OTEL YÖNETİM SİSTEMİ

TASARIM RAPORU (DESIGN REPORT)

1. GİRİŞ

Bu doküman, Java programlama dili kullanılarak geliştirilen **Otel Yönetim Sistemi** için hazırlanmış tasarım raporudur. Raporun amacı, sistemin analiz aşamasında belirlenen gereksinimlerinin, uygulama öncesinde **kontrollü, sade ve sürdürülebilir** bir yapıya nasıl dönüştürüldüğünü açıklamaktır.

Tasarım sürecinde öncelik, sistemin karmaşık hale gelmesini önlemek ve her bileşenin açık bir sorumluluğa sahip olmasını sağlamaktır. Bu nedenle tasarım kararları alınırken; okunabilirlik, hata ayıklama kolaylığı ve değişikliklere karşı dayanıklılık temel ölçütler olarak ele alınmıştır.

Hazırlanan bu tasarım, geliştirilen ve test edilen uygulama ile birebir uyumludur ve gereksiz detaylardan kaçınılarak yalnızca sistemi anlamak ve sürdürmek için gerekli olan unsurlara odaklanılmıştır.

2.TASARIM HEDEFLERİ

2.1 Temel Tasarım Hedefleri

2.1.1 İş kurallarını kullanıcı arayüzünden ayırmak (Separation of Concerns)

- **Açıklama:** Tarih çakışma kontrolü (isDateAvailable), fiyat hesaplama (calculatePrice), oda uygunluk durumu ve Check-in/Check-out sırasındaki durum güncellemeleri gibi kritik mantık tamamen ReservationService sınıfında toplanmıştır. MenuController (UI katmanı) yalnızca kullanıcıdan tarih ve ID bilgilerini alır; kararı servis verir.
- **Gerekçe:** Arayüz katmanı ile iş mantığı katmanını ayırmak (SoC), yazılımın modülerliğini sağlar. İleride konsol yerine Web arayüzü yapılsa bile rezervasyon kuralları değişmez.
- **Avantaj:** Hata ayıklama kolaylaşır; hesaplama hatası varsa ReservationService'e, giriş hatası varsa MenuController'a bakılır.

- **Uygulama:** MenuController → ReservationService.createReservation() → FileManager akışı.

2.1.2 Dosya işlemlerini merkezi ve kontrollü bir yapı altında toplamak

- **Açıklama:** rooms.txt, customers.txt ve reservations.txt dosyalarına erişim, okuma, yazma ve ID üretme işlemleri sadece FileManager sınıfı içinde tanımlanmıştır.
- **Gerekçe:** Dosya yollarının ve okuma mantığının kodun içine dağılması, veri tutarlılığını bozar. Veri erişimini tek bir sınıftan yönetmek güvenlidir.
- **Avantaj:** Veri bütünlüğü sağlanır. Bir dosya formatı değişirse (örneğin CSV yapısı değişirse) sadece FileManager güncellenir.
- **Uygulama:** Servis katmanı dosyayı doğrudan açmaz, fileManager.readRooms() metodunu çağırır.

2.1.3 Kod tekrarını önlemek (DRY - Don't Repeat Yourself)

- **Açıklama:** Nesnelerin dosyaya yazılacak formatı (CSV) her sınıfın kendi içinde (toFileString) tanımlanmıştır. Ayrıca dosya güncelleme işlemleri için FileManager içinde genel güncelleme metotları (updateAllRooms) oluşturulmuştur.
- **Gerekçe:** Aynı formatlama kodunu veya dosya yazma bloğunu her yerde tekrar yazmak, bakım maliyetini artırır ve hata riskini doğurur.
- **Avantaj:** Kod daha temiz ve okunabilir hale gelir.
- **Uygulama:** Room, Customer ve Reservation sınıflarının hepsinde standart toFileString() metodu kullanılması.

2.1.4 Genişletilebilir bir mimari oluşturmak (Extensibility)

- **Açıklama:** Sistem; Varlık (Entity), Veri Erişimi (FileManager) ve Servis (Service) katmanlarına ayrılmıştır. Bu yapı sayesinde sisteme yeni özellikler (Örn: Oda temizlik takibi veya Fatura modülü) mevcut kodu bozmadan eklenebilir.
- **Gerekçe:** Yazılımın yaşam döngüsü boyunca yeni istekler gelecektir. Kodun değişime direnç göstermemesi gerekir (Open/Closed Principle).
- **Avantaj:** Yeni bir özellik eklemek, tüm sistemi yeniden yazmayı gerektirmez.
- **Uygulama:** ReservationService sınıfına, mevcut yapıyı bozmadan cancelReservation() gibi yeni metotlar eklenebilir.

2.1.5 Analiz raporunda tanımlanan tüm Gereksinimleri karşılamak

- **Açıklama:** Analiz raporundaki UC1 (Oda Ekle) işleminden UC8 (Check-out) işlemine kadar tüm senaryolar kodda karşılık bulmuştur. Özellikle tarih çakışma kontrolü (FR3.3) kodda isDateAvailable metoduyla kesin olarak sağlanmıştır.

