FİNAL RAPORU (FINAL REPORT)

DEPO YÖNETİM SİSTEMİ

Ders: Yazılım İnşaası **Proje Ekibi:**

- Murat Efe ÖZOĞUL - 2410238037
- Ömer ÇELİK - 2410238034
- Berat GÜNGÖR - 2410238050 **Teslim Tarihi:** 04.01.2026

İÇİNDEKİLER

---

1. GİRİŞ

Bu rapor, Java programlama dili kullanılarak geliştirilen **Depo Yönetim Sistemi** yazılımının final aşamasını belgelemektedir. Proje, analiz aşamasında belirlenen "dinamik stok takibi" ve tasarım aşamasında kurgulanan "parçalı raf dağıtım algoritması" temelleri üzerine inşa edilmiştir.

Final raporunun amacı; soyut tasarımın somut kod bloklarına (`Manager` sınıfları, `Binary Serialization`, `Entity` yapıları) nasıl dönüştürüldüğünü teknik detaylarla açıklamak ve sistemin operasyonel yetkinliğini ortaya koymaktır. Rapor kapsamında özellikle `UrunManager` sınıfındaki karmaşık stok dağıtım mantığı ve `DosyaIslemleri` sınıfındaki generic veri kalıcılığı çözümleri detaylandırılmıştır.

2. GERÇEKLEŞTİRİM SÜRECİ

Depo Yönetim Sistemi, **Java SE (Standard Edition)** kullanılarak, Nesne Yönelimli Programlama (OOP) prensiplerine tam sadık kalınarak geliştirilmiştir.

2.1 Yazılım Mimarisi ve Kod Yapısı

Proje, sorumlulukların net olarak ayrıldığı (Separation of Concerns) modüler bir yapıya sahiptir:

- **Varlık Katmanı (Entities):** `Urun`, `Raf` ve `Mudur` sınıfları veriyi taşıyan temel yapı taşlarıdır. Veri bütünlüğünü sağlamak için `implements Serializable` arayüzü kullanılmış, böylece nesneler doğrudan binary formatta saklanabilir hale getirilmiştir.
- **İş Mantığı Katmanı (Managers):**
  - `UrunManager.java`: Ürün ekleme, silme, taşıma ve **Parçalı Dağıtım (Algoritmik Stok)** işlemlerini yönetir.
  - `RafManager.java`: Raf kapasite kontrollerini ve raf silme güvenliğini sağlar.
  - `MudurManager.java`: Kimlik doğrulama ve giriş hakkı (`hakTukendiMi`) kontrollerini yapar.
- **Veri Erişim Katmanı:** `DosyaIslemleri.java` sınıfı, Generic (`<T>`) yapıda tasarlanarak kod tekrarı önlenmiş; tek bir `kaydet/yukle` metodu ile tüm nesne tiplerinin yönetilmesi sağlanmıştır.
- **Sunum ve Yardımcılar:** `Main.java` kullanıcı menüsünü yönetirken, `Listeleme.java` verilerin formatlı şekilde konsola basılmasından sorumludur.

2.2 Kullanılan Araçlar

- **Dil:** Java
- **IDE:** IntelliJ IDEA / Eclipse
- **Veri Saklama:** Java Binary Serialization (`.bin` dosyaları: *urunler.bin, raflar.bin, mudur.bin*)
- **Versiyon Kontrol:** Git & GitHub

---

3. ANALİZ VE TASARIM RAPORLARI İLE UYUM

Kodlama süreci, önceki raporlarda belirtilen gereksinimlere (FR ve NFR) birebir uyumlu şekilde tamamlanmıştır.

3.1 Fonksiyonel Gereksinimlerin Karşılanması

Gereksinim	Kod Karşılığı (Implementasyon)	Durum
Güvenli Giriş	MudurManager sınıfında kullanıcı adı/şifre kontrolü ve döngüsel hak mekanizması (3 yanlış giriş limiti).	**Tamamlandı**
Parçalı Stok Dağıtımı	UrunManager.urunKayitEkle metodunda, kapasite yetmediğinde devreye giren while döngüsü ve "Evet/Hayır" teyit mekanizması.	**Tamamlandı**
Veri Tutarlılığı (Raf Silme)	RafManager.rafCikarma metodunun, silmeden önce urunManager.rafDoluMu(index) çağrısı yaparak dolu rafların silinmesini engellemesi.	**Tamamlandı**
Ürün Takibi	Urun.java sınıfında tek bir raf indexi yerine List<Integer> rafListesi kullanılarak ürünün birden çok raftaki izinin sürülmesi.	**Tamamlandı**

