**İSTANBUL MEDENİYET ÜNİVERSİTESİ**

**DOĞA BİLİMLERİ VE MÜHENDİSLİK FAKÜLTESİ**

**BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ**

**BİL201 PROJE ÖDEVİ**

**HAZIRLAYANLAR**

**HİLAL BETÜL DERELİ 23120205071**

**ZEYNEP ÖZGE TEMEL 23120205072**

**AHMET SAMİ GÜL 23120205022**

**Main:**

Ana metot, Swing GUI ile başlayan bir uygulama döngüsünü içerir.

**CargoTrackingGui:**

handleViewHistory: Belirtilen müşteri ID'sine göre gönderi geçmişini görüntüler. Müşteri bulunursa gönderi geçmişi listelenir. JTextArea ile kullanıcıya detaylı bilgi sunulur.

**Zaman Karmaşıklığı:**

* **En Kötü Durum (Worst-Case):** O(n \* m)
  + n: Müşteri sayısı (CustomerLinkedList uzunluğu).
  + m: Her müşterinin gönderi geçmişindeki gönderi sayısı.
* Tüm müşteriler ve gönderileri dolaşılır.

handleUpdateStatus: Belirtilen gönderi ID'sinin durumunu "Teslim Edildi" olarak günceller. Müşteri listesinde gezinerek uygun gönderiyi bulur ve durumu günceller.

**Zaman Karmaşıklığı:**

* **En Kötü Durum (Worst-Case):** O(n \* m)
  + n: Müşteri sayısı.
  + m: Her müşterinin gönderi geçmişindeki gönderi sayısı.
* Tüm müşteriler ve gönderileri dolaşılır.

handleAddCustomer : Yeni bir müşteri ekler ve otomatik bir ID atar. Kullanıcıdan müşteri adı ve soyadı alınır. Müşteri, CustomerLinkedList'e eklenir.

**Zaman Karmaşıklığı:**

* **En Kötü Durum (Worst-Case):** O(1)
  + Listeye yeni bir düğüm eklemek sabit zamanda gerçekleşir.

handleAddShipment :Belirtilen şehir ve ilçeye bir gönderi ekler. Müşteri ID'si doğrulanır ve gönderi müşteri gönderi geçmişine kaydedilir. Gönderi ayrıca bir yığın ve öncelik kuyruğuna eklenir. Kullanıcıdan şehir, ilçe ve müşteri ID'si bilgileri alınır. Gönderi, ShipmentStack ve PriorityQueue veri yapılarına eklenir.

**Zaman Karmaşıklığı:**

* **Şehir ve İlçe Seçimi:** O(c + d)
  + c: Şehir sayısı.
  + d: Seçilen şehrin ilçelerinin sayısı.
* **Müşteri Bulma:** O(n)
  + n: Müşteri sayısı.
* **Yığın ve Kuyruğa Ekleme:** O(log k)
  + k: Öncelik kuyruğundaki öğe sayısı.
* **Toplam:** O(c + d + n + log k)

handleSearchShipment: Belirtilen müşteri ID'sine göre teslim edilmiş gönderileri arar. Müşteri bulunursa, teslim edilmiş gönderiler bir listeye eklenir ve kullanıcıya gösterilir.

**Zaman Karmaşıklığı:**

* **En Kötü Durum (Worst-Case):** O(n \* m)
  + n: Müşteri sayısı.
  + m: Gönderi geçmişindeki öğe sayısı.

handleSortShipments: Müşteri ID'sine göre teslim edilmemiş gönderileri sıralar. Gönderiler mergeSort algoritmasıyla sıralanır ve kullanıcıya gösterilir.

**Zaman Karmaşıklığı:**

* **Arama:** O(n \* m)
  + n: Müşteri sayısı.
  + m: Gönderi geçmişindeki öğe sayısı.
* **Sıralama:** O(p log p)
  + p: Teslim edilmemiş gönderi sayısı.

handleViewShipments: Belirtilen müşteri ID'sine ait gönderileri teslimat süresine göre sıralar ve gösterir. Gönderiler öncelik kuyruğundan alınarak görüntülenir.

