---

1. TASARIM RAPORU (DESIGN REPORT)

DEPO YÖNETİM SİSTEMİ

Ders: Yazılım İnşaası **Proje Ekibi:**

1. Murat Efe ÖZOĞUL - 2410238037
2. Ömer ÇELİK - 2410238034
3. Berat GÜNGÖR - 2410238050
4. **Teslim Tarihi:** 04.01.2026

İÇİNDEKİLER

---

1. GİRİŞ

1.1 Dokümanın Amacı

Bu doküman, Java programlama dili ile geliştirilen **Depo Yönetim Sistemi** yazılımının detaylı tasarımını (Detailed Design) belgelemektedir. Analiz aşamasında belirlenen fonksiyonel gereksinimlerin (stok takibi, raf yönetimi, güvenli giriş), teknik mimariye nasıl dönüştürüldüğü, sınıfların iç yapısı ve algoritmaların işleyiş mantığı burada açıklanmıştır.

1.2 Projenin Kapsamı

Sistem; veritabanı kurulumu gerektirmeyen, taşınabilir (portable), konsol tabanlı bir masaüstü uygulamasıdır. Özellikle **parçalı stok dağıtımı** (bir ürünün birden fazla rafta tutulabilmesi) ve **veri tutarlılığı** (dolu rafların silinememesi) üzerine özelleşmiş bir mimariye sahiptir.

---

2. TASARIM HEDEFLERİ VE İLKELERİ

2.1 Temel Tasarım Hedefleri

2.1.1 İş Kurallarını ve Kullanıcı Arayüzünü Ayrıştırmak (Separation of Concerns)

- **Açıklama: Kullanıcıdan veri alma, menü gösterme ve çıktı üretme işlemleri `Main` ve `Listeleme` sınıflarında (Sunum Katmanı); stok hesaplama, kapasite kontrolü, giriş doğrulama ve dağıtım algoritmaları ise `MudurManager`, `UrunManager` ve `RafManager` sınıflarında (İş Mantığı Katmanı) toplanmıştır.**
- **Gerekçe: Arayüz kodları ile iş mantığının iç içe geçmesi (Spaghetti Code), kodun okunabilirliğini düşürür ve bakımı zorlaştırır. Bu ayrım sayesinde (SoC prensibi), arayüz değişse bile (örneğin GUI'ye geçiş) iş mantığı etkilenmez.**
- **Avantaj: Kodun modülerliği artar, hata ayıklama kolaylaşır ve her katman bağımsız olarak test edilebilir.**
- **Uygulama: `Main` sınıfı hiçbir iş kuralı içermez, sadece kullanıcı girdisini alır ve ilgili Manager sınıfının metodunu (`urunManager.urunKayitEkle()`) çağırır.**

2.1.2 Veri Erişimini Merkezileştirmek ve Soyutlamak

- **Açıklama: Tüm dosya okuma/yazma işlemleri, ID yönetimi veya serileştirme detayları `DosyaIslemleri` sınıfında toplanmıştır. Diğer sınıflar dosya formatını (Binary/Text) veya yazma yöntemini bilmez, sadece bu sınıfı kullanır.**
- **Gerekçe: Dosya erişim kodlarının sisteme dağılması veri tutarsızlığına ve kod tekrarına yol açar. Merkezi yapı, I/O hatalarını tek noktadan yönetmeyi sağlar.**

- **Avantaj: Kod tekrarı (Code Duplication) önlenir (DRY Prensibi). Gelecekte veri saklama yöntemi değişirse (örneğin JSON veya SQL) sadece `DosyaIslemleri` sınıfını değiştirmek yeterli olur.**
- **Uygulama: `DosyaIslemleri` sınıfında tanımlanan generic `<T>` metotlar (`kaydet`, `yukle`) sayesinde tüm varlıklar (`Urun`, `Raf`, `Mudur`) aynı altyapıyı kullanır.**

2.1.3 Veri Bütünlüğünü ve Tutarlılığını Garanti Altına Almak

- **Açıklama: Sistem, veri kaybına yol açabilecek kritik işlemlerde (özellikle Raf Silme) sıkı kontroller uygular. İlişkisel veri bütünlüğü (Referential Integrity) kod seviyesinde sağlanır.**
- **Gerekçe: Kullanıcı hatası sonucu içerisinde ürün bulunan bir rafın silinmesi, stok envanterinde tutarsızlığa ("Hayalet Ürünler") neden olur.**
- **Avantaj: Sistem "Fail-Safe" (Hata durumunda güvenli) çalışır; kullanıcı hatalarına karşı dirençlidir.**
- **Uygulama: `RafManager`, silme işleminden önce `UrunManager` ile iletişime geçerek `rafDoluMu(index)` kontrolü yapar. Onay gelmezse silme işlemini reddeder.**