3.2 Tasarım Uyumu

- **MenuController** yalnızca kullanıcı etkileşiminden (menü gösterimi, girdi alma, yönlendirme) sorumludur → Sunum katmanı tam uyumlu.
- **RentalService** tüm iş kurallarını (doğrulama, stok kontrolü, ücret hesaplama) merkezi olarak yönetir → Façade benzeri davranış sergiler.
- **FileManager** Singleton olarak uygulanmış olup veri bütünlüğünü tek noktadan sağlar → Veri katmanı tam uyumlu.
- **Item, Customer, Rental** sınıfları analizdeki alan modeline birebir karşılık gelir.
- Katmanlı mimari (UI → Service → Data) bağımlılık yönüyle doğru şekilde kurulmuştur.

Tasarım raporunda belirtilen tüm sınıflar ve ilişkiler kodda eksiksiz yer almaktadır.

4. GERÇEKLEŞTİRİLEN SİSTEM ÖZELLİKLERİ

Sistem, analiz aşamasında belirlenen temel gereksinimlerin ötesine geçerek, operasyonel esneklik ve veri güvenliğini sağlayan gelişmiş özelliklerle donatılmıştır. Aşağıda bu özelliklerin teknik detayları ve kod seviyesindeki karşılıkları açıklanmıştır.

4.1 Akıllı ve Parçalı Stok Dağıtım Algoritması (Smart Stock Distribution)

Sistemin en karmaşık ve ayırt edici özelliğidir. `UrunManager.java` sınıfında `urunKayitEkle` ve `urunEkleme` metodları içerisinde implemente edilen bu algoritma, "Hepsi ya da Hiçbiri" (All-or-Nothing) yaklaşımı yerine esnek yerleşim modelini benimser.

- **Algoritma Akışı:**
    1. **Kapasite Kontrolü: Kullanıcı bir rafa ürün eklemek istediğinde, sistem önce `rafManager.getRafKapasitesi(index)` ile anlık doluluk durumunu sorgular.**
    2. **Karar Mekanizması: Eğer hedef rafın kapasitesi yetersizse, sistem işlemi iptal etmek yerine kullanıcıya etkileşimli bir çözüm sunar: *"Kalan miktar X adet. Diğer raflara dağıtmak ister misiniz? (E/H)"*.**
    3. **İteratif Dağıtım Döngüsü: Kullanıcı onay verdiğinde `while(kalanDagitilacak > 0)` döngüsü başlar. Kullanıcı her adımda yeni bir raf seçer, sistem o rafın kapasitesini kontrol eder ve uygun miktarı yerleştirir.**
    4. **Mantıksal Bağlantı (1-to-N İlişki): Fiziksel olarak farklı raflara dağılan ürünler, yazılım tarafında tek bir `Urun` nesnesi olarak tutulur. `Urun.java` sınıfına eklenen `List<Integer> rafListesi` yapısı sayesinde, bir ürün nesnesi bulunduğu tüm rafların ID'lerini dinamik bir liste halinde saklar. Bu**

sayede kullanıcı `Listeleme` işlemi yaptığında, ürünün tek bir yerde değil, örneğin "Raf 1, Raf 3, Raf 5"te bulunduğu bilgisini toplu olarak görür.

4.2 Referans Bütünlüğü ve Güvenli Raf Yönetimi (Referential Integrity)

Veri tabanı sistemlerinde bulunan "Foreign Key Constraint" mantığı, dosya tabanlı bu sistemde kod seviyesinde simüle edilmiştir. `RafManager.java` sınıfı, raf silme işlemlerinde `UrunManager` sınıfına bağımlı (Dependency) çalışır.

- Çapraz Kontrol (Cross-Check): `rafCikarma` fonksiyonu çalıştırıldığında, silinmek istenen rafın ID'si `urunManager.rafDoluMu(index)` metoduna gönderilir.
- Silme Koruması: `UrunManager` tüm ürün listesini tarar; eğer belirtilen rafta (veya o rafın bulunduğu `rafListesi` içinde) tek bir ürün bile varsa `true` döner. Bu durumda `RafManager` silme işlemini bloklar ve kullanıcıya *"HATA: Bu raf doludur, önce boşaltınız!"* uyarısını verir. Bu mekanizma, ürünlerin "rafsız" kalmasını (Orphan Data) ve envanter tutarsızlığını kesin olarak engeller.
- İndeks Kaydırma: Bir raf başarıyla silindiğinde, o raftan sonra gelen tüm rafların indeksleri değişeceğinden, sistem `urunManager.rafSilindiktenSonraGuncelle` metodunu tetikleyerek ürünlerin üzerindeki raf etiketlerini de otomatik olarak günceller.

