1: Akıllı Ev Cihazı (Smart Home Device)

(Kapsülleme ve Sınıf Temelleri pratiği)

Senaryo: Akıllı bir ev sisteminin en basit parçası olan, uzaktan kontrol edilebilir bir lambanın modelini oluşturacaksınız.

İstekler:

- `SmartLamp` adında bir sınıf oluşturun.
- Bu sınıfın `_brightness` (0-100 arası bir `int`) ve `_isOn` (`bool`) adında `private` alanları olsun.
- Parlaklık değerini ayarlayan `public int Brightness` adında bir property yazın. `set` bloğunda, gelen değerin 0 ile 100 arasında olduğunu kontrol edin. Eğer bu aralıkta değilse, atama yapmayın ve "Hata: Parlaklık değeri 0-100 arasında olmalıdır." gibi bir uyarı mesajı verin.
- Lambayı açan `TurnOn()` ve kapatan `TurnOff()` adında iki `public` metot yazın. Bu metotlar `_isOn` alanını uygun şekilde değiştirmelidir.
- Lambanın mevcut durumunu ("Lamba açık, parlaklık: %75" veya "Lamba kapalı." gibi) ekrana yazdıran bir `ShowStatus()` metodu yazın.
- `Main` metodunda bu sınıftan bir nesne oluşturup tüm metot ve property'lerini (hatalı değer atama dahil) test edin.

2: RPG Oyunu Karakteri (RPG Character)

(Constructor ve Sınıf Davranışları pratiği)

Senaryo: Bir rol yapma oyunundaki temel bir oyuncu karakterinin özelliklerini ve hayatta kalma mekaniklerini tasarlayacaksınız.

İstekler:

- `Player` adında bir sınıf tasarlayın.
- `_health` (`int`), `_attackPower` (`int`), `_level` (`int`) adında `private` alanları olsun.
- Oyuncu ilk oluşturulduğunda (`new Player()`) canının 100, gücünün 10 ve seviyesinin 1 olmasını sağlayan bir **yapıcı metot (constructor)** yazın.
- `TakeDamage(int damage)` adında bir metot yazın. Bu metot canı azaltmalı ama canın 0'ın altına düşmesini engellemelidir. Can 0 veya altına düşerse, canı 0'a eşitleyip "Oyuncu hayatını kaybetti." mesajı versin.
- `LevelUp()` adında bir metot yazın. Seviyeyi 1 artırsın, canı 100'e tamamlasın (full'lesin) ve saldırı gücünü (`_attackPower`) 5 artırsın.
- `Main` metodunda bir oyuncu oluşturun, birkaç kez `TakeDamage` metodu ile hasar almasını sağlayın, sonra `LevelUp` ile seviye atlatın ve durumunu kontrol edin.

3: Farklı Medya Oynatıcı Sistemi (Media Player System)

(Kalıtım ve Polimorfizm pratiği)

Senaryo: Farklı türdeki medya dosyalarını (müzik, video) oynatabilen bir sistemin temelini atacaksınız.

İstekler:

- `MediaFile` adında bir **temel sınıf** oluşturun. `FileName` (`string`) ve `Duration` (`int`, saniye cinsinden) gibi ortak property'leri olsun. Bir de `virtual void Play()` metodu olsun ve "Medya dosyası oynatılıyor." yazsın.
- Bu sınıftan türeyen `MusicFile` (`Artist` diye ek bir `string` property'si olsun) ve `VideoFile` (`Resolution` diye ek bir `string` property'si olsun, örn: "1080p") adında iki sınıf oluşturun.
- Her iki türetilmiş sınıfta da `Play()` metodunu **`override`** edin.
    - `MusicFile`: "`Artist` - `FileName` çalınıyor..." yazsın.
    - `VideoFile`: "`FileName` adlı video `Resolution` çözünürlükte oynatılıyor..." yazsın.
- `Main` metodunda farklı tiplerde medya dosyalarını (birkaç müzik, birkaç video) bir `MediaFile[]` dizisine atın ve bir `foreach` döngüsüyle hepsinin `Play()` metodunu çağırarak polimorfizmin gücünü gözlemleyin.