- **Gerekçe:** Tasarımın başarısı, analiz edilen ihtiyaçları ne kadar doğru karşıladığı ile ölçülür.
- **Avantaj:** Müşteri ve oda yönetimi eksiksiz çalışır, sistem güven verir.
- **Uygulama:** Her menü seçeneği (MenuController), analizdeki bir Use Case'i tetikler.

Bu kriterler, sistemin fonksiyonel olmayan gereksinimlerini (Performans, Kullanılabilirlik vb.) karşılamak için belirlenmiştir.

2.2.1 Kullanılabilirlik (Usability)

- **Menü tabanlı sade CLI arayüz:** Grafik arayüz yerine, hataya yer bırakmayan adım adım ilerleyen konsol menüsü tercih edilmiştir.
- **Avantaj:** Eğitim seviyesi ne olursa olsun personel tarafından kolayca öğrenilebilir.
- **Açıklayıcı Hata Mesajları:** Kullanıcı geçmişe dönük tarih girdiğinde veya dolu odayı seçtiğinde, program çökmez; "Hata: Tarihler geçersiz" veya "Hata: Oda dolu" gibi net Türkçe mesajlar verir.
- **Avantaj:** Kullanıcı hatasını kendi düzeltebilir.

2.2.2 Bakım Kolaylığı (Maintainability)

- **Tek Sorumluluk Prensibi (SRP):** Her sınıfın sadece bir görevi vardır. Room sadece veri tutar, ReservationService sadece hesap yapar.
- **Avantaj:** Bir sınıfta yapılan değişiklik diğerlerini domino taşı gibi etkilemez.

2.2.3 Performans (Performance)

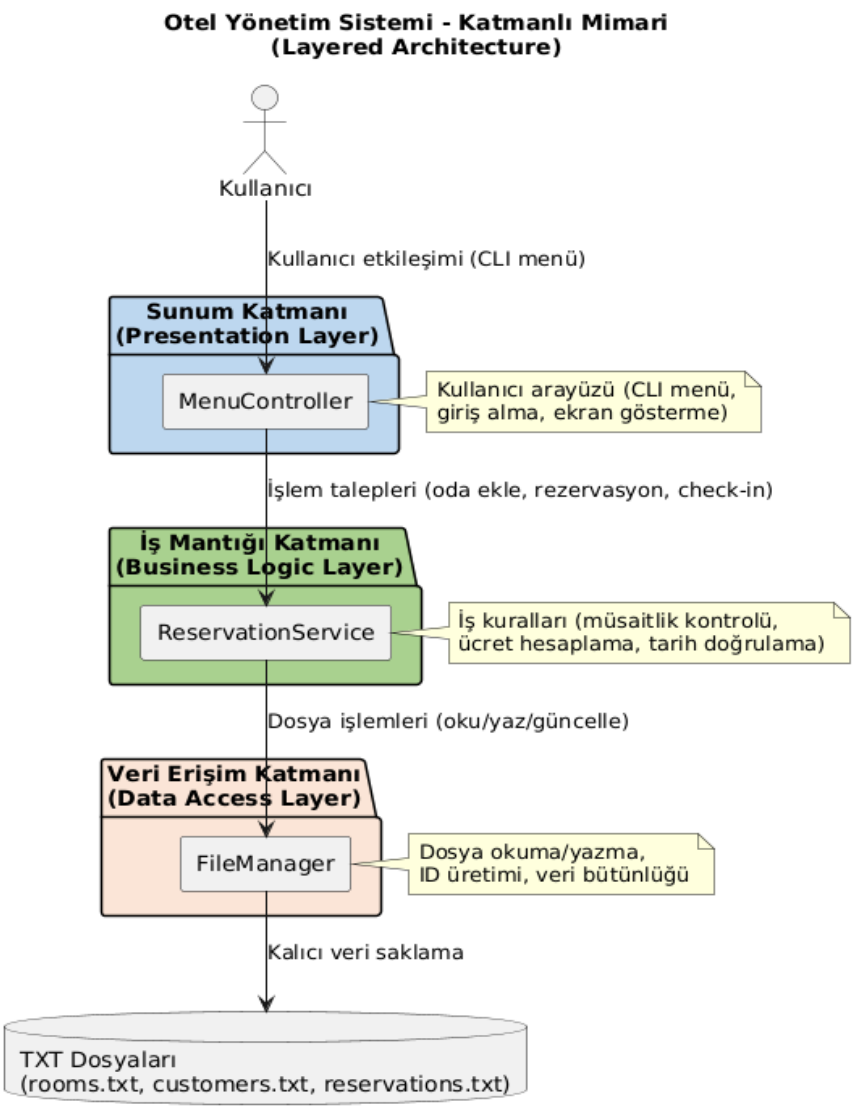
Bellek İçi İşlem: Dosyalar program açılışında bir kez okunur ve listelere (ArrayList) aktarılır. Arama ve listeleme işlemleri RAM üzerinden yapılır.

