İZMİT ROTA PLANLAMA SİSTEMİ

Sümeyye Nur Altun
230201032
Bilgisayar Mühendisliği
Kocaeli Üniversitesi

Burak Demir
230201030
Bilgisayar Mühendisliği
Kocaeli Üniversitesi

I. OZET

Bu rapor, "İzmit Rota Planlama Sistemi" isimli Programlama Laboratuvarı 2 dersinin 1. projesini açıklamak ve sunumunu gerçekleştirmek amacıyla hazırlanmıştır. Raporda, projenin özeti, giriş kısmı, kullanılan yöntemler, deneysel sonuçlar, sonuçlar, yazar katkıları ve kaynakça yer almaktadır. Proje geliştirilirken, Java programlama dili, Java'nın Swing, GraphStream, JXMapViewer, Jackson, JIconFont gibi kütüphaneleri kullanılmıştır.

II. GİRİŞ

Bu projede, çözülmesi gereken dört ana problem bulunmaktadır. İlk problem, veri setinden duraklara ait bilgileri çekmek ve bu verilerle bir graph oluşturmaktır. İkinci problem, kullanıcının belirttiği konum bilgisine göre, gidebileceği en uygun rotaları hesaplamaktır. Üçüncü problem, projenin gereksinimlerine uygun olarak sınıf yapılarını nesne yönelimli mimariye ve SOLID prensiplerine göre tasarlamaktır. Dördüncü ve son problem ise, kullanıcı dostu bir arayüz geliştirerek, uygulamanın kullanımını kolay ve verimli hale getirmektir.

III. YÖNTEMLER

1. Problem İçin Çözüm Önerisi

parseVeriSeti() sınıfında, JSON formatındaki veri seti **Jackson** kütüphanesi kullanılarak işlenir. Bu işlem sırasında her durağa ait koordinatlar, aktarma noktaları (transferler) ve diğer gerekli bilgile**r** çıkarılır. Daha sonra, **VehicleFactory** aracılığıyla uygun taşıt nesneleri oluşturulur. Duraklardan gelen bilgiler kullanılarak **GraphStream** kütüphanesi ile bir grafik görselleştirilir. Bu grafikte, **duraklar düğüm** olarak, sonraki duraklarla olan bağlantılar ise kenar olarak eklenir. Ayrıca, mesafe, süre ve ücret bilgileri kenarlara atanarak grafiğe dahil edilir.

2. Problem İçin Çözüm Önerisi

GraphStream grafiğindeki düğümler (duraklar) ve kenarlar (bağlantılar) kullanılarak yollar bulunuyor. findAllRoutes() başlangıç ve bitiş noktaları arasındaki tüm yolları bulur. findShortestRoute() ile en kısa yol bulunur. Dijkstra mantığıyla en kısa mesafeye sahip yolu seçer.

2. Problem İçin Çözüm Önerisi

Vehicle ve Passenger soyut sınıflarıyla genişletilebilir bir yapı sunarken, her sınıf tek bir sorumluluğu yerine getirir. Örneğin RouteFinder sadece rota bulma işlemlerinden sorumludur. Alt sınıflar (Bus, Student gibi) üst sınıfların davranışlarını bozmaz, arayüzler ihtiyaca göre ayrılmıştır. Vehicle sadece durak bilgisi döndürürken, OdemeYontemi sadece ödeme işlemlerini yönetir. Bağımlılıklar soyut katmanlar üzerinden yönetilerek VehicleFactory gibi yapılar sayesinde sistem esnek ve sürdürülebilir hale getirilmiştir.