**Zaman Karmaşıklığı:**

* **Arama:** O(n)
* **Kuyruk Gezinme:** O(k)
  + k: Kuyruktaki gönderi sayısı.

handleViewRecentShipments: Belirtilen müşteri ID'sine ait son 5 gönderiyi listeler. Gönderiler bir yığından alınır ve kullanıcıya gösterilir.

**Zaman Karmaşıklığı:**

* **Arama:** O(n)
* **Gönderi Gezinme:** O(1)

HandleViewRoutes:Kargo teslimat rotalarını şehir ve ilçe bazında gösterir. Teslimat ağacı dolaşılır ve kullanıcıya tüm rotalar listelenir.

**Zaman Karmaşıklığı:**

* **Ağaç Gezinme:** O(t)
  + t: Ağaçtaki toplam şehir ve ilçe sayısı.

findCityInTree(City root, String cityName):Bu metot, belirli bir şehir adını tree yapımızı gezerek bulur. Kökteki şehir adı kontrol edilir. Alt şehirler rekürsif olarak gezilir. Zaman Karmaşıklığı: Worst case’imizde ağacın tüm düğümlerini ziyaret edilebileceği için O(N)’dir. Burada N ağaçtaki toplam şehir sayısıdır.

**City:**

City sınıfı, bir şehir modelini temsil eder. Her şehir, birkaç alt şehir içerebilir (alt şehirler bir liste olarak tutulur), teslimat süresi ve rota bilgisi gibi özelliklere sahip olabilir. Bu sınıf, bir şehirler arası bağlantıyı ve teslimat sürelerini modellemek için kullanılabilir.

Getter ve Setter’ların zmana karmaşası O(1)’dir.

addSubCity(City): Alt şehir eklemek için kullanılan bu metod, bir şehri subCities listesinin sonuna ekler. Java'daki ArrayList'in add() metodu O(1) zaman karmaşıklığına sahiptir, çünkü genellikle listenin sonuna ekleme yapıldığında sabit zaman alır. Ancak, eğer kapasite dolarsa ve yeniden boyutlandırma yapılması gerekirse, bu işlem O(n) karmaşıklığına sahip olabilir (bu durum nadiren gerçekleşir).

**Customer:**

Customer sınıfı, bir müşteri bilgisini temsil eder. Müşterinin kişisel bilgileri (isim, soyisim, müşteri kimliği) ve gönderi geçmişi ile ilgili bilgileri tutan bir yapıdır. Müşteri, birden fazla gönderiye sahip olabilir ve bu gönderiler, bir ShipmentLinkedList yapısı içinde saklanır. Ayrıca, son gönderiler bir Stack veri yapısında tutulur. İçindeki tüm metotların zaman karmaşası O(1)’dir.

**CustomerIDGenerator:**

Bu sınıf her müşteri için yeni bir numara oluşturur. Oluşturulan müşteri kimliği, "C" harfiyle başlar ve ardından üç haneli bir sayıya (örneğin, C001, C002 gibi) sahip olur. Zaman karmaşası O(1)’dir.

**CustomerLinkedList:**

CustomerLinkedList sınıfı, bir bağlı liste (linked list) veri yapısını kullanarak müşterileri saklar. Bu sınıf, her müşteri için benzersiz bir kimlik oluşturur ve bu kimlikleri bağlı liste şeklinde tutar. Her bir müşteri, bir CustomerNode nesnesi içinde saklanır. Müşteriler, sırasıyla bu listeye eklenir.

* **addCustomer(String firstname, String lastname):** Yeni bir müşteri oluşturur ve bu müşteriyi bağlı listeye ekler. Müşteri kimliği, CustomerIDGenerator sınıfından alınarak oluşturulur. CustomerNode'da yeni bir düğüm oluşturulur ve listeye eklenir. İlk olarak, eğer head null ise (yani liste boşsa), yeni düğüm doğrudan head'e atanır. Eğer liste boş değilse, listenin sonuna kadar gidilir ve yeni düğüm, listenin sonuna eklenir.
* **Zaman Karmaşıklığı**: **O(n)**'dir, çünkü listeye ekleme işlemi sırasında listenin sonuna ulaşmak için tüm listeyi taramak gerekmektedir. temp.next != null kontrolüyle listenin sonuna kadar gidilir. Eğer liste çok büyükse, ekleme işlemi daha uzun sürebilir.