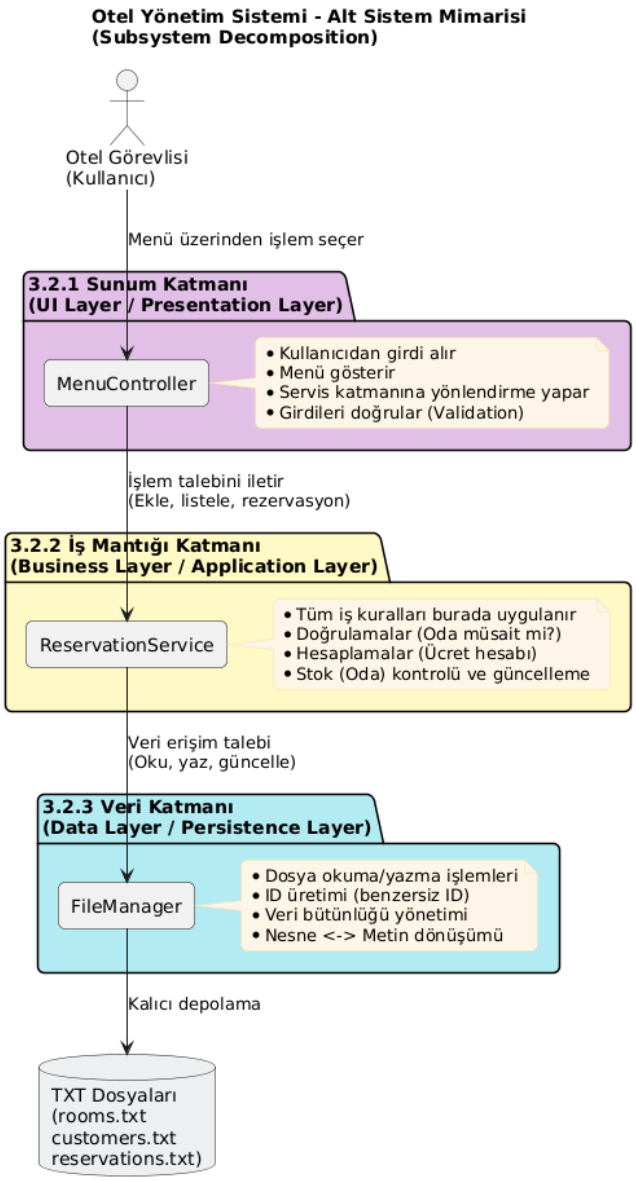
Avantaj: Disk okuma yavaşlığı yaşanmaz, işlemler (oda arama vb.) anlıktır.

2.3 Tasarımın Kısıtları ve Dezavantajları (Trade-offs)

Mevcut tasarım kararlarının getirdiği bazı kısıtlamalar ve dezavantajlar aşağıda belirtilmiştir:

- **Veri Bütünlüğü Riski (Concurrency):** Sistem dosya tabanlı olduğu için, aynı anda iki resepsiyonist işlem yapmaya çalışırsa (dosya kilitleme olmadığı için) son kaydedilen veri öncekini ezebilir.
- **Çözüm:** Şu an tek kullanıcılı tasarlanmıştır. İleride Veritabanı (SQL) geçişi gerektirir.
- **Büyük Veri Performansı:** Rezervasyon sayısı binlere ulaştığında, her güncellemede dosyanın tamamının (updateAllReservations) yeniden yazılması işlemi yavaşlatacaktır.
- **Çözüm:** Dosya yerine veritabanı kullanılarak sadece değişen satırın güncellenmesi sağlanabilir.
- **Veri Kaybı Riski:** Elektrik kesintisi veya ani kapanmada, RAM'deki veriler dosyaya yazılmamışsa son işlemler kaybolabilir.
- **Çözüm:** Kritik işlemlerden hemen sonra write metodunun çağrılması (mevcut kodda uygulanmıştır).


Otel Yönetim Sistemi - Katmanlı Mimari
(Layered Architecture)
Kullanıcı
Kullanıcı etkileşimi (CLI menü)
Sunum Katmanı
(Presentation Layer)
MenuController
Kullanıcı arayüzü (CLI menü,
giriş alma, ekran gösterme)
İşlem talepleri (oda ekle, rezervasyon, check-in)
İş Mantığı Katmanı
(Business Logic Layer)
ReservationService
İş kuralları (müsaitlik kontrolü,
ücret hesaplama, tarih doğrulama)
Dosya işlemleri (oku/yaz/güncelle)
Veri Erişim Katmanı
(Data Access Layer)
FileManager
Dosya okuma/yazma,
ID üretimi, veri bütünlüğü
Kalıcı veri saklama
TXT Dosyaları
(rooms.txt, customers.txt, reservations.txt)



3.SİSTEM MİMARİSİ

Sistem, katmanlı mimari (Layered Architecture) yaklaşımıyla üç ana alt sisteme ayrılmıştır. Bu ayrışım, sorumlulukların net bir şekilde bölünmesini, bakım kolaylığını, genişletilebilirliği ve test edilebilirliği sağlar. Her alt sistem, analiz raporunda tanımlanan fonksiyonel ve fonksiyonel olmayan gereksinimleri karşılayacak şekilde tasarlanmıştır.