4: Ödeme Sistemi Tasarımı (Payment System Design)

(Kalıtım ve Sanal Metotlar pratiği)

Senaryo: Bir e-ticaret sitesinin farklı ödeme yöntemlerini destekleyebilecek esnek bir altyapısını kuracaksınız.

İstekler:

- `PaymentMethod` adında bir **temel sınıf** oluşturun. `Amount` (`decimal`) diye bir property'si olsun. `virtual bool MakePayment()` metodu olsun ve ödemenin temel olarak başarısız olduğunu varsayarak `false` döndürsün ve bir uyarı mesajı versin.
- Bu sınıftan `CreditCardPayment` (`CardNumber` özelliği olsun) ve `BankTransferPayment` (`BankName` özelliği olsun) sınıflarını türetin.
- Her iki sınıfta da `MakePayment()` metodunu **`override`** edin. İçinde, kendi yöntemine özgü ("`CardNumber` numaralı kart ile `Amount` tutarında ödeme yapıldı." gibi) başarılı bir mesaj yazdırıp `true` döndürsünler.
- `Main` metodunda bir sepet tutarı belirleyin. Bir `CreditCardPayment` nesnesi ve bir `BankTransferPayment` nesnesi oluşturup her ikisiyle de ödeme yapmayı deneyin ve dönen `bool` sonuca göre kullanıcıya "Ödeme başarılı!" veya "Ödeme başarısız!" mesajı gösterin.

5: Otomat Simülasyonu (Vending Machine Simulation)

(Sınıf İlişkileri ve Kapsülleme pratiği)

Senaryo: İçindeki ürünleri ve parayı yöneten bir otomat makinesi tasarlayacaksınız. Bu ödev, birden fazla sınıfın birbiriyle nasıl etkileşime girdiğini anlamak için harikadır.

İstekler:

- `Product` adında bir sınıf oluşturun. `Name` (`string`), `Price` (`decimal`) ve `Stock` (`int`) property'leri olsun.
- `VendingMachine` adında bir sınıf oluşturun.
  - İçinde `private List<Product> _products` ve `private decimal _currentBalance` (makineye atılan para) alanları olsun.
  - `AddProduct(Product product)`: Makineye yeni bir ürün ekler.
  - `InsertCoin(decimal amount)`: Kullanıcının makineye para atmasını sağlar, `_currentBalance`'ı artırır.
  - `PurchaseProduct(string productName)`:
    - İsmi verilen ürünü listede bulur.
    - Ürün yoksa veya stoğu bitmişse hata verir.
    - Kullanıcının bakiyesi (`_currentBalance`) ürünün fiyatından azsa hata verir.
    - Başarılı olursa, ürünün stoğunu 1 azaltır, ürün fiyatını `_currentBalance`'tan düşürür ve "Afiyet olsun! Para üstünüz: X TL" mesajını verir.
- `Main` metodunda bir otomat yaratın, içine birkaç farklı ürün ekleyin, para atın ve ürün satın almayı deneyin (başarılı ve başarısız senaryoları test edin).

6: Kargo Teslimat Stratejisi (Cargo Delivery Strategy)

(Abstract Sınıflar ve Kalıtım pratiği)

Senaryo: Bir kargo şirketinin farklı araçlarla (kamyon, drone) yaptığı teslimatların sürelerini hesaplayan bir sistem kuracaksınız.