**CustomerNode:**

Her bir müşteri için bir düğüm (node) oluşturur. Bu sınıf, her müşteri için bir Customer nesnesini saklar ve aynı zamanda bir next referansı tutar. next, bu düğümün bağlı olduğu bir sonraki CustomerNode nesnesini işaret eder. Yani, her CustomerNode, bir müşteri bilgisi ve bir bağlantı (referans) içerir. Zaman karmaşıklığı O(1)’dir.

**DeliveryTree:**

DeliveryTree sınıfı, teslimat sürelerini ve şehirler arasındaki ilişkileri yönetmek için tasarlanmış bir veri yapısıdır. Bu sınıf, bir kök şehir (root) ile başlayarak, bu şehri ve onun alt şehirlerini (subCities) bir ağaç (tree) yapısı şeklinde tutar. Her bir şehir, teslimat süresine sahip olabilir ve bu süre, ağacın derinliğine göre hesaplanır. Bu sınıf aynı zamanda ağacı ekrana yazdırabilmek için bir metot da içerir.

Getter ve setter’ların zaman karmaşıklığı o(1)’dir.

assignDeliveryTimes(), displayTree(), assignDeliveryTimesRecursive() ve displayTreeRecursive() zaman karmaşıklığı o(N)’dir çünkü her birinde n kadar şehir gezilir.

**PriorityQueue:**

Teslimat sürelerine göre kargoları sıralayan bir öncelik sırası uygulamasıdır. Bu sınıf, kargoları teslimat sürelerine göre sıralar ve her zaman teslimat süresi en kısa olan kargoyu önce çıkarır. PriorityQueue sınıfı, **min-heap** veri yapısını kullanarak en kısa teslimat süresine sahip kargoyu hızlı bir şekilde almayı sağlar.

* **ekle()**: **O(log n)** — Kargo ekleme işlemi, heap yapısının doğru şekilde sıralanmasını sağlamak için yukarı doğru yer değiştirme (heapify-up) işlemi yapar.
* **kargoBul()**: **O(n)** — heap üzerinde doğrusal arama yapar.
* **goster()**: **O(n log n)** — Kargoları sıralamak için extractMin() metodu her seferinde **O(log n)** zaman alır ve bu işlem n kez yapılır.
* **al()**: **O(log n)** — Kök eleman çıkarıldıktan sonra **Heapify-Down** işlemi yapılır.
* **heapifyDown()**: **O(log n)** — Düğümün altındaki elemanlarla karşılaştırılarak doğru konumuna yerleştirilir.

**Shipment:**

Kargo nesnesinin oluşturulduğu sınıf. Her bir kargo, bir **shipmentID**'ye, **shipmentDate**'e, **deliveryStatus**'a ve **deliveryTime**'a sahiptir. Bu sınıf, bir kargonun temel bilgilerini tutar ve kargonun teslimat süresi, durumu gibi bilgilerini yönetir. Tüm getter ve setter’ların ayrıca toString metodunun da zaman karmaşası o(1)’dir.

**ShipmentIDGenerator:**

ShipmentIDGenerator sınıfı, her yeni kargo gönderisi için benzersiz bir **shipmentID** oluşturmak amacıyla kullanılır. Bu sınıfın **static** bir **shipmentCounter** (kargo sayacı) değişkeni vardır. Her yeni kargo oluşturulduğunda, bu sayaç artırılır ve yeni bir ID oluşturulur. Her ID, kolayca okunabilir bir formatta oluşturulmakta ve **O(1)** zaman karmaşıklığı ile hızlıca üretilmektedir.

