCasePlugin());
        editor.AddPlugin(new WordCountPlugin());

        string result1 = editor.RunPlugins(myText);
        Console.WriteLine($"\nOrijinal Metin: '{myText}'");
        Console.WriteLine($"İşlenmiş Metin: '{result1}'\n");


Console.WriteLine("\n-----------------------------------------------------
-\n");

        // Yeni bir editör veya mevcut olanın eklentileri temizlenebilir.
        TextEditor editor2 = new TextEditor();
        Console.WriteLine("--- Senaryo 2: Büyük Harf -> Kelime Sayısı ->
Zaman ---\n");
        editor2.AddPlugin(new UpperCasePlugin());
        editor2.AddPlugin(new WordCountPlugin());
        editor2.AddPlugin(new TimestampPlugin());

        string result2 = editor2.RunPlugins(myText);
        Console.WriteLine($"\nOrijinal Metin: '{myText}'");
        Console.WriteLine($"İşlenmiş Metin: '{result2}'\n");
        Console.WriteLine("Not: Eklenti sırasının sonucu nasıl
değiştirdiğine dikkat edin.");
```

```
    }
}
```

8: Dijital Kütüphane Sistemi (Digital Library System)

Bu örnek, `ToString()` metodunu `override` etmenin gücünü gösterir. Temel sınıf `LibraryItem` ve ondan türeyen sınıfların hepsi, kendilerini nasıl metin olarak temsil edeceklerini tanımlar. `Library` sınıfı, `ListAllItems` metodunda sadece `Console.WriteLine(item)` çağırarak, polimorfizm sayesinde her nesnenin kendi `ToString()` metodunu çalıştırmasını sağlar.

```
using System;
using System.Collections.Generic;

// Program.cs
// Temel Sınıf
public class LibraryItem
{
    public string Title { get; set; }
    public string Author { get; set; }
    public int PublicationYear { get; set; }

    public override string ToString()
    {
        return $"[Bilinmeyen Eser] {Title} - {Author}
({PublicationYear})";
    }
}

// Basılı Kitap Sınıfı
public class Book : LibraryItem
{
    public string ISBN { get; set; }

    public override string ToString()
    {
        return $"[Basılı Kitap] {Title} - {Author} ({PublicationYear}),
ISBN: {ISBN}";
```

```
    }
}

// E-Kitap Sınıfı
public class EBook : LibraryItem
{
    public double FileSizeMB { get; set; }

    public override string ToString()
    {
        return $"[E-Kitap] {Title} - {Author} ({PublicationYear}), Boyut:
{FileSizeMB} MB";
    }
}

// Sesli Kitap Sınıfı
public class AudioBook : LibraryItem
{
    public double DurationInHours { get; set; }

    public override string ToString()
    {
        return $"[Sesli Kitap] {Title} - {Author} ({PublicationYear}),
Süre: {DurationInHours} saat";
    }
}

// Kütüphane Yönetim Sınıfı
public class Library
{
    private List<LibraryItem> _items = new List<LibraryItem>();

    public void AddItem(LibraryItem item)
    {
        _items.Add(item);
    }

    public void ListAllItems()
    {
        Console.WriteLine("--- Kütüphane Kataloğu ---");
        if (_items.Count == 0)
        {
            Console.WriteLine("Kütüphanede eser bulunmamaktadır.");
```

```
            return;
        }

        foreach (var item in _items)
        {
            // Override edilen ToString() metodu burada otomatik olarak
çağrılır.
            Console.WriteLine(item);
        }
    }
}

public class Program
{
    public static void Main(string[] args)
    {
        Library myLibrary = new Library();

        // Kütüphaneye farklı türlerde eserler ekleme
        myLibrary.AddItem(new Book { Title = "Yüzüklerin Efendisi",
Author = "J.R.R. Tolkien", PublicationYear = 1954, ISBN =
"978-0618640157" });
        myLibrary.AddItem(new EBook { Title = "Dune", Author = "Frank
Herbert", PublicationYear = 1965, FileSizeMB = 5.2 });
        myLibrary.AddItem(new AudioBook { Title = "Sapiens", Author =
"Yuval Noah Harari", PublicationYear = 2011, DurationInHours = 15.5 });
        myLibrary.AddItem(new Book { Title = "1984", Author = "George
Orwell", PublicationYear = 1949, ISBN = "978-0451524935" });

        // Tüm eserleri listeleme
        myLibrary.ListAllItems();
    }
}
```

9: Abonelik Modeli Tasarımı (Subscription Model Design)

Bu çözümde, `Subscription` temel sınıfından türeyen `MonthlySubscription` ve `AnnualSubscription` sınıfları, kendi fiyatlarını ve yenilenme mantıklarını `constructor` ve `override` edilmiş `GetRenewalDate` metodu aracılığıyla belirler.