3.1 Sunum Katmanı Alt Sistemi (User Interface Subsystem / Presentation Layer)

Amaç:

Kullanıcı (otel resepsiyonisti) ile sistem arasındaki tüm etkileşimi yönetir. Kullanıcı girdilerini alır, menüleri ve listeleri konsol üzerinde gösterir, işlem sonuçlarını kullanıcıya bildirir.

Ana Bileşen:

- **MenuController**

Sorumluluklar:

- Menü Yönetimi: Ana menüyü ve alt işlem menülerini (Oda Ekle, Rezervasyon Listele, Check-in vb.) konsol üzerinde gösterir.
- Girdi Alma: Kullanıcıdan sayısal seçim (Switch-Case yapısı için) ve metin girdileri (Ad, Tarih) alır.
- Girdi Doğrulama: Girdiler üzerinde temel doğrulamaları yapar (boş giriş, sayısal değer hatası için try-catch blokları, tarih formatı kontrolü).
- Yönlendirme: Kullanıcı seçimine göre ilgili iş mantığı yöntemlerini (ReservationService) veya listeleme için veri yöntemlerini (FileManager) çağırır.
- Geri Bildirim: İşlem sonuçlarını (başarı mesajları, "Oda Bulunamadı" gibi hata uyarıları) kullanıcıya net ve anlaşılır şekilde iletir.
- Akış Yönetimi: Çok aşamalı işlemlerde (örneğin Rezervasyon Yap: Müşteri ID al → Oda ID al → Tarihleri al) akışı adım adım yönetir.

Kullanılan Teknolojiler:

Standart Java Input/Output sınıfları (System.out, Scanner). Harici GUI kütüphanesi kullanılmaz.

Avantajlar:

- Kullanıcı deneyimi sade ve sezgiseldir; eğitim gerektirmez.
- İş mantığından tamamen bağımsızdır; UI değişse (örneğin gelecekte Web Arayüzü veya JavaFX eklenirse) alttaki iş mantığı ve veri katmanları etkilenmez.

3.2 İş Mantığı Katmanı Alt Sistemi (Management Subsystem / Business Logic Layer)

Amaç:

Sistemin tüm iş kurallarını, doğrulamaları ve hesaplamaları merkezi olarak yönetir. Sunum katmanından gelen talepleri işler ve veri katmanıyla koordineli çalışır.

Ana Bileşen:

- **ReservationService**

Sorumluluklar:

- Gereksinim Uygulama: Tüm fonksiyonel gereksinimleri (Rezervasyon oluşturma, Check-in, Check-out) uygular.

Rezervasyon Yönetimi:

- Tarih Kontrolü: Giriş tarihi çıkış tarihinden önce olmalı, geçmişe dönük işlem yapılmamalıdır.
- Müsaitlik Kontrolü: isDateAvailable() metodu ile aynı odaya çakışan tarihlerde rezervasyon yapılmasını engeller.
- Fiyat Hesaplama: calculatePrice() metodu ile gün sayısı ve oda fiyatını çarparak toplam tutarı belirler.

Süreç Yönetimi (Check-in/Out):

- Check-in: Sadece giriş tarihi "bugün" olan rezervasyonları kabul eder, oda durumunu "DOLU", rezervasyon durumunu "AKTIF" yapar.
- Check-out: Sadece "AKTIF" durumdaki işlemleri sonlandırır, odayı "BOŞ", rezervasyonu "TAMAMLANDI" statüsüne çeker.
- Merkezi Kontrol: İş kurallarını merkezi olarak tutar; böylece bir kural değiştiğinde (örn. Fiyat hesaplama formülü) tek bir yerden güncellenir.
- Façade Deseni: Sunum katmanı karmaşık mantıksal kontrolleri (tarih çakışması vb.) görmez, sadece basit yöntem çağrıları (createReservation) yapar.

Avantajlar:

- İş kuralları tek yerde toplandığı için bakım ve değişiklik kolaydır.
- Mantıksal hatalar (örn. Dolu odaya kayıt) burada yakalanır ve arayüze hata olarak bildirilir.

3.3 Veri Katmanı Alt Sistemi (Model & Persistence Subsystem / Data Layer)

Amaç: Tüm verilerin kalıcı olarak saklanmasını ve erişilmesini sağlar. Dosya sistemiyle etkileşimde bulunur, veri bütünlüğünü korur.

Ana Bileşenler:

FileManager

- Model sınıfları (Room, Customer, Reservation) – bu sınıflar veri nesnelerini temsil eder ve dosya satırına dönüştürme yöntemleri içerir.

Sorumluluklar:

Dosya İşlemleri (FileManager):

- rooms.txt, customers.txt, reservations.txt dosyalarını okur (BufferedReader) ve yazar (BufferedWriter).
- Dosya yoksa hata fırlatmak yerine boş liste dönerek sistemin çökmesini engeller.

- ID Üretimi: generateId() metodu ile dosyadaki son ID'yi okuyip bir artırarak benzersiz anahtar (Primary Key) üretir.
- Veri Güncelleme: Check-in ve Check-out işlemlerinde veri değişikliği gerektiği için, updateAllRooms ve updateAllReservations metotları ile RAM'deki güncel listeyi dosyanın üzerine yazar (Overwrite).