2.1.4 Esnek ve Akıllı Stok Yönetimi (Parçalı Dağıtım)

- **Açıklama: Ürünlerin tek bir rafa sığmadığı senaryolar için esnek bir nesne modeli tasarlanmıştır. Sistem, statik raf ataması yerine dinamik ve çoklu raf yerleşimini destekler.**
- **Gerekçe: Gerçek dünya depo senaryolarında büyük parti mallar tek bir rafa sığmayabilir. Sistemin bu durumu yönetememesi (işlemi iptal etmesi) operasyonel verimsizlik yaratır.**
- **Avantaj: Depo kapasitesi maksimum verimle kullanılır ve kullanıcıya operasyonel esneklik sağlanır.**
- **Uygulama: `Urun` sınıfındaki `List<Integer> rafListesi` alanı sayesinde, tek bir ürün nesnesi mantıksal olarak birden fazla fiziksel raf ID'si ile ilişkilendirilebilir (One-to-Many İlişki).**

2.1.5 Analiz Gereksinimlerini Birebir Karşılamak

- **Açıklama: Analiz raporunda tanımlanan tüm Use Case'ler (UC1–UC9), özellikle "Güvenli Giriş" ve "İnteraktif Dağıtım" senaryoları sistemde eksiksiz uygulanmıştır.**

- **Gerekçe: Yazılımın başarısı, belirlenen gereksinimleri ne kadar doğru karşıladığı ile ölçülür.**
- **Uygulama: `MudurManager` içindeki sayaçlı döngü ile 3 hak kuralı ve `UrunManager` içindeki `while` döngüsü ile interaktif dağıtım senaryosu koda dökülmüştür.**

2.2 Tasarım Kriterleri (Design Criteria)

Tasarım kriterleri, sistemin fonksiyonel olmayan gereksinimlerini (NFR) karşılamak üzere üç ana başlıkta toplanmıştır:

2.2.1 Kullanılabilirlik (Usability)

- **Hata Toleransı (Robustness): Kullanıcı sayısal bir alana harf girdiğinde programın çökmesi engellenmiştir. `Scanner` girişleri `try-catch` blokları ve `hasNextInt()` kontrolleriyle sarmalanmıştır.**
- **İnteraktif Yönlendirme: Karmaşık işlemlerde (Stok Dağıtımı), sistem kullanıcıyı "Evet/Hayır" soruları ve adım adım yönlendirmelerle (Wizard benzeri akış) destekler.**
- **Net Geri Bildirim: Her işlem sonucunda (Başarılı, Hata, Uyarı) kullanıcıya konsol üzerinden açık mesajlar verilir.**

2.2.2 Performans (Performance)

- **Binary Serialization: Veriler metin (TXT) yerine Java'nın yerel `.bin` formatında saklanır. Bu yöntem, metin ayrıştırma (parsing) maliyetini ortadan kaldırır ve I/O işlemlerini hızlandırır.**
- **In-Memory İşlem: Uygulama açılışta tüm verileri RAM'e yükler (`ArrayList`). Arama, listeleme ve güncelleme işlemleri RAM üzerinde yapılır, bu sayede disk erişim gecikmeleri minimuma indirilir.**