**ShipmentLinkedKist:**

ShipmentLinkedList sınıfı, **bağlantılı liste** veri yapısını kullanarak, her biri bir Shipment nesnesini temsil eden düğümlerden oluşan bir listeyi yönetir.

addShipment(): Bu metod, bir kargo nesnesini (shipment) bağlantılı listeye ekler ve kargoların teslimat tarihine göre sıralı bir şekilde listede yer almasını sağlar.

* **Çalışma Prensibi**:
  + **Yeni Kargo Eklenmesi**: İlk olarak, eklenmek istenen kargo (shipment) ile ilgili yeni bir ShipmentNode nesnesi oluşturulur.
  + **Başlangıç Durumu**: Eğer liste boşsa (head == null) ya da eklenmek istenen kargonun teslimat tarihi, listenin başındaki kargonun teslimat tarihinden önceyse (shipment.getShipmentDate().isBefore(head.shipment.getShipmentDate())), yeni kargo başa eklenir ve head göstergesi yeni düğüme yönlendirilir.
  + **Yerleştirme Durumu**: Eğer liste boş değilse ve yeni kargonun teslimat tarihi listenin başındaki kargonun teslimat tarihinden sonra ise, listeyi baştan sona kadar tararız. Her geçişte, eklenmek istenen kargonun teslimat tarihi, geçilen düğümün teslimat tarihinden önce olan bir yere gelene kadar durdurulur. Bu noktada yeni düğüm (kargo) yerleştirilir.
  + **Ekleme Konumu**: Son olarak, yeni kargo doğru konumuna yerleştirilir. Yeni düğümün next değeri, eski düğümün next değerine atanır ve eski düğümün next değeri, yeni düğüme yönlendirilir.
* **Zaman Karmaşıklığı**:
  + **Best Case** (En iyi durum - kargo başa ekleniyor): **O(1)** — Yeni kargo, başa eklenir.
  + **Worst Case** (En kötü durum - kargo sona ekleniyor): **O(n)** — Kargo listenin sonuna eklenirse, tüm listeyi gezmek gerekir.
  + **Ortalama Durum**: **O(n)** — Kargo, çoğu durumda listenin ortasına veya sonuna eklenir.

**ShipmentNode:**

ShipmentNode sınıfı, bağlantılı liste veri yapısında kullanılan bir düğüm sınıfıdır. Bu sınıf, her bir kargo nesnesini ve bir sonraki düğüme referans tutar. ShipmentNode sınıfı, ShipmentLinkedList gibi bağlantılı listelerde verilerin sıralı bir şekilde tutulmasına yardımcı olur.

**ShipmentSearch:**

ShipmentSearch sınıfı, ikili arama algoritması kullanarak, bir liste içerisindeki belirli bir kargonun arama işlemini gerçekleştirir. Zaman Karmaşıklığı O(log n)’dir. İkili arama algoritması, her adımda arama alanını ikiye böler. Bu nedenle, algoritmanın çalışma süresi logaritmik olur.

**ShipmentSort:**

ShipmentSort sınıfı, Merge Sort algoritmasını kullanarak kargo (shipment) nesnelerini teslimat sürelerine göre sıralamak için tasarlanmış bir sınıftır. Bu sınıfın ana amacı, List<Shipment> türündeki bir listeyi sıralamaktır. Zaman karmaşıklığı, **O(n log n)**'dir.

**Stack:**

Stack sınıfı, Kargo nesnelerini tutmak için kullanılan yığın veri yapısını temsil eder. Bu sınıf, kargo gönderilerini eklemek, çıkarmak, yığındaki öğeleri görüntülemek ve yığının büyüklüğünü öğrenmek gibi temel yığın işlevlerini içerir.

* **push(Shipment shipment)**: **O(n)** (yığın dolu olduğunda, en eski öğeyi bulmak için n öğe taranır)
* **pop()**: **O(1)** (üst öğe çıkarılır, sabit zaman alır)
* **display()**: **O(n)** (tüm öğeler sırasıyla yazdırılır)
* **isEmpty()**: **O(1)** (sadece size kontrol edilir)
* **isFull()**: **O(1)** (sadece size kontrol edilir)
* **size()**: **O(1)** (sadece size değeri döndürülür)