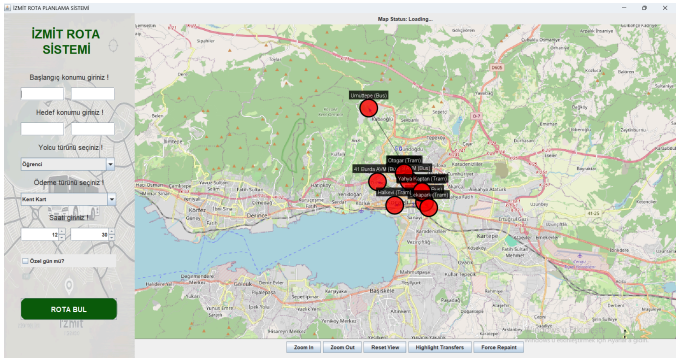
4. Problem İçin Çözüm Önerisi

Bu problemin çözümü için rota bilgisi ekranına üç farklı araç için üç buton ekledik. Böylece, araçlar kategorize edilerek kullanıcıya daha kolay bir seçim yapma imkanı sağlanıyor. Aynı zamanda, rota bilgisi ekranında kullanıcıya en az transfer, en kısa mesafe ve en kısa sürede ulaşabileceği rotaları göstererek onları bilgilendiriyoruz. Toplu taşıma kısmında ise sadece otobüs veya sadece tramvay vb.

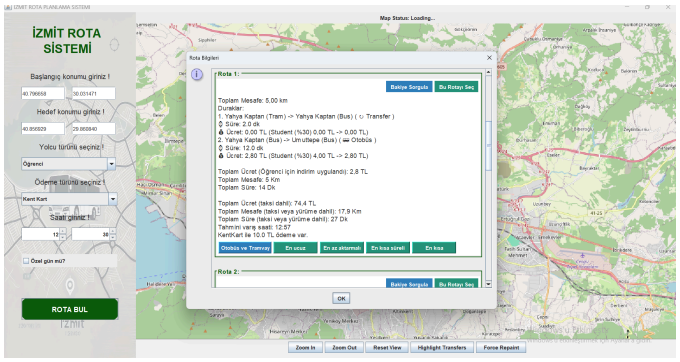
ile gidilebilen rotaları belirleyerek, kullanıcının seçebileceği alternatif rotaların tamamını sunuyoruz.

IV. SONUÇ

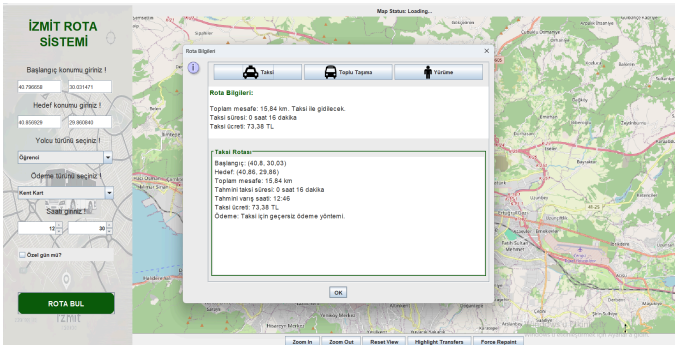
Tüm bu işlemler sonucunda, veri setindeki durak bilgilerini çekerek bir graph oluşturmayı, bu graphı harita üzerine yerleştirerek kullanıcıya daha estetik bir arayüz sunmayı, gidilebilecek rotaları göstermeyi ve sınıf yapılarını nesne yönelimli mimari ve SOLID prensiplerine uygun bir şekilde tasarlamayı başarıyla gerçekleştirdik.



Harita ve Graphtan oluşan arayüz. Graphtaki düğümler durakları göstermektedir.



Kullanıcının girdiği enlem ve boylam bilgisine göre oluşturan rotalar ve rotalarla ilgili bilgiler.



Farklı ulaşım seçenekleri.

V. YAZAR KATKILARI

Projemizin büyük bir kısmını ekip çalışmasıyla yürüterek her iki ekip üyesi de projenin her aşamasında görev almış ve düzeltmeler yapmıştır. Tüm kısımlarda her iki kişinin görüşleri dikkate alınmış ve her iki taraf da eşit katkı sağlamıştır. Yapılamayan kısımlarda birbirimize yardımcı olarak projeyi başarıyla tamamladık.