2.2.3 Bakım Kolaylığı (Maintainability)

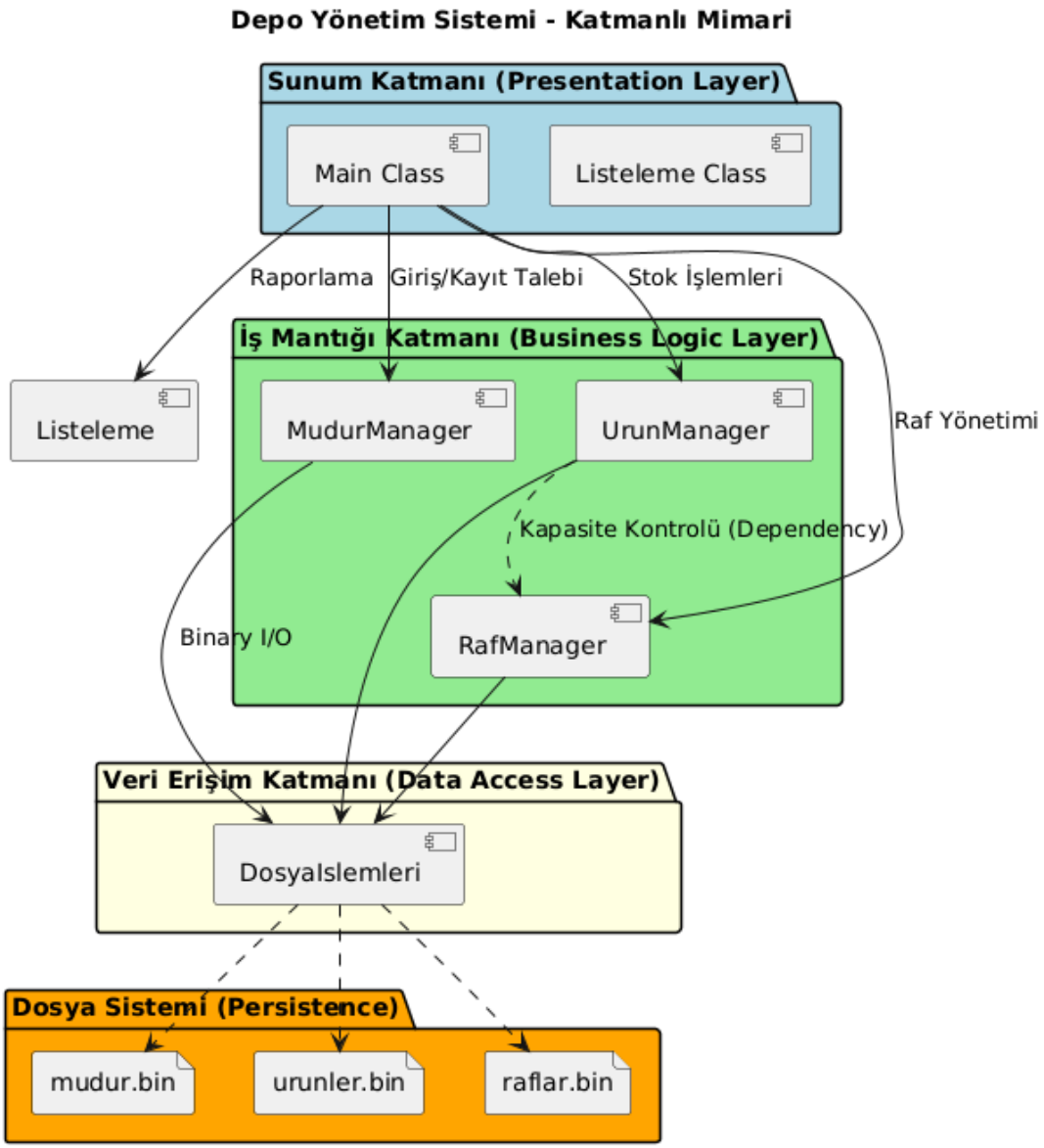
- **Modüler Mimari: Sınıflar, Tek Sorumluluk Prensibi'ne (SRP) göre tasarlanmıştır. Bir modüldeki (örneğin Raf yönetimi) değişiklik veya hata, diğer modülleri (Giriş sistemi) etkilemez.**
- **Generic Yapı: Veri erişim katmanının generic (`<T>`) yapısı sayesinde, sisteme ileride "Personel" veya "Fatura" gibi yeni varlıklar eklendiğinde dosya okuma/yazma kodlarının yeniden yazılmasına gerek yoktur.**