Model Sınıfları:

- Room: Oda tipi, kapasite, fiyat ve anlık durum (BOŞ/DOLU).
- Customer: Müşteri kimlik ve iletişim bilgileri.
- Reservation: Rezervasyon tarihleri, tutar ve yaşam döngüsü durumu (REZERVE/AKTIF/TAMAMLANDI).
- Her sınıf, kendini CSV formatına dönüştüren (toFileString) metodu içerir.

Avantajlar:

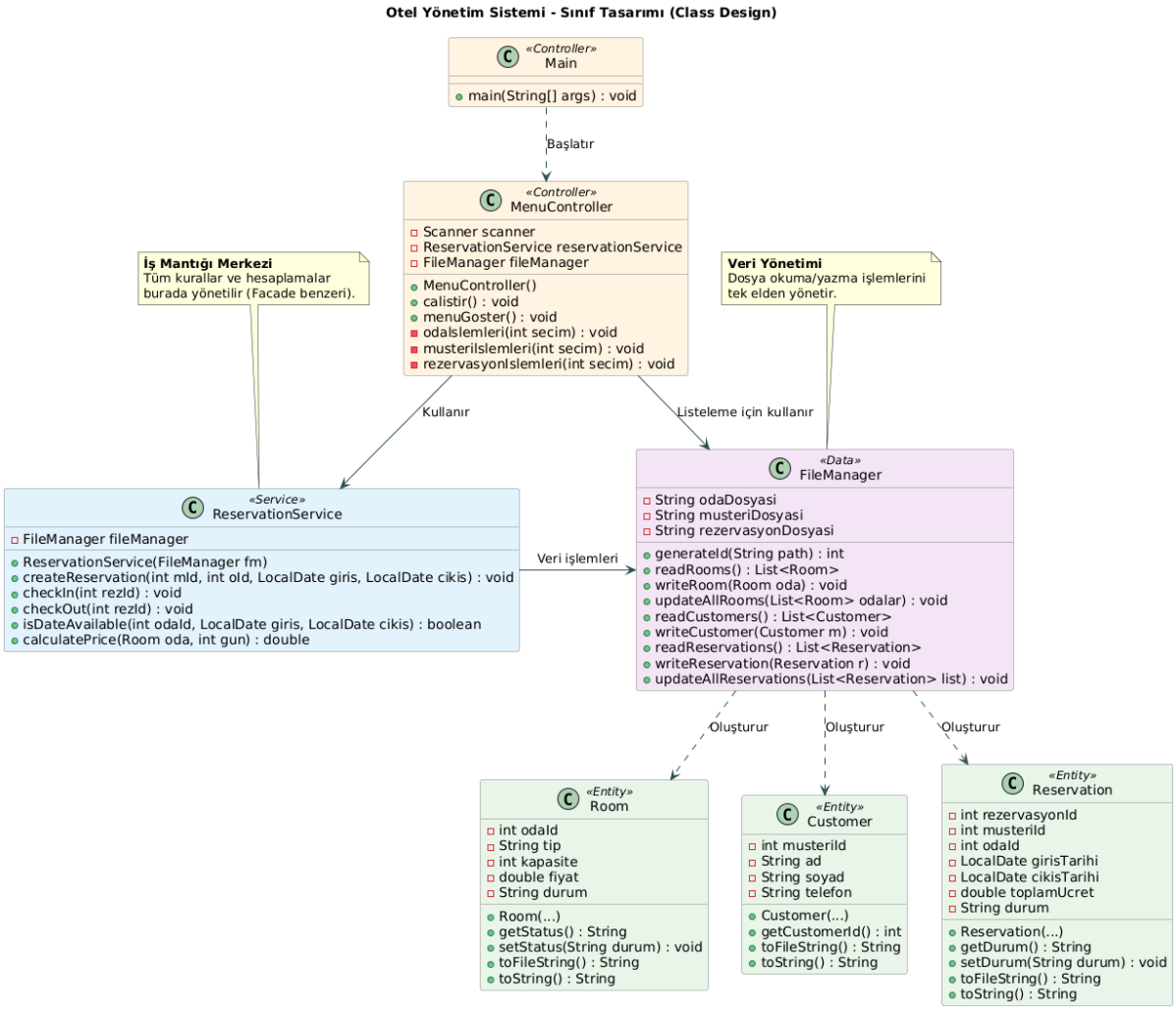
- Veritabanı kurulumu gerektirmediği için sistem hafif ve taşınabilirdir (.txt dosyaları ile her yerde çalışır).
- Veri erişimi merkezi olduğu için gelecekte gerçek bir veritabanına (MySQL, SQLite vb.) geçiş yapılırsa sadece FileManager sınıfının değiştirilmesi yeterlidir.
- ID üretimi ve veri formatlama standartlaştırılmıştır.