Burak Demir Katkıları

Proje kapsamında, veri setinden verileri çekme, çekilen verilere göre graph oluşturma, bu graphı harita üzerine yerleştirme ve kullanıcının girdiği konum bilgisine göre gidilebilecek rotaları bulma gibi backend işlemleriyle ilgilenmiştir. Bu alanlarda önemli katkılar sağlamıştır.

Sümeyye Nur Altun Katkıları

Projenin ön yüz tasarımında görev alarak kullanıcı dostu bir arayüz oluşturmuştur. Ayrıca, sınıf yapılarını nesne yönelimli mimari ve SOLID prensiplerine uygun olarak oluşturmuş ve kullanıcının en az transfer, en kısa mesafe gibi bilgileri görmesini sağlayacak düzenlemeleri yapmıştır. Toplu taşıma kullanılan rotalar için özel düzenlemeler de gerçekleştirerek, kullanıcıya çeşitli alternatif rotaları göstermiştir.

VI. KAYNAKLAR

- HTTPS://WWW.GENCAYYILDIZ.COM/BLOG/C-FACTORY-METHOD-DESIGN-PATTERNFACTORY-METHOD-TASARIM-DESENI/

- https://www.gencayyildiz.com/blog/tag/solid/
- https://youtu.be/ZJLCIePsnVc?si=qxXb5UzxXHalTE2A
- https://www.youtube.com/watch?v=_jDNAf3CzeY

VII. KABA KOD

1. Başla
2. OdemeYontemi, Passenger, Vehicle gibi classları ve onların alt sınıflarını oluştur.
3. parseVeriSeti class'ını kullanarak veri setindeki verileri önceden oluşturulan durak class'larının örneklerinin barındırıldığı bir listede tut.
4. GraphManager class'ını kullanarak üstteki durak class'larının barındırıldığı listeyi kullanarak graph oluşturma.
5. Swing ile arayüzü oluştur.
6. Kullanıcının girdiği konuma göre RouteFinder classında gidilebilecek rotaları bul. Daha sonrasında en ideal rotaları listele.
7. Kullanılan rotalara ve kullanıcının girdiği bilgilere göre nesneler oluştur.
8. Kullanıcı başlangıç ve bitiş noktalarını girer.
   En yakın duraklar bulunur.
   RouteFinder ile tüm olası rotalar ve en kısa yol hesaplanır
   Seçilen yolcu türüne göre PassengerCreator ile yolcu nesnesi oluşturulur.
9. Sonuç Gösterimi:
   Rota bilgileri Dialog penceresi ile gösterilir
   Taksi, Toplu Taşıma ve Yürüme butonları ile farklı ulaşım seçenekleri sunulur
   Tüm olası rotalar listelenir
   En ucuz, en kısa, en az aktarmalı rotalar vurgulanır
   Seçilen ödeme yöntemine göre ücret hesaplanır. Özel günlerde ve indirimli yolcu tiplerinde fiyatlandırma gösterilir
10. Rota Seçimi:
    Kullanıcı istediği rotayı "Bu Rotayı Seç" butonu ile seçebilir
    Seçilen rota harita üzerinde yeşil renkle vurgulanır.
    Aktarma noktaları kırmızı renkle gösterilir.

VIII. Yazılım Mimarisi ve SOLID Prensipleri

Projemizin yazılım mimarisi, modern yazılım geliştirme pratiği olan SOLID prensiplerini temel alarak tasarlanmıştır. Aşağıda bu prensiplerin projemize nasıl uygulandığını görebilirsiniz:

S - Single Responsibility (Tek Sorumluluk Prensibi)
Her sınıf belirli bir görevi yerine getirerek, karmaşıklığı azaltmaktadır:
- **MapVisualizer**: Kullanıcı arayüzünü ve harita görselleştirmesini yönetir.
- **RouteFinde**r: Düğümler arasındaki olası rotaları hesaplar ve HashMap kullanarak önbelleğe alır.
- **PassengerCreator**: Yolcu nesnelerinin dinamik oluşturulması için factory pattern uygular.
- **calculateRouteAttributes**: Rotaların mesafe, fiyat ve süre hesaplamalarını yapar.
- **BiletHesapla**: Yolcu tipine göre bilet fiyatlandırmasını hesaplar.