4.3 Girdi Doğrulama ve Hata Toleransı (Input Validation & Robustness)

Kullanıcı deneyimini (UX) iyileştirmek ve çalışma zamanı hatalarını (Runtime Errors) önlemek için tüm veri giriş noktaları koruma altına alınmıştır.

- Tip Güvenliği: `Manager` sınıflarında merkezi olarak kullanılan `sayiIstegi` yardımcı metodu, `Scanner` girişlerini `try-catch` benzeri bir mantıkla denetler. Kullanıcı sayısal bir alana harf veya sembol girdiğinde, sistem `scanner.hasNextInt()` kontrolü ile bunu yakalar, `InputMismatchException` hatasını engeller, tampon belleği (`scanner.next()`) temizler ve kullanıcıdan tekrar giriş yapmasını ister.

- **Negatif Değer Kontrolü: Stok miktarı, raf kapasitesi gibi değerlerin negatif girilmesi `while` döngüleri ile engellenmiş, mantıksal hataların önüne geçilmiştir.**

4.4 Generic Veri Erişim Katmanı (Generic Data Access)

Kod tekrarını önlemek (DRY Prensibi) ve genişletilebilirliği artırmak amacıyla veri erişim katmanı jenerik yapıda tasarlanmıştır.

- **Tek Merkezli Yönetim: `DosyaIslemleri.java` sınıfı `<T>` parametresi ile çalışır. Bu sayede `Urun`, `Raf` veya `Mudur` nesneleri için ayrı ayrı okuma/yazma metodları yazılmasına gerek kalmamıştır.**
- **Binary Serialization: Veriler metin dosyası yerine Java'nın yerel `.bin` formatında (`ObjectOutputStream`) saklanır. Bu yöntem, verilerin dışarıdan müdahale ile bozulmasını zorlaştırır ve nesne bütünlüğünü (Object Graph) koruyarak kaydeder.**

4.5 Kademeli Yetkilendirme Sistemi

Sistem güvenliği, `MudurManager` sınıfındaki "3 Hak Kuralı" ile sağlanmıştır.

- **Giriş algoritması, hatalı şifre denemelerini sayar. 3 kez hatalı giriş yapıldığında `System.exit(0)` komutu ile program kendini kapatır. Bu, "Brute Force" (Kaba Kuvvet) denemelerine karşı basit ama etkili bir önlemdir.**

5. KULLANICI KILAVUZU

5.1 Kurulum ve Başlatma

Sistem veritabanı kurulumu gerektirmez. `Main.java` derlenip çalıştırıldığında, eğer daha önce oluşturulmuş `.bin` dosyaları yoksa sistem otomatik olarak "İlk Kurulum Modu"na geçer ve yönetici kaydı oluşturmanızı ister.

5.2 Ana Menü İşlevleri

Uygulama açıldığında kullanıcıyı karşılayan menü seçenekleri:

1. **Yeni Ürün Kaydı Ekleme:** Depoda hiç olmayan yeni bir ürün (Seri No bazlı) oluşturur. Parçalı dağıtım burada devreye girer.
2. **Ürün Ekleme:** Mevcut bir ürünün (Seri No ile bulunur) stoğunu artırır.
3. **Ürün Çıkarma:** Stok düşer veya stok 0 olursa ürün tamamen silinir.
4. **Ürün Taşıma:** Bir ürünü Raf A'dan Raf B'ye taşır; kapasite kontrolü yapılır.
5. **İstenilen Ürün Bilgisi:** Seri no ile arama yapar.
6. **Tüm Liste:** Depodaki her şeyi listeler.
7. **Raf Ekleme:** Depo kapasitesini artırır.
8. **Raf Çıkarma:** Boş rafları siler.
9. **Çıkış:** Programı güvenli kapatır.