İstekler:

- `DeliveryVehicle` adında bir **`abstract`** sınıf oluşturun.
    - `Speed` (`double`, km/s) ve `Capacity` (`double`, kg) gibi normal property'leri olsun.
    - `public abstract double CalculateDeliveryTime(double distance)` adında bir **soyut metodu** olsun.
- `Truck` ve `Drone` adında iki sınıfı `DeliveryVehicle` sınıfından türetin.
- Her iki sınıfın `constructor`'ı kendi `Speed` ve `Capacity` değerlerini ayarlasın.
- Her iki sınıfta da `CalculateDeliveryTime` metodunu **`override`** edin.
    - `Truck`: Teslimat süresini hesaplarken, her 50 km'de bir 15 dakikalık (0.25 saat) mola verdiğini hesaba katsın. (`süre = mesafe / hız + mola_sayisi * 0.25`).
    - `Drone`: Teslimat süresini hesaplarken, batarya değişimi için gidiş-dönüş süresini hesaba katsın. Yani süreyi 2 ile çarpsın. (`süre = (mesafe / hız) * 2`)
- `Main` metodunda belirli bir mesafe için hem kamyonun hem de drone'un teslimat süresini hesaplatıp karşılaştırın.

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

(Interface ve Bağımsızlık pratiği)

Senaryo: Bir metin editörüne, birbirinden bağımsız çalışabilen ve kolayca yenileri eklenebilen bir "eklenti" (plugin) altyapısı tasarlayacaksınız. Bu, modern yazılım mimarisinin temelini anlamak için mükemmel bir ödevdir.

İstekler:

- `IPlugin` adında bir **`interface`** oluşturun. İçinde `string Execute(string text)` adında bir metot imzası bulunsun.
- Bu arayüzü uygulayan üç farklı sınıf oluşturun:
  - `UpperCasePlugin`: Aldığı metnin tamamını büyük harfe çevirip döndürür.
  - `WordCountPlugin`: Aldığı metnin sonuna "(Kelime Sayısı: X)" bilgisini ekleyip döndürür.
  - `TimestampPlugin`: Aldığı metnin başına `[GG.AA.YYYY SS:DD]` formatında o anki zaman damgasını ekleyip döndürür.
- `TextEditor` adında bir sınıf oluşturun.
  - `private List<IPlugin> _plugins` listesi olsun.
  - `AddPlugin(IPlugin plugin)` metodu ile editöre yeni eklentiler eklenebilsin.
  - `RunPlugins(string initialText)` metodu, aldığı metni sırayla listedeki tüm eklentilerden geçirsin (bir eklentinin çıktısı, diğerinin girdisi olacak şekilde) ve son halini döndürsün.
- `Main` metodunda bir editör yaratın, ona istediğiniz sırada eklentileri ekleyin ve bir metin üzerinde çalıştırarak sonucu görün. Eklentilerin sırasını değiştirip sonucun nasıl değiştiğini gözlemleyin.

8: Dijital Kütüphane Sistemi (Digital Library System)

(Kalıtım, Polimorfizm ve `ToString()` Override pratiği)

Senaryo: Farklı formatlardaki kitapları (basılı, e-kitap, sesli kitap) yöneten bir kütüphane sistemi tasarlayacaksınız.

İstekler:

- `LibraryItem` adında bir temel sınıf oluşturun. `Title`, `Author`, `PublicationYear` property'leri olsun.
- Bu temel sınıftan `Book` (`ISBN` özelliği olsun), `EBook` (`FileSizeMB` özelliği olsun) ve `AudioBook` (`DurationInHours` özelliği olsun) sınıflarını türetin.
- Tüm bu sınıflarda (temel sınıf dahil) `ToString()` metodunu **`override`** edin. `ToString()` metodu, o nesneye ait tüm bilgileri (türü dahil) anlamlı bir metin olarak döndürmelidir.
    - Örnek `Book` çıktısı: `[Basılı Kitap] Yüzüklerin Efendisi - J.R.R. Tolkien (1954), ISBN: 12345`
    - Örnek `EBook` çıktısı: `[E-Kitap] Dune - Frank Herbert (1965), Boyut: 5.2 MB`
- `Library` adında bir sınıf oluşturun. İçinde `private List<LibraryItem> _items` listesi olsun ve kitap eklemek için `AddItem(LibraryItem item)` metodu bulunsun.
- `Library` sınıfına, tüm kitapların detaylarını listeleyen `ListAllItems()` adında bir metot ekleyin. Bu metot, `_items` listesindeki her bir eleman için `Console.WriteLine(item);` komutunu çağırmalıdır. (Override ettiğiniz `ToString` sayesinde bu sihirli bir şekilde çalışacaktır).
- `Main`'de bir kütüphane oluşturun, içine her türden birkaç kitap ekleyin ve tüm kitapları listeleyin.