O - Open/Closed (Açık/Kapalı Prensibi)
Kod yapısı değişiklik gerektirmeden genişletilebilir şekilde tasarlanmıştır:
- **Vehicle** abstract sınıfı, yeni araç tipleri eklemek için temel oluşturur (örn. Bus, Tram).
- **Passenger** abstract sınıfı, farklı yolcu türleri (Öğrenci, Yaşlı, Engelli, Genel) için genişletilebilir yapı sağlar.
- **OdemeYontemi** soyut sınıfı, KentKart, Nakit ve KrediKartı gibi farklı ödeme yöntemlerinin eklenmesine olanak tanır.

L - Liskov Substitution (Yerine Geçme Prensibi)
Alt sınıflar, üst sınıfların davranışlarını bozmadan genişletir:
- **Bus** ve **Tram** sınıfları, **Vehicle** sınıfını genişleterek ulaşım araçlarının davranışlarını korur.
- **Student**, **Elderly**, **Disabled** ve **General** sınıfları, **Passenger** sınıfının getIndirimOrani() metodunu tutarlı şekilde uygular.

- **KentKart**, **Nakit** ve **KrediKarti** sınıfları, **OdemeYontemi** sınıfının odemeYap() davranışını korur.

I - Interface Segregation (Arayüz Ayrımı Prensibi)
Sınıflar, yalnızca ihtiyaç duydukları metotları içerir:
- **Vehicle** sınıfı, araçlar için yalnızca gerekli olan getNextStops() ve getTransfer() metotlarını tanımlar.
- **Passenger** sınıfı, yolcular için gerekli olan getIndirimOrani() metodunu tanımlar.
- **OdemeYontemi** sınıfı, ödeme yöntemleri için sadece odemeYap() metodu gerektirir.

D - Dependency Inversion (Bağımlılığı Tersine Çevirme Prensibi)
Yüksek seviyeli modüller, düşük seviyeli modüllere doğrudan bağımlı değildir:
- **MapVisualizer** sınıfı, Graph yapısını kullanırken doğrudan implementasyona değil, arayüze bağlıdır.
- **RouteFinder** sınıfı, rota hesaplama algoritmasını soyutlar ve grafiği parametre olarak alır.
- **formatRouteInfo** sınıfı, farklı yolcu türlerini Passenger arayüzü üzerinden kullanır.

Veri Yapılarının Stratejik Kullanımı

Projemiz, veri yapılarını verimli kullanarak SOLID prensiplerine bağlılığı güçlendirmektedir:

- **HashMap**: PassengerCreator sınıfında, yolcu türlerini dinamik olarak yönetmek için kullanılır. Bu, yeni yolcu tipleri eklendiğinde kodun değiştirilmesine gerek kalmadan sisteme entegre edilebilmesini sağlar:

```
private static final Map<String,
    Class<? extends Passenger>> passengerTypes = new HashMap<>();
static {
    passengerTypes.put("Genel", General.class);
    passengerTypes.put("Öğrenci", Student.class);
    // Diğer yolcu tipleri kolayca eklenebilir
}
```

```java
 private static final Map<String, Class<? extends Passenger>> passengerTypes = new HashMap<>();
  static {
     passengerTypes.put("Genel", General.class);
     passengerTypes.put("Öğrenci", Student.class);
     // Diğer yolcu tipleri kolayca eklenebilir
  }
```

