Online Kargo Takip Sistemi Dokümantasyonu

1. Giriş

Bu proje, kargo takip sisteminin temel işlevlerini (müşteri yönetimi, kargo rotalama, kargo önceliklendirme, gönderi sorgulama vb.) gerçekleştiren bir konsol tabanlı uygulamadır. Projede birçok veri yapısı bir arada kullanılmıştır:

- Müşteri listesi için **Linked List**
- Son 5 gönderi sorgusu için **Stack**
- Kargo teslim önceliği için **Priority Queue**
- Kargo dağıtım şehirlerini temsil etmek için **Tree**
- Kargo sorgulamalarında **Sorting** ve **Searching**

Bu sayede farklı veri yapılarının nasıl entegre edilebileceğini ve algoritmaların gerçek bir problem üzerinde nasıl kullanılabileceğini göstermeyi amaçladık.

2. Proje Mimarisi ve Veri Yapılarının Seçimi

2.1. Müşteri Verilerinin Yönetimi (Linked List)

- **Neden Linked List?**
  Müşterilerin eklenmesi/silinmesi vb. işlemlerde dinamik yapı sunması ve ekleme sırasında (özellikle son ekleme) klasik bir dizi yapısına nazaran kolaylık sağlaması için tercih edilmiştir.
- **Kullanım:**
  - `CustomerList` sınıfı, `CustomerNode` düğümleri üzerinden tek bağlı liste (singly linked list) olarak tasarlandı.
  - Her düğümde bir `Customer` nesnesi tutulur.

2.2. Gönderi Geçmişi (Linked List) ve Son 5 Gönderi (Stack)

- **Gönderi Geçmişi (Linked List)**

    - Müşterinin kargo gönderi kayıtları `shipmentHead` referansı üzerinden bir bağlı listeyle izlenir.
    - Tarih sırasına göre ekleme (sorted insert) yapılır.
- **Son 5 Gönderi (Stack)**
    - Son beş gönderiye hızlı erişim gerektiren proje maddesi için stack (yığın) seçildi. LIFO prensibi sayesinde en son eklenen kargoları kolayca görüntüleyebiliyoruz.
    - `Stack<Shipment>` yapısında 5 eleman sınırını aştığımızda en eski gönderiyi çıkarıp yerine yeniyi ekleme işlemi yapıldı.

2.3. Kargo Önceliklendirme (Priority Queue)

- **Neden Priority Queue?**
  Projede "teslimat süresi" düşük olan kargoların öncelikli işleme alınması isteniyor. Priority Queue, bu öncelik sıralamasını otomatik olarak yönetir.
- **Kullanım:**
    - `PriorityQueue<Shipment>` Java'da doğal olarak desteklenir.
    - `ShipmentComparator` sınıfı, `Shipment` nesnelerinin `deliveryTime` alanına göre karşılaştırma yapar.
    - Uygulamada 8. menü seçeneği "Kargo öncelik kuyruğuna ekle ve en öncelikli kargoyu işleme al" şeklindedir.

2.4. Kargo Rotalama (Tree)

- **Neden Tree?**
  Şehirlerin hiyerarşik bir yapıda (merkez -> alt şehirler -> onların alt şehirleri) temsil edilmesi için ağaç mantığı uygundur.
- **Kullanım:**
    - `RouteTree` sınıfı, kök düğüm olarak İstanbul (IST01) merkezine sahiptir.
    - Her `RouteNode` bir şehir bilgisi (ad, ID) ve alt şehirleri (`children`) tutar.
    - Derinlik (depth) hesaplamasıyla "teslimat süresi" otomatik belirlenir. Örneğin derinlik 2 ise 2 gün.

2.5. Sorting & Searching

- **Neden Merge Sort ve Binary Search?**
    1. **Teslim Edilmemiş Kargolar**: Teslim süresine göre sıralamada proje gereği Merge Sort veya Quick Sort kullanmamız isteniyor. Biz `mergeSort` fonksiyonunu tercih ettik.

2. **Teslim Edilmiş Kargolar**: Kargo ID'si bazında `binarySearchByID` yapıyoruz. `Binary Search`'ün çalışabilmesi için önce listeyi kargo ID'sine göre sıralıyoruz. Bu da **O(m log m)** karmaşıklığa sahip oluyor.

3. Zaman ve Uzay Karmaşıklığı Analizi

Aşağıda, projedeki temel veri yapısı ve algoritmalara dair özet karmaşıklık analizleri yer almaktadır:

İşlem	Zaman Karmaşıklığı	Uzay Karmaşıklığı
CustomerList.addCustomer() (LinkedList sonuna ek)	O(n)	O(1)
CustomerList.findCustomerByID()	O(n)	O(1)
Customer.addShipmentSorted() (LinkedList)	O(n)	O(1)
Stack push/pop (Son 5 gönderi)	O(1)	O(1) (sabit boyut stack)
PriorityQueue.offer()	O(log n)	O(n) (içerideki eleman sayısına göre)
PriorityQueue.poll()	O(log n)	O(n)
RouteTree.getCityDepth() (DFS)	O(k) (tüm ağaç düğümleri)	O(h) (derinlik kadar)
Merge Sort	O(k log k)	O(k)
Binary Search	O(log k)	O(1)

Not: Tabloya eklenen `n`, `k` değerleri ilgili koleksiyondaki elemanları ifade etmektedir (örneğin, `n` = toplam müşteri sayısı, `k` = kargo sayısı gibi).