3. SİSTEM MİMARİSİ

3.1 Mimari Yaklaşım (Katmanlı Mimari)

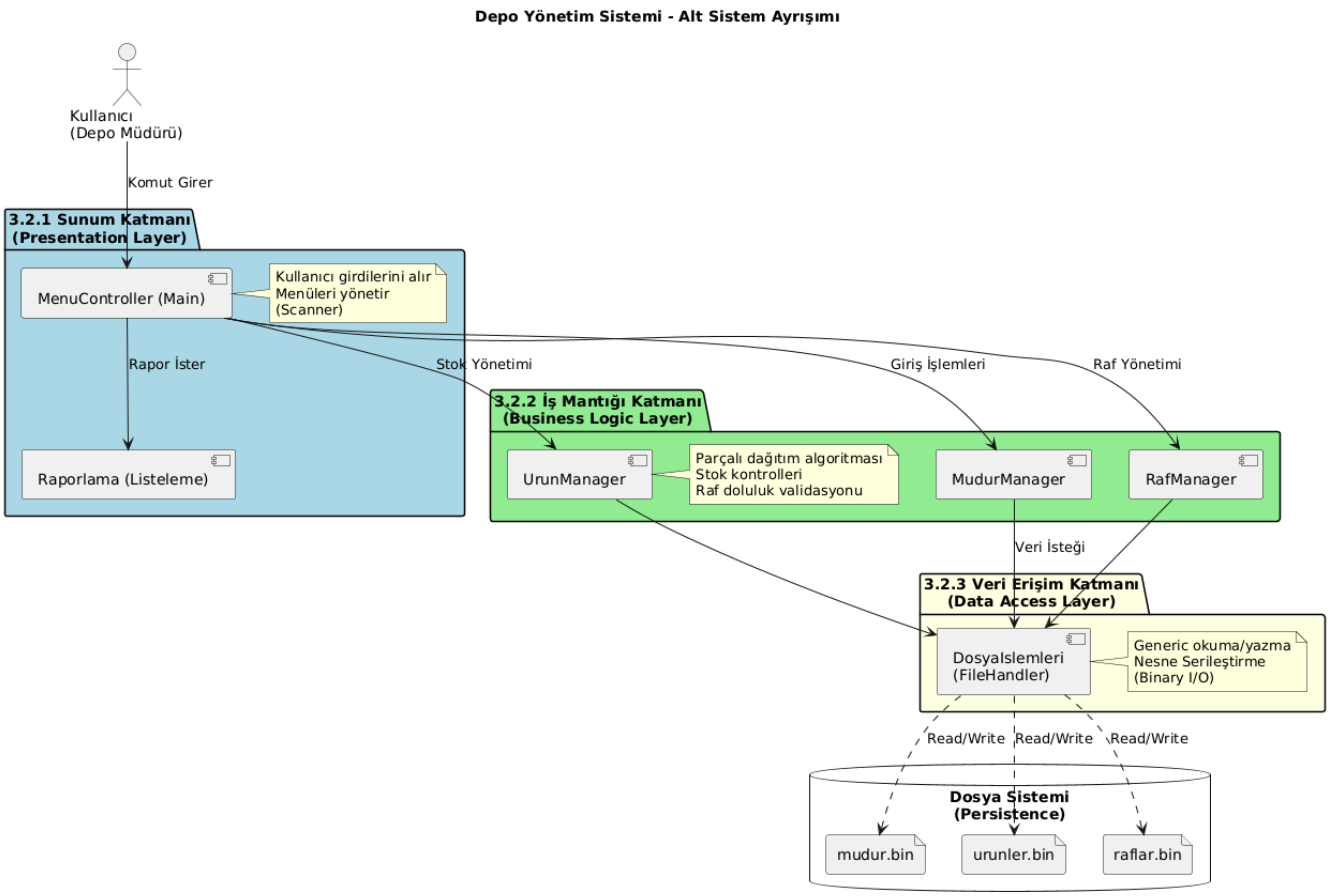
Sistem, sorumlulukların yatay olarak bölündüğü **3-Katmanlı Mimari** modelini benimsemiştir.

Şekil 1: Katmanlı Mimari Diyagramı



3.2 Alt Sistem Ayrışımı

- Sistem, sorumlulukların net bir şekilde ayrıldığı üç temel alt sisteme bölünmüştür. Bu ayrım, modüller arası bağımlılığı (coupling) azaltarak bakım kolaylığı sağlar.
- **Şekil 2: Alt Sistem Ayrışım Diyagramı**
- *(Aşağıdaki kodu PlantUML ile görselleştirebilirsin)*