```
using System;

// Program.cs
// Temel Abonelik Sınıfı
public class Subscription
{
    public DateTime StartDate { get; set; }
    public decimal Price { get; protected set; }
    public bool IsActive { get; set; }

    public Subscription()
    {
        StartDate = DateTime.Now;
        IsActive = true;
    }

    public virtual DateTime GetRenewalDate()
    {
        // Temel sınıfta genel bir varsayım yok, bu yüzden başlangıç
tarihini döndürebiliriz.
        // Türetilmiş sınıflar bunu mantıklı bir şekilde ezecektir.
        return StartDate;
    }
}

// Aylık Abonelik Sınıfı
public class MonthlySubscription : Subscription
{
    public MonthlySubscription() : base()
    {
        Price = 100m;
    }

    public override DateTime GetRenewalDate()
    {
        return StartDate.AddMonths(1);
    }
}
```

```
// Yıllık Abonelik Sınıfı
public class AnnualSubscription : Subscription
{
    public AnnualSubscription() : base()
    {
        // Aylık 100 TL'den 12 ay 1200 TL, %20 indirimle 960 TL
        Price = (100m * 12) * 0.8m;
    }

    public override DateTime GetRenewalDate()
    {
        return StartDate.AddYears(1);
    }
}


public class Program
{
    public static void Main(string[] args)
    {
        // Aylık abonelik oluşturma
        MonthlySubscription monthly = new MonthlySubscription();
        Console.WriteLine("--- Aylık Abonelik ---");
        Console.WriteLine($"Başlangıç Tarihi:
{monthly.StartDate:dd.MM.yyyy}");
        Console.WriteLine($"Fiyat: {monthly.Price:C}");
        Console.WriteLine($"Yenilenme Tarihi:
{monthly.GetRenewalDate():dd.MM.yyyy}");
        Console.WriteLine();

        // Yıllık abonelik oluşturma
        AnnualSubscription annual = new AnnualSubscription();
        Console.WriteLine("--- Yıllık Abonelik ---");
        Console.WriteLine($"Başlangıç Tarihi:
{annual.StartDate:dd.MM.yyyy}");
        Console.WriteLine($"Fiyat: {annual.Price:C} (%20 indirimli)");
        Console.WriteLine($"Yenilenme Tarihi:
{annual.GetRenewalDate():dd.MM.yyyy}");
    }
}
```

10: Basit Takvim ve Randevu Sistemi (Calendar & Appointment System)

Bu örnekte, `static` bir `Calendar` sınıfı, tüm randevuları merkezi bir yerden yönetir. Bu sayede programın herhangi bir yerinden `Calendar.AddAppointment()` gibi metotlarla randevu eklenip sorgulanabilir. `DateTime` nesnesinin sadece tarih (`Date`) kısmının karşılaştırılması, aynı güne ait farklı saatlerdeki randevuların doğru bir şekilde bulunmasını sağlar.

```
using System;
using System.Collections.Generic;
using System.Linq;

// Program.cs
// Randevu verisini tutan sınıf
public class Appointment
{
    public string Title { get; set; }
    public string Description { get; set; }
    public DateTime AppointmentTime { get; set; }
}

// Randevuları merkezi olarak yöneten static sınıf
public static class Calendar
{
    private static List<Appointment> _allAppointments = new
List<Appointment>();

    // Yeni bir randevu ekler
    public static void AddAppointment(Appointment newAppointment)
    {
        _allAppointments.Add(newAppointment);
        Console.WriteLine($"Randevu eklendi: '{newAppointment.Title}' ->
{newAppointment.AppointmentTime}");
    }

    // Belirli bir tarihe ait randevuları döndürür
    public static List<Appointment> GetAppointmentsForDate(DateTime date)
    {
```

```
        // Sadece tarih (gün/ay/yıl) kısmını karşılaştır
        return _allAppointments
            .Where(apt => apt.AppointmentTime.Date == date.Date)
            .OrderBy(apt => apt.AppointmentTime) // Saate göre sırala
            .ToList();
    }

    // Belirli bir tarihin randevularını ekrana yazdırır
    public static void PrintAppointmentsForDate(DateTime date)
    {
        List<Appointment> todaysAppointments =
GetAppointmentsForDate(date);

        Console.WriteLine($"\n--- {date:dd MMMM yyyy} Günü Randevuları
---");

        if (todaysAppointments.Count == 0)
        {
            Console.WriteLine("Bugün için planlanmış bir randevu
bulunmamaktadır.");
        }
        else
        {
            foreach (var apt in todaysAppointments)
            {
                Console.WriteLine($"Saat: {apt.AppointmentTime:HH:mm} -
Başlık: {apt.Title} ({apt.Description})");
            }
        }
    }
}

public class Program
{
    public static void Main(string[] args)
    {
        // 1. Farklı tarihler ve saatler için randevular ekleme
        Calendar.AddAppointment(new Appointment { Title = "Dişçi
Kontrolü", Description = "Dr. Ahmet", AppointmentTime = new
DateTime(2025, 10, 28, 10, 30, 0) });
        Calendar.AddAppointment(new Appointment { Title = "Proje
Toplantısı", Description = "Ekip ile", AppointmentTime = new
DateTime(2025, 10, 28, 15, 00, 0) });
```

```
        Calendar.AddAppointment(new Appointment { Title = "Akşam Yemeği",
Description = "Aile", AppointmentTime = new DateTime(2025, 10, 29, 19,
00, 0) });
        Calendar.AddAppointment(new Appointment { Title = "Sabah Sporu",
Description = "Koşu", AppointmentTime = new DateTime(2025, 10, 28, 08,
00, 0) });
        Console.WriteLine("\n----------------------------------\n");

        // 2. Kullanıcıdan tarih isteme
        Console.Write("Lütfen randevularını görmek istediğiniz tarihi
girin (örn: 28.10.2025): ");
        try
        {
            string userInput = Console.ReadLine();
            DateTime selectedDate = DateTime.Parse(userInput);

            // 3. İstenen tarihteki randevuları yazdırma
            Calendar.PrintAppointmentsForDate(selectedDate);
        }
        catch (FormatException)
        {
            Console.WriteLine("Hatalı tarih formatı. Lütfen GG.AA.YYYY
formatında girin.");
        }

        // Örnek olarak randevu olmayan bir günü de sorgulayalım
        Calendar.PrintAppointmentsForDate(new DateTime(2025, 11, 1));
    }

}
```