- **HashSet**: returnRouteVehicleInfo metodunda, rota boyunca kullanılan araç tiplerini takip etmek için kullanılır. Bu, özellikle karmaşık rotalarda tekrarlanan araç tiplerinin filtrelenmesini sağlar:

```
Set<Class<?>> routeVehicleInfo = new HashSet<>();
for (Node durak : Route) {
    Object vehicle = durak.getAttribute("Vehicle");
    if (vehicle != null) {
        routeVehicleInfo.add(vehicle.getClass());
    }
}
```

```java
      Set<Class<?>> routeVehicleInfo = new HashSet<>();
  for (Node durak : Route) {
     Object vehicle = durak.getAttribute("Vehicle");
     if (vehicle != null) {
        routeVehicleInfo.add(vehicle.getClass());
     }
  }
```

- **Map ve List kombinasyonu**: RouteFinder sınıfında önbellek mekanizması için kullanılır, bu da performansı önemli ölçüde artırır:

```
private final Map<String, List<List<Node>>> allRoutesCache = new HashMap<>();
private final Map<String, List<Node>> shortestRoutesCache = new HashMap<>();
```

```java
    private final Map<String, List<List<Node>>> allRoutesCache = new HashMap<>();
    private final Map<String, List<Node>> shortestRoutesCache = new HashMap<>();
```

- **PriorityQueue**: RouteFinder sınıfında, Dijkstra algoritması uygulanırken rotaları mesafeye göre sıralamak için kullanılır:

```
PriorityQueue<PathDistance> queue = new PriorityQueue<>();
queue.add(new PathDistance(initialPath, 0.0));
```

```java
PriorityQueue<PathDistance> queue = new PriorityQueue<>();
queue.add(new PathDistance(initialPath, 0.0));
```

Bu veri yapılarının akıllıca kullanımı, sistemin işlevselliğini bozmadan genişletilebilmesine ve optimize edilmiş performans sunmasına olanak tanır. Özellikle HashMap kullanımı, farklı bileşenler arasındaki bağlantıyı azaltır ve yeni özelliklerin eklenmesi sırasında mevcut kodun değiştirilmesine olan ihtiyacı en aza indirir.

Sonuç olarak, projemizdeki mimari yaklaşım ve veri yapılarının etkin kullanımı, bakımı kolay, esnek ve genişletilebilir bir sistem sunmaktadır. Bu sayede gelecekteki geliştirmeler ve değişen gereksinimlere hızlı adapte olabilen sağlam bir yazılım altyapısı oluşturulmuştur.

Projenin Öne Çıkan Özellikleri

1. Gerçek Zamanlı Harita Görselleştirme: JXMapViewer kütüphanesi kullanılarak OSM (OpenStreetMap) verileri üzerinde gerçek zamanlı harita görüntüleme ve etkileşim.
2. Karma Ulaşım Modları: Yolcular tek bir arayüzde toplu taşıma, taksi ve yürüme gibi farklı ulaşım seçeneklerini bir arada görebilmekte.
3. Yolcu Tipi Bazlı Tarife Hesaplama: Öğrenci, yaşlı, engelli gibi farklı yolcu tipleri için dinamik indirim oranları ile otomatik ücret hesaplama.
4. Çoklu Ödeme Yöntemi Desteği: Kent Kart, nakit ve kredi kartı gibi farklı ödeme yöntemlerinin desteklenmesi ve her birine özel işlemler.
5. Özel Gün Tarife Uygulaması: Belirli özel günlerde ücretsiz seyahat imkanı ve tarife değişikliklerinin otomatik uygulanması.
6. Aktarma Optimizasyonu: Minimum aktarma ile en verimli rotaların hesaplanması ve görselleştirilmesi.
7. Çoklu Kriter Bazlı Rota Önerisi: En ucuz, en kısa mesafe, en az süre ve en az aktarma gibi kriterlere göre otomatik rota önerileri.
8. İnteraktif Durak Seçimi: Harita üzerinde tıklama ile başlangıç ve hedef noktalarının interaktif seçimi.
9. Gerçek Zamanlı Bakiye Sorgulama: Kent Kart kullanıcıları için anlık bakiye sorgulama özelliği.
10. Görsel Rota Vurgulama: Seçilen rotaların harita üzerinde renkli olarak vurgulanması ve kolay takip edilebilirliği.
11. Önbellek Mekanizması: Rota hesaplamaları için HashMap tabanlı önbellek sistemi ile performans optimizasyonu.
12. Esnek Veri Yapısı: HashSet ve HashMap kullanımı ile dinamik veri yönetimi ve etkin bellek kullanımı.
13. Sezgisel Kullanıcı Arayüzü: İkonik butonlar, renk kodlaması ve görsel ipuçları ile kullanıcı deneyimini iyileştiren arayüz tasarımı.