Alt Sistemlerin Sorumlulukları

1. **Sunum Katmanı (Presentation Layer):**
    - **Bileşenler: `Main.java`, `Listeleme.java`.**
    - **Görevi: Kullanıcıdan `Scanner` aracılığıyla komut almak, hatalı girişleri (harf girilmesi vb.) yakalamak ve işlem sonuçlarını konsol ekranında göstermektir. İş mantığı kurallarını (örneğin "stok negatif olamaz") bilmez, sadece sonucu gösterir.**
2. **İş Mantığı Katmanı (Business Layer):**
    - **Bileşenler: `UrunManager`, `RafManager`, `MudurManager`.**
    - **Görevi: Sistemin "beyni"dir.**
        - **UrunManager: Parçalı ekleme algoritmasını çalıştırır, stok artırır/azaltır.**
        - **RafManager: Depo kapasitesini hesaplar, raf silinirken doluluk kontrolü yapar.**
        - **MudurManager: Giriş denemelerini sayar (3 hak kontrolü).**
3. **Veri Erişim Katmanı (Data Access Layer):**
    - **Bileşenler: `DosyaIslemleri`.**
    - **Görevi: İş mantığı katmanından gelen nesneleri (Object) alıp diske yazar (Serialization) veya diskten okuyup nesneye çevirir (Deserialization). Dosya isimlerini ve yollarını sadece bu katman bilir.**

3.3 Özet Tablo: Alt Sistemler ve İlişkiler

Aşağıdaki tablo, sistemin katmanlı yapısındaki bileşenlerin sorumluluklarını ve bağımlılık yönlerini özetlemektedir.

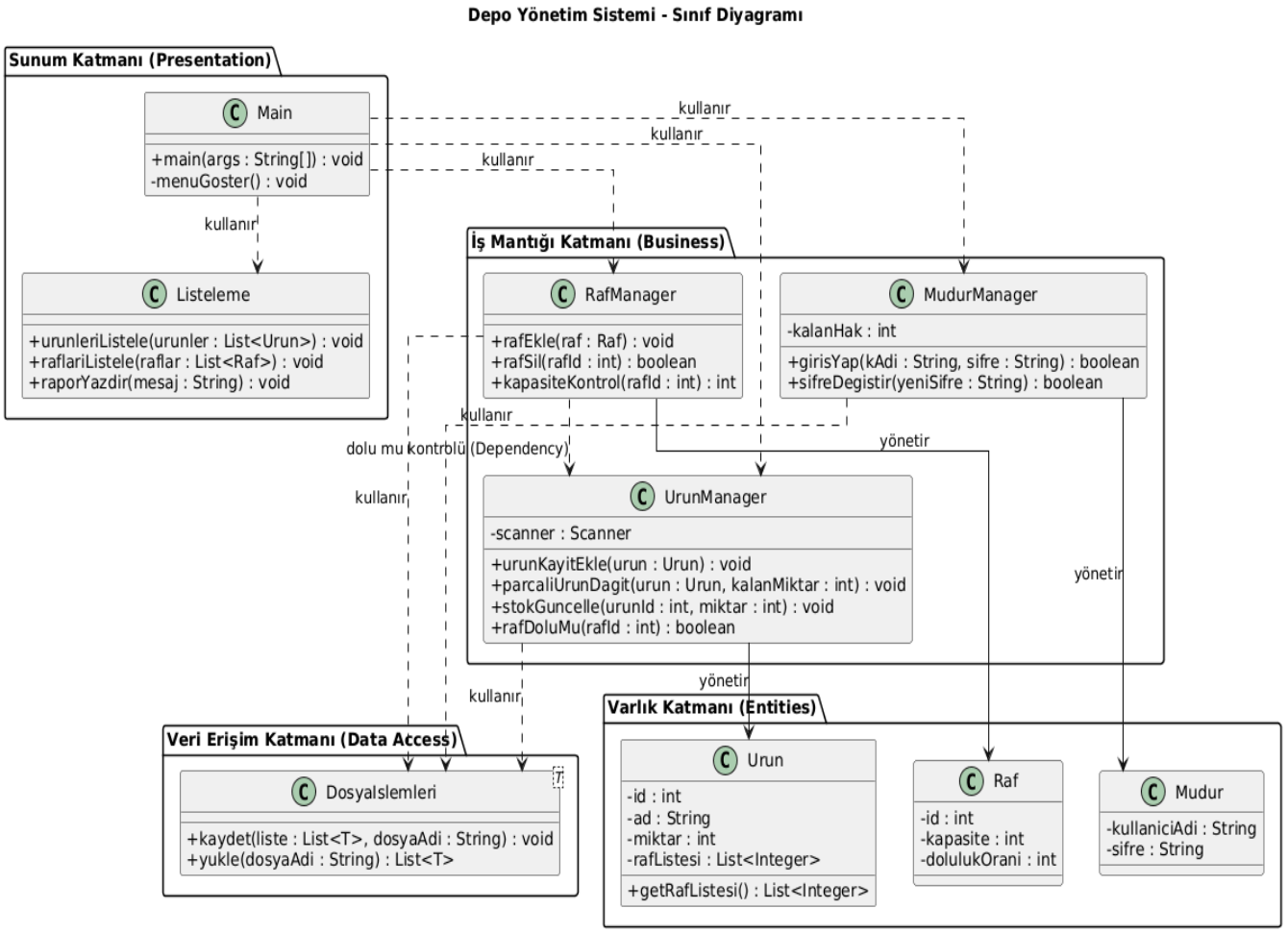
Alt Sistem	Ana Bileşenler	Sorumluluk Özeti	Bağımlılık Yönü
3.2.1 Sunum Katmanı	Main, Listeleme	Kullanıcı etkileşimi, menü yönetimi, girdi doğrulama (Scanner) ve rapor çıktıları.	→ İş Mantığı Katmanı
3.2.2 İş Mantığı Katmanı	MudurManager, UrunManager, RafManager	İş kuralları, stok dağıtım algoritmaları, kapasite kontrolleri ve güvenlik (3 hak) yönetimi.	→ Veri Erişim Katmanı
3.2.3 Veri Erişim Katmanı	Dosyalslemleri	Generic dosya okuma/yazma, nesne serileştirme (Serialization) ve veri kalıcılığı.	← İş Mantığı Katmanı (Sadece çağrılır)