4. Kod Yapısı (Özet)

1. **Shipment**: Gönderinin ID, tarih, teslim durumu, teslim süresi, hedef şehir bilgilerini içerir.
2. **Customer**: Müşteri ID, isim ve gönderi listesi (LinkedList) ile son 5 gönderi (Stack).
3. **CustomerList**: Tüm müşterileri bir LinkedList yapısında tutar. Ekleme, bulma gibi metodlar içerir.
4. **PriorityQueue**: `shipmentPQ` ile teslim süresi düşük olan kargoyu önceliklendirir.

5. **RouteTree**: Şehirlerin ağaç yapısında tutulduğu sınıftır. Derinlik hesabı ve ağaç yazdırma metodları içerir.
6. **SortAndSearch**: Merge Sort ve Binary Search metodlarını barındırır.
7. **CargoSystem (main)**:
    - Menü tabanlı arayüz (konsol).
    - Kullanıcıdan girdi alarak ilgili işlemleri gerçekleştirir (müşteri ekleme, kargo ekleme, sorgulama vs.).

5. Test Senaryoları

Aşağıda örnek test senaryoları verilmiştir. Uygulamanın menüsünde sırasıyla hangi seçimlerin yapılacağı ve beklenen sonuçlar özetlenmiştir.

Senaryo 1: Yeni Müşteri Ekleme

1. **Menüde** `1` seçilir: "Yeni müşteri ekle"
2. **Girdi:**
    - Müşteri ID: `123`
    - Müşteri Ad Soyad: `Ali Can`
3. **Beklenen Çıktı:**
    - "Müşteri eklendi."

Ardından `CustomerList`'e bir düğüm eklenmiş olur.

Senaryo 2: Mevcut Müşteriye Kargo Ekleme

1. **Menüde** `2` seçilir: "Müşteriye kargo gönderimi ekle"
2. **Girdi:**
    - Müşteri ID: `123` (bir önceki eklenen müşteri)
    - Gönderi ID: `9001`
    - Tarih: `20241212` (YYYYMMDD format)
    - Teslim edildi mi?: `false`
    - Şehir adı: `Bursa` (rota ağacında tanımlı bir şehir)
3. **Beklenen Çıktı:**
    - "Kargo eklendi. Teslim süresi (otomatik): X gün." (Burada Bursa'nın derinliğine göre bir sayı yazacak)

Senaryo 3: Son 5 Gönderi Görüntüleme

1. **Menüde** `5` seçilir.
2. **Girdi:**
    - Müşteri ID: `123`
3. **Beklenen Çıktı:**
    - Eğer eklenmiş gönderi varsa, en yeni gönderi en üstte olacak şekilde listelenir.

Senaryo 4: Teslim Edilmiş Kargo Arama (Binary Search)

1. Mevcut veya yeni bir müşteriye, "teslim edildi = true" olan birkaç kargo ekleyelim.
2. **Menüde** `3` seçilir: "Kargo durumu sorgula (teslim edilmişlerde binary search)"
3. **Girdi:**
    - Aranacak Gönderi ID: `9001`
4. **Beklenen Çıktı:**
    - "Kargo bulundu: 9001 (Teslim Edildi), Şehir: …" ya da "Kargo bulunamadı"

Senaryo 5: Teslim Edilmemiş Kargoları Sıralama (Merge Sort)

1. Farklı müşterilere "delivered = false" olan farklı teslim sürelerine sahip kargolar ekle.
2. **Menüde** `6` seç: "Teslim edilmemiş kargoları teslim süresine göre sıralı listele"
3. **Beklenen Çıktı:**
    - Ekranda kargolar artan teslim süresine göre listelenir.

Senaryo 6: Kargo Öncelik Kuyruğu (Priority Queue)

1. **Menüde** `8` seç: "Kargo öncelik kuyruğuna ekle ve en öncelikli kargoyu işleme al."
2. **Girdi:**
    - Gönderi ID: `5555`
    - Şehir adı: `Tekirdag`
3. **Beklenen Çıktı:**
    - "Kargo öncelik kuyruğuna eklendi (Şehir: Tekirdag, Süre: x)."
    - "En öncelikli kargo işleniyor…"
    - (Çıkan poll sonucu) "İşlenen kargo: 5555 Süre: x Şehir: Tekirdag"

6. Sonuç ve Öneriler

- Projede **LinkedList, Stack, PriorityQueue, Tree** gibi veri yapıları başarılı şekilde entegre edilmiştir.
- Sorting ve Searching algoritmaları (Merge Sort, Binary Search) eklenerek kargo sorguları optimize edilmiştir.

- Performans analizinde görüldüğü gibi, büyük veri kümelerinde bile makul sürelerde sonuç elde etmek için uygun algoritmalar seçilmiştir.
- Geliştirme aşamasında ek özellikler (GUI, veri tabanı entegrasyonu vb.) istenirse projeye rahatça eklenebilir.

7. Kaynaklar

- Java resmi dokümantasyonu (Oracle)
- Ders notları ve proje ödevi dokümanı