Sorular

1. *Tüm proje tamamlandıktan sonra, daha önce hiç kullanılmamış yeni bir ulaşım yöntemi ortaya çıktığında (örneğin, elektrikli scooter) sistemde hangi değişiklikler yapılmalıdır?*

Araç sınıfından kalıtım alan yeni bir scooter sınıfı tanımlanmalı bu şekilde sınıfı genişletmiş oluruz. VehicleFactory.registerVehicle() ile Scooter kayıt edilmeli. Eğer araç türlerini if-else ile tanımlamış olsaydık OCP'ye uygun olmazdı ama Factory Pattern kullandığımız için mevcut kodu değiştirmeden yeni türler ekleyebiliyoruz.

2. *Otonom taksi ve benzeri yeni ulaşım araçlarının projeye eklenmesi için ne tür değişiklikler gereklidir?*

Vehicle sınıfından türeyen bir OtonomTaksi sınıfı oluşturulmalıdır. VehicleFactory.registerVehicle() metodu çağrılarak OtonomTaksi sınıfı kayıt edilir ve nesnesi oluşturulur. Eğer rota hesaplamalarında özel bir algoritma gerekiyorsa, RouteFinder güncellenmelidir. Kullanıcıdan seçebileceği bir araç olarak istenirse onun için buton yaratılabilinir ve butonun action listeneri oluşturulur. Seçtiği araç tipine göre nesne oluşur.

3. *Daha önce yazılmış fonksiyonlardan hangilerinde değişiklik yapılmalıdır?*

Eklenen aracın duraktan durağa mı yoksa serbest mi gezeceğine bağlı olarak fonksiyonlarda değişiklik yapılabilir. Eğer duraktan durağa gidecekse yeni araç için sınıf oluşturmak ve RouteFinder classında VehicleFactory.registerVehicle() metodunu çağırarak kayıt etmek gerekir. Eğer kullanıcının seçimine bağlı olacaksa onun için buton yapılır ve butonun action listenerında nesne oluşturulur.

4. *Açık/Kapalı Prensibi (Open/Closed Principle) doğrultusunda, mevcut fonksiyonlarda herhangi bir değişiklik yapmadan yeni ulaşım araçları sisteme nasıl entegre edilebilir? Nesne hiyerarşisi başlangıçta nasıl tanımlanmış olsaydı, yeni araçların eklenmesi daha kolay olurdu?*

Projemiz genel olarak OCP prensibine uygun olarak tasarlanmıştır. Vehicle sınıfı tüm araçlar için bir temel oluşturuyor ve VehicleFactory sayesinde yeni araç türleri eklenebiliyor. Ödeme yöntemleri dinamik hale getirildi. Örneğin Taksi sınıfı KentKartı desteklemiyor. Yeni ödeme yöntemi eklendiğinde kodu değiştirmek gerekmiyor. Yeni araç eklenirken kodda büyük bir değişiklik yapmak gerekmiyor.

5. *65 yaş ve üzeri bireyler için ücretsiz seyahat hakkının 20 seyahat ile sınırlandırılması gerektiğinde, bu değişiklik projeye sonradan nasıl eklenebilir? başlangıçta nasıl tanımlanmış olsaydı, yeni