---

4. SINIF TASARIMI VE DETAYLI API

Sistemin kod yapısını ve sınıflar arası ilişkileri gösteren detaylı diyagram aşağıdadır.

Şekil 2: Detaylı Sınıf Diyagramı (Class Diagram)



4.1 Sınıf Detayları

Bu bölüm, Depo Yönetim Sistemi'ni oluşturan sınıfların sorumluluklarını, sahip oldukları temel metodları ve diğer sınıflarla olan etkileşimlerini (API) açıklamaktadır. Sınıflar, mimari tasarıma uygun olarak katmanlar altında gruplandırılmıştır.

A. Sunum Katmanı Sınıfları (Presentation Layer)

1. Main (Ana Sınıf)

- **Görevi:** Uygulamanın giriş noktasıdır (Entry Point). Programın yaşam döngüsünü yönetir.

- **Temel Metodlar:**
    - `main(String[] args)`: Uygulamayı başlatır, veri dosyalarının varlığını kontrol eder ve `MudurManager` üzerinden giriş ekranını çağırır.
    - `menuGoster()`: Kullanıcıya yapabileceği işlemleri (Ürün Ekleme, Raf Listeleme vb.) konsol üzerinden seçenekler halinde sunar ve `Scanner` ile seçim alır.
- **Not:** Bu sınıf herhangi bir iş kuralı (business logic) içermez; sadece ilgili Manager sınıflarını tetikler.

2. Listeleme

- **Görevi:** Verilerin kullanıcıya okunabilir ve düzenli bir formatta (tablo benzeri yapılarla) sunulmasından sorumludur.
- **Temel Metodlar:**
    - `urunleriListele(List<Urun> urunler)`: Bellekteki ürün listesini alır; ID, ad, miktar ve bulundukları raf bilgilerini formatlayarak ekrana basar.
    - `raflariListele(List<Raf> raflar)`: Depodaki rafların doluluk oranlarını ve kapasitelerini görselleştirir.
    - `raporYazdir(String mesaj)`: İşlem sonuçlarını (Başarılı/Hata) kullanıcıya standart bir formatta bildirir.

B. İş Mantığı Katmanı Sınıfları (Business Layer)

3. UrunManager

- **Görevi:** Sistemin en karmaşık sınıfıdır. Ürün ekleme, stok güncelleme ve parçalı dağıtım algoritmalarını yönetir.
- **Temel Metodlar:**
    - `urunKayitEkle(Urun urun)`: Kullanıcıdan alınan ürün bilgilerini işler. Eğer ürün tek bir rafa sığmıyorsa, kullanıcıya sorarak işlemi `parcaliUrunDagit` metoduna devreder.
    - `parcaliUrunDagit(Urun urun, int kalanMiktar)`: **(Kritik Algoritma)** Kapasite yetersizliğinde devreye girer. Kalan miktarı sırasıyla boş kapasitesi olan diğer raflara dağıtır ve ürünün `rafListesi` özelliğini günceller.
    - `stokGuncelle(int urunId, int miktar)`: Mevcut bir ürünün stok miktarını artırır veya azaltır.
    - `rafDoluMu(int rafId)`: Belirtilen rafta ürün olup olmadığını kontrol eder. Bu metod, `RafManager` tarafından raf silme işlemi öncesinde güvenlik kontrolü için çağrılır.

4. RafManager

- **Görevi:** Depo kapasitesini ve raf düzenini yönetir.
- **Temel Metodlar:**
    - `rafEkle(Raf raf)`: Sisteme yeni bir raf tanımlar.

    - `rafSil(int rafId)`: Bir rafı silmeden önce `UrunManager` ile iletişime geçerek rafın boş olup olmadığını doğrular (Referential Integrity). Dolu rafların silinmesini engeller.
    - `kapasiteKontrol(int rafId)`: Belirtilen rafın anlık olarak ne kadar boş alanı kaldığını hesaplar ve döner.

5. MudurManager

- **Görevi:** Güvenlik ve yetkilendirme işlemlerini yürütür.
- **Temel Metodlar:**
    - `girisYap(String kAdi, String sifre)`: Kullanıcı bilgilerini doğrular. Analiz raporunda belirtilen "3 yanlış giriş hakkı" kuralını (`kalanHak` sayacı ile) burada işletir.
    - `sifreDegistir(String yeniSifre)`: Mevcut oturum sahibinin şifresini güvenli bir şekilde günceller.

C. Veri Erişim Katmanı Sınıfları (Data Access Layer)

6. DosyaIslemleri<T> (Generic Class)

- **Görevi:** Tüm nesnelerin (Ürün, Raf, Müdür) disk üzerindeki `.bin` dosyalarına yazılmasını ve okunmasını sağlar.
- **Özelliği:** Generic (`<T>`) yapısı sayesinde, her varlık tipi için ayrı ayrı kod yazılmasını engeller (DRY Prensibi).
- **Temel Metodlar:**
    - `kaydet(List<T> liste, String dosyaAdi)`: Verilen nesne listesini Java Serialization API kullanarak binary formatta diske yazar.
    - `yukle(String dosyaAdi)`: Diskten okuduğu binary veriyi `List<T>` tipine dönüştürerek (Deserialization) RAM'e yükler.

D. Varlık Katmanı Sınıfları (Entity Layer)

7. Urun

- **Görevi:** Ürün verilerini taşıyan veri modelidir (POJO).
- **Önemli Alanlar:**
    - `id`, `ad`, `miktar`: Temel ürün bilgileri.
    - `List<Integer> rafListesi`: **(Kritik Tasarım)** Bir ürünün birden fazla rafta bulunabilmesini sağlayan dinamik listedir. Tek bir raf ID'si yerine raf ID'lerinin listesini tutar.

8. Raf

- **Görevi:** Raf verilerini taşıyan veri modelidir.
- **Önemli Alanlar:**
    - `id`: Raf numarası (örn: 1, 2, 3).
    - `kapasite`: Rafın alabileceği maksimum ürün miktarı.

9. Mudur

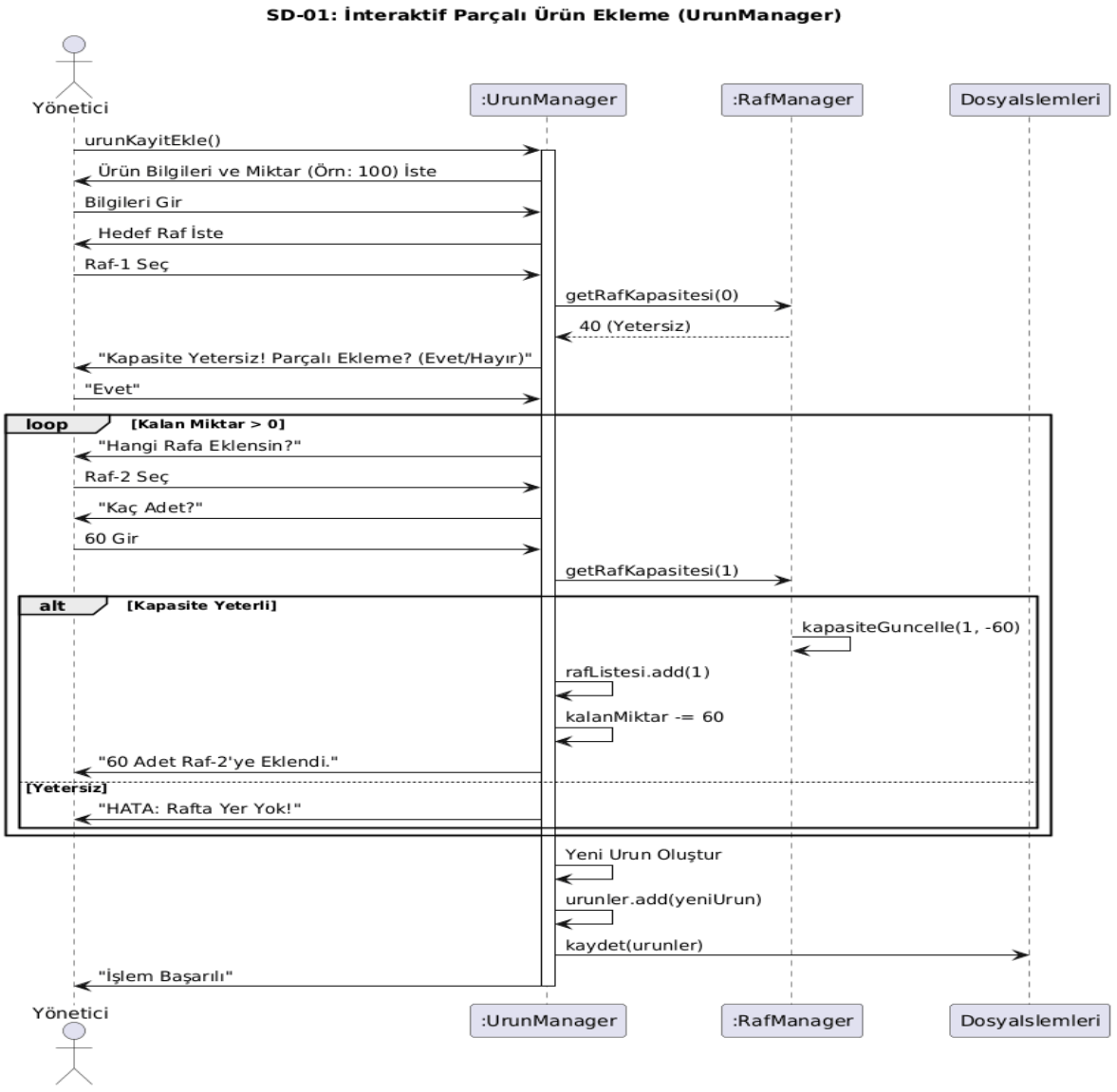
- **Görevi:** Yönetici kimlik bilgilerini taşıyan basit veri sınıfıdır.
- **Önemli Alanlar:** `kullaniciAdi`, `sifre`.
-

---

5. DİNAMİK TASARIM (SEQUENCE DİYAGRAMLARI)

Sistemin en kritik algoritması olan "Kapasite Yetersizliğinde İnteraktif Parçalı Ekleme" sürecinin akış diyagramı aşağıdadır.

Şekil 3: İnteraktif Parçalı Ürün Ekleme Senaryosu



6. KRİTİK TASARIM KARARLARI

1. **Veri Kalıcılığı (Binary Serialization):**
    - **Karar:** SQL yerine Java Serialization (`.bin` dosyaları).
    - **Gerekçe:** Kurulumsuz çalışabilirlik (Portability) ve nesne grafiğinin (Object Graph) veri kaybı olmadan saklanması.
2. **İnteraktif Algoritma:**
    - **Karar:** Otomatik dağıtım yerine kullanıcıya soran yapı.
    - **Gerekçe:** Depo düzeninde son kararın operatörde olması operasyonel esneklik sağlar.
3. **Hata Yönetimi:**
    - **Karar:** `try-catch` ve `while` döngüleri.
    - **Gerekçe:** Kullanıcı sayı yerine harf girdiğinde programın çökmesini engellemek (`hasNextInt` kontrolü).

---

7. SONUÇ

Bu Tasarım Raporu, **Depo Yönetim Sistemi**'nin kodlama aşamasında uygulanan mimariyi belgelemektedir. Sistem; **Katmanlı Mimari** yapısı, **Generic Veri Erişimi** ve **İnteraktif Stok Algoritmaları** ile modüler, güvenli ve genişletilebilir bir yapıdadır. Analiz aşamasındaki tüm gereksinimler, Java'nın OOP yetenekleri kullanılarak başarıyla tasarıma aktarılmıştır.