9: Abonelik Modeli Tasarımı (Subscription Model Design)

(Kalıtım, Constructor ve Polimorfizm pratiği)

Senaryo: Bir online hizmetin farklı abonelik tiplerini (Aylık, Yıllık) ve bu aboneliklerin fiyatlandırma ve yenilenme tarihlerini yöneten bir sistem tasarlayacaksınız.

İstekler:

- `Subscription` adında bir temel sınıf oluşturun. `StartDate` (`DateTime`), `Price` (`decimal`) ve `IsActive` (`bool`) property'leri olsun.
- Bu sınıftan `MonthlySubscription` ve `AnnualSubscription` sınıflarını türetin.
- `MonthlySubscription` oluşturulurken fiyatı 100 TL olarak, `AnnualSubscription` oluşturulurken ise %20 indirimli olarak (12 ay için 960 TL) ayarlanmalıdır. Bu işlemi `constructor` içinde yapın.
- Temel sınıfa `public virtual DateTime GetRenewalDate()` adında bir metot ekleyin. Bu metot, `StartDate`'e göre bir sonraki yenilenme tarihini hesaplamalıdır.
    - `MonthlySubscription` için `override` edin: `StartDate`'e 1 ay eklesin.
    - `AnnualSubscription` için `override` edin: `StartDate`'e 1 yıl eklesin.
- `Main`'de hem aylık hem de yıllık bir abonelik oluşturun. Fiyatlarını ve bir sonraki yenilenme tarihlerini ekrana yazdırarak doğruluğunu kontrol edin.

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

(Static Sınıflar, Sınıf İlişkileri ve `DateTime` pratiği)

Senaryo: Belirli bir tarihe randevular ekleyip o tarihteki randevuları listeleyen basit bir konsol tabanlı takvim sistemi oluşturacaksınız.

İstekler:

- `Appointment` adında bir sınıf oluşturun. `Title` (`string`), `Description` (`string`) ve `AppointmentTime` (`DateTime`) property'leri olsun.
- `Calendar` adında **`static`** bir sınıf oluşturun. Bu sınıf, tüm randevuları merkezi olarak yönetecek.
    - `private static List<Appointment> _allAppointments` listesi olsun.
    - `public static void AddAppointment(Appointment newAppointment)`: Yeni bir randevuyu listeye ekler.
    - `public static List<Appointment> GetAppointmentsForDate(DateTime date)`: Verilen tarihe ait (saati dikkate almadan, sadece gün/ay/yıl olarak) tüm randevuları yeni bir liste olarak döndürür.
    - `public static void PrintAppointmentsForDate(DateTime date)`: Yukarıdaki metodu kullanarak belirli bir tarihin randevularını bulur ve formatlı bir şekilde ("Saat: SS:DD - Başlık: X") ekrana yazdırır. Eğer o gün randevu yoksa "Bugün için planlanmış bir randevu bulunmamaktadır." mesajı verir.
- `Main` metodunda, `Calendar` sınıfını kullanarak farklı tarihlere ve saatlere birkaç randevu ekleyin. Ardından kullanıcıdan bir tarih girmesini isteyin (`DateTime.Parse(Console.ReadLine())` kullanabilirsiniz) ve o tarihteki randevuları ekrana bastırın.