Özet Tablo: Alt Sistemler ve İlişkiler

Alt sistem	**Ana bileşen**	**Sorumlulukl Özeti**	**Bağımlılık Yönü**
3.2 Sunum Katmanı	Menu controller	Kullanıcı etkileşimi, menü yönetimi, girdi doğrulama	→ İş Mantığı Katmanı
3.2 İş Mantığı Katmanı	Reservation Service	İş kuralları, müsaitlik/tarih kontrolü, fiyat hesaplama	→ Veri Katmanı
3.3 Verı Katmanı	FileManager +Model	Dosya I/O, ID üretimi, veri nesneleri, kalıcı saklama	← İş Mantığı Katmanı (okuma/yazma/güncelleme)

4. SINIF TASARIMI (LOW-LEVEL DESIGN)

Sistem, nesne yönelimli tasarım prensiplerine (OOP) uygun olarak **6 ana sınıftan** oluşmaktadır. Bu sınıflar, katmanlı mimarinin gereksinimlerini karşılayacak şekilde tasarlanmış olup, sorumlulukları net bir şekilde ayrılmıştır. Aşağıda her sınıfın sorumlulukları, alanları, temel yöntemleri ve diğer sınıflarla ilişkileri detaylı olarak açıklanmıştır.



4.1 Room Sınıfı

Sorumluluklar:

- Sistemdeki odaların fiziksel özelliklerini ve anlık durumunu tutmak.
- Odanın durumunu (BOŞ/DOLU) güncellemek.

Alanlar:

- `odaId`: Benzersiz oda kimliği (int)
- `tip`: Oda tipi (Örn: "Single", "Double") (String)
- `kapasite`: Yatak sayısı (int)
- `fiyat`: Gecelik konaklama ücreti (double)
- `durum`: "BOŞ" veya "DOLU" (String)

Temel Yöntemler:

- `setStatus(String durum)`: Check-in ve Check-out işlemlerinde odanın durumunu günceller.
- `toFileString()`: Nesneyi `rooms.txt` dosyasına yazılacak CSV formatına (`id,tip,kapasite,fiyat,durum`) çevirir.
- `toString()`: Konsol ekranında oda bilgilerini kullanıcı dostu formatta gösterir.

4.2 Customer Sınıfı

Sorumluluklar:

- Müşteri kimlik ve iletişim bilgilerini güvenli ve tutarlı şekilde saklamak.

Alanlar:

- `musteriId`: Benzersiz müşteri kimliği (int)
- `ad`: Müşteri adı (String)
- `soyad`: Müşteri soyadı (String)
- `telefon`: İletişim numarası (String)

Temel Yöntemler:

- `toFileString()`: Nesneyi `customers.txt` formatına çevirir.
- `toString()`: Müşteri listesini ekrana yazdırmak için kullanılır.

4.3 Reservation Sınıfı

Sorumluluklar:

- Tek bir rezervasyon işlemini temsil etmek.
- Rezervasyonun yaşam döngüsünü (REZERVE -> AKTIF -> TAMAMLANDI) takip etmek.
- Giriş/Çıkış tarihlerini ve hesaplanan ücreti tutmak.

Alanlar:

- `rezervasyonId`: Benzersiz işlem kimliği (int)
- `musteriId`: Rezervasyonu yapan müşteri (int - Foreign Key mantığı)
- `odaId`: Rezerve edilen oda (int - Foreign Key mantığı)
- `girisTarihi`: Giriş yapılacak tarih (LocalDate)
- `cikisTarihi`: Çıkış yapılacak tarih (LocalDate)
- `toplamUcret`: Hesaplanan tutar (double)
- `durum`: Rezervasyonun güncel durumu (String)

Temel Yöntemler:

- `setDurum(String durum)`: Check-in sırasında durumu "AKTIF", Check-out sırasında "TAMAMLANDI" yapar.
- `calculatePrice()`: Toplam ücreti döndürür (Hesaplama serviste yapılıp buraya set edilir).
- `toFileString()`: Nesneyi `reservations.txt` formatına çevirir.

4.4 FileManager Sınıfı

Sorumluluklar:

- Tüm `.txt` dosyalarına (rooms, customers, reservations) merkezi erişim sağlamak.
- Veri bütünlüğünü korumak (dosya yeniden yazma ile).
- Benzersiz ID üretimi (Auto-Increment) yapmak.

Özellikler:

- Dosya yolları (`rooms.txt` vb.) private değişkenlerde saklanır, dışarıdan değiştirilemez.

Temel Yöntemler:

- `readRooms()`, `readCustomers()`, `readReservations()`: İlgili dosyaları okuyup `ArrayList` nesnelerine çevirir.
- `writeRoom()`, `writeCustomer()`, `writeReservation()`: Yeni kayıtları dosyanın sonuna ekler (Append mode).

- `updateAllRooms()`, `updateAllReservations()`: Check-in/Check-out sonrası güncellenmiş listeyi dosyanın üzerine yazar (Overwrite).
- `generateId(String dosyaYolu)`: Dosyadaki en büyük ID'yi bulup +1 fazlasını döndürür.

4.5 ReservationService Sınıfı

Sorumluluklar:

- Tüm iş kurallarını uygulamak (tarih kontrolü, müsaitlik, fiyat hesaplama).
- Rezervasyon, giriş ve çıkış süreçlerini yönetmek.
- **Façade** benzeri davranış sergileyerek UI katmanını karmaşık dosya işlemlerinden korumak.

Temel Yöntemler:

- `createReservation()`: Tarihlerin geçerliliğini ve odanın müsaitliğini kontrol eder, ardından kaydı oluşturur.
- `isDateAvailable()`: Seçilen tarih aralığında odanın başka bir rezervasyonla çakışıp çakışmadığını denetler (FR3.3).
- `checkIn()`: Rezervasyon tarihini kontrol eder, odayı "DOLU" ve rezervasyonu "AKTIF" yapar.
- `checkOut()`: Odayı "BOŞ" ve rezervasyonu "TAMAMLANDI" yapar.
- `calculatePrice()`: Gün sayısı * Oda Fiyatı formülünü uygular.

4.6 MenuController Sınıfı

Sorumluluklar:

- Kullanıcı arayüzünü (CLI menü) yönetmek.
- Kullanıcı girdilerini almak (`Scanner`) ve doğrulamak (`try-catch`).
- Kullanıcıyı ilgili `ReservationService` veya `FileManager` metoduna yönlendirmek.

Temel Yöntemler:

- `calistir()`: Programın ana döngüsünü (`while`) başlatır.
- `menuGoster()`: Seçenekleri ekrana yazar.
- `odaIslemleri()`, `musteriIslemleri()`, `rezervasyonIslemleri()`: Seçilen menü adımına göre ilgili alt işlemleri (örn. veri isteme) yürütür.

5. TASARIM KARARLARI

Proje kapsamında alınan temel tasarım kararları; dersin kısıtları, nesne yönelimli tasarım prensipleri (OOP), bakım kolaylığı ve taşınabilirlik kriterleri göz önünde bulundurularak şekillenmiştir. Aşağıda en önemli üç tasarım kararı detaylı olarak açıklanmakta ve gerekçeleri, avantaj/dezavantajları (trade-off'ları) ile birlikte sunulmaktadır.

5.1 Veritabanı Yerine Dosya Kullanımı

Karar: Sistemde kalıcı veri saklama için herhangi bir İlişkisel Veritabanı Yönetim Sistemi (RDBMS - MySQL, PostgreSQL vb.) kullanılmamış; bunun yerine düz metin dosyaları (`rooms.txt`, `customers.txt`, `reservations.txt`) tercih edilmiştir.

Gerekçe ve Detaylar:

- `FileManager` sınıfı, veri erişimi için tek yetkili nokta olarak tasarlanmıştır.
- Veri bütünlüğü, Check-in ve Check-out işlemlerinde dosyanın tamamının güncel liste ile yeniden yazılması (`updateAllRooms`, `updateAllReservations`) yöntemiyle sağlanır.
- Benzersiz ID üretimi (`generateId`), dosyadaki son kaydın ID'si okunup artırılarak manuel olarak yönetilir.

Avantajlar:

- **Taşınabilirlik:** Sistem kurulum gerektirmez. Java yüklü herhangi bir bilgisayarda `.jar` dosyası veya kaynak kod çalıştırılabilir.
- **Bağımlılık Yok:** JDBC sürücüsü veya harici kütüphane kurulumuna ihtiyaç duyulmaz.
- **Eğitim Odaklılık:** Dosya I/O işlemleri ve String manipülasyonu (CSV parsing) becerilerini geliştirir.

Dezavantajlar (Trade-off'lar):

- **Performans Darboğazı:** Veri güncellemek için dosyanın tamamının silinip yeniden yazılması gerekir. Binlerce kayıt olduğunda bu işlem yavaşlar (Zaman Karmaşıklığı: O(N)).
- **Eşzamanlılık Sorunu:** İki resepsiyonist aynı anda işlem yaparsa veri kaybı yaşanabilir (Dosya kilitleme mekanizması yoktur).
- **Sorgu Zorluğu:** SQL'deki `WHERE`, `JOIN` gibi güçlü sorgular olmadığı için filtreleme işlemleri Java tarafında (RAM'de) yapılmak zorundadır.

Sonuç: Projenin kapsamı ve eğitim amacı doğrultusunda dosya sistemi yeterli görülmüştür. Ancak mimari, gelecekte `FileManager` sınıfı değiştirilerek SQL veritabanına geçişe uygun tasarlanmıştır (Open/Closed Prensibi).

5.2 CLI (Komut Satırı) Arayüz Tercihi

Karar: Kullanıcı arayüzü olarak grafik arayüz (GUI - JavaFX/Swing) yerine **Komut Satırı Arayüzü (CLI)** tercih edilmiştir.

Gerekçe ve Detaylar:

- `MenuController` sınıfı, kullanıcı etkileşimini `Scanner` ve `System.out.println` kullanarak yönetir.
- Ekranlar sade metin listeleri ve numaralı menüler (1. Ekle, 2. Listele vb.) şeklinde tasarlanmıştır.

Avantajlar:

- **Geliştirme Hızı:** Arayüz tasarımına (buton yerleşimi, renkler vb.) zaman harcamak yerine, arka plandaki iş mantığına (`ReservationService`) odaklanılmıştır.
- **Kaynak Tüketimi:** Çok düşük sistem kaynağı (RAM/CPU) tüketir.
- **Platform Bağımsızlığı:** Windows terminal, Linux bash veya macOS terminalinde aynı şekilde çalışır.

Dezavantajlar (Trade-off'lar):

- **Kullanıcı Deneyimi:** Görsel öğelerden yoksundur. Kullanım klavye ile sınırlıdır, fare desteği yoktur.
- **Veri Gösterimi:** Büyük tabloları veya takvim görünümlerini konsolda göstermek zordur.

Sonuç: CLI tercihi, kodun karmaşıklığını azalttığı ve doğrudan temel algoritmalara odaklanmayı sağladığı için bu proje için en uygun seçimdir.

5.3 İş Mantığı Katmanının (ReservationService) Kullanımı

Karar: Tüm iş kurallarını (Fiyat hesaplama, Müsaitlik kontrolü, Tarih doğrulama) `MenuController` içine gömmek yerine, ayrı bir **`ReservationService`** sınıfı oluşturulmuştur.

Gerekçe ve Detaylar:

- `MenuController` (Sunum Katmanı), `FileManager` (Veri Katmanı) ile doğrudan karmaşık işlemler yapmaz. Tüm talepler `ReservationService` üzerinden geçer.
- Örnek: "Oda Müsait mi?" kontrolü (`isDateAvailable`) arayüzde değil, serviste yapılır.
- Bu yapı, **Façade (Ön Yüz)** tasarım desenine benzer bir rol üstlenir.

Avantajlar:

- **Sorumlulukların Ayrılması (SoC):** Arayüz kodları ile hesaplama kodları birbirine karışmaz.
- **Tekrar Kullanılabilirlik:** Fiyat hesaplama mantığı (`calculatePrice`) hem rezervasyon oluştururken hem de faturayı görüntülerken aynı metot üzerinden çağrılır.
- **Test Edilebilirlik:** `ReservationService` sınıfı, kullanıcı arayüzünden bağımsız olarak test edilebilir (Örn: `checkIn` metodu tek başına çalıştırılıp sonucu görülebilir).

Dezavantajlar (Trade-off'lar):

- **Kod Hacmi:** Projeye ekstra bir sınıf ve metot katmanı ekler, dosya sayısı artar.
- **Dolaylı Erişim:** Veriye ulaşmak için bir katman daha geçilmesi gerekir, ancak modern işlemcilerde bu performans kaybı ihmal edilebilir düzeydedir.

Sonuç: `ReservationService` kullanımı, projenin modülerliğini sağlayan en kritik karardır. Bu sayede kod "Spagetti Kod" olmaktan kurtarılmış, profesyonel bir mimariye kavuşturulmuştur.

6. SONUÇ

Bu proje kapsamında geliştirilen **Otel Yönetim Sistemi**, Yazılım İnşası dersi boyunca teorik olarak ele alınan nesne yönelimli analiz, tasarım ve kodlama süreçlerinin başarılı bir pratik uygulamasıdır. Proje, analiz raporunda tanımlanan tüm fonksiyonel (Oda/Müşteri Yönetimi, Rezervasyon, Check-in/Out) ve fonksiyonel olmayan gereksinimleri eksiksiz karşılamış; ayrıca hedeflenen tasarım kriterlerine (bakım kolaylığı, modülerlik) tam uyum sağlamıştır.

Analiz aşamasında belirlenen kullanım senaryoları (Use Case'ler), tasarım aşamasında **Katmanlı Mimari (Presentation – Business – Data)** yapısına dönüştürülmüştür. Bu mimari yaklaşım sayesinde:

1. **İş Kuralları Ayrıştırıldı:** Fiyat hesaplama ve tarih çakışma kontrolleri `ReservationService` sınıfında toplanarak, arayüz kodunun (`MenuController`) karmaşıklaşması önlenmiştir.
2. **Veri Bağımsızlığı Sağlandı:** Tüm dosya işlemleri `FileManager` sınıfına izole edilerek, gelecekte SQL veritabanına geçiş için esnek bir zemin hazırlanmıştır.
3. **Veri Bütünlüğü Korundu:** Dosya tabanlı sistemin kısıtlarına rağmen, RAM üzerinde güncel liste tutma ve `updateAll...` yöntemleri ile verilerin tutarlı kalması sağlanmıştır.

Uygulama aşamasında **Java** programlama dili kullanılarak tüm sınıflar (`Room`, `Customer`, `Reservation`, `FileManager`, `MenuController`) tasarlandığı gibi hayata geçirilmiştir. Kullanıcı arayüzü olarak tercih edilen **CLI (Komut Satırı)** yapısı, görsel karmaşıklığa girmeden sistemin temel algoritmalarına odaklanılmasını sağlamış ve hata yönetimi (`try-catch` blokları) ile güçlendirilmiştir. SOLID prensiplerinden özellikle **Tek Sorumluluk Prensibi (SRP)** titizlikle uygulanmış, her sınıfın sadece kendi görevini yapması sağlanmıştır.