5.3 Örnek Senaryo

- **Senaryo:** Raf 1'in kapasitesi 50. Kullanıcı 150 adet "Kalem" eklemek istiyor.
- **İşlem:** Sistem 50 adedi Raf 1'e koyar. Kullanıcıya "Kalan 100 adedi dağıtmak ister misiniz?" diye sorar. Kullanıcı Raf 2 ve Raf 3'ü seçerek işlemi tamamlar. Listelemede Kalem ürünü için "Raflar: 1, 2, 3" görünür.

---

6. TASARIMDAN SAPMALAR VE ALINAN KARARLAR

Gerçekleştirim aşamasında verimliliği artırmak için yapılan teknik tercihler:

1. **Transient Scanner:** Sınıfların `Serializable` olması gerekliliği nedeniyle, `Scanner` nesneleri `transient` olarak işaretlenmiştir. Bu sayede serileştirme hatası (`NotSerializableException`) alınmadan nesneler diske kaydedilebilmiştir.
2. **Generic Dosya İşlemleri:** Tasarımda her sınıf için ayrı dosya yöneticisi düşünülmüşken, kodlamada `DosyaIslemleri` sınıfı generic (`<T>`) yapılarak tüm dosya operasyonları tek bir sınıfta merkezileştirilmiştir (DRY Prensibi).
3. **Index Yönetimi:** Kullanıcı arayüzünde "1. Raf" olarak görünen veriler, kod arka planında `index = girilen - 1` mantığıyla (0 tabanlı index) işlenerek kullanıcı deneyimi ile programlama mantığı arasındaki fark soyutlanmıştır.

---

7. KARŞILAŞILAN ZORLUKLAR VE ÇÖZÜMLER

- **Zorluk:** Parçalı ürün dağıtımında, işlemin yarısında kullanıcının vazgeçmesi veya hata yapması durumunda veri tutarsızlığı oluşması riski.
    - **Çözüm:** `UrunManager` içinde geçici bir liste (`gerceklesenRaflar`) tutuldu. Dağıtım işlemi ancak tamamen başarıyla biterse ana `urunler` listesine ve dosyalara kayıt yapıldı (Transactional yaklaşım).

- **Zorluk:** Silinen rafın indexlerinin kayması sonucu, ürünlerin yanlış raflarda görünmesi.
    - **Çözüm:** `Urun` sınıfına `rafIndexleriniKaydir` ve `UrunManager` sınıfına `rafSilindiktenSonraGuncelle` metodları eklendi. Bir raf silindiğinde, ondan sonra gelen tüm ürünlerin raf referansları (`rafIndex--`) otomatik güncellendi.
- **Zorluk:** `ObjectInputStream` ile dosya okurken dosyanın boş olması durumunda alınan `EOFException`.
    - **Çözüm:** `DosyaIslemleri.yukle` metodunda `try-catch` bloğu ile bu hata yakalanarak `null` döndürülmesi sağlandı ve Manager sınıflarında `null` kontrolü ile yeni liste oluşturuldu.

---

8. EKSİKLER VE GELECEKTEKİ GELİŞTİRME ÖNERİLERİ

Proje mevcut haliyle temel depo ihtiyaçlarını (CRUD, Stok Takibi, Güvenlik) karşılamaktadır. İleriye dönük geliştirme önerileri:

- **Grafik Arayüz (GUI):** JavaFX entegrasyonu ile konsol yerine görsel butonlar kullanılabilir.
- **Loglama:** `MudurManager` işlemlerinin (kim, ne zaman girdi) bir log dosyasına (`log.txt`) kaydedilmesi.
- **Barkod Okuma:** `Scanner` yerine fiziksel barkod okuyucu inputu entegre edilebilir.

---

9. SONUÇ

Bu proje ile, analiz raporunda hedeflenen "Esnek ve Güvenli Depo Yönetimi" sistemi başarıyla hayata geçirilmiştir. Java'nın nesne yönelimli yetenekleri (Kapsülleme, Kalıtım) ve dosya işleme mekanizmaları etkin kullanılarak; veri kaybına dayanıklı, kullanıcı hatalarını tolere eden (`Try-Catch` blokları) ve karmaşık stok senaryolarını (`Parçalı Dağıtım`) yönetebilen bir yazılım ortaya konmuştur.

Grup üyeleri, yazılım inşa sürecinin analizden kodlamaya kadar olan tüm aşamalarını deneyimlemiş ve katmanlı mimari prensiplerini pratiğe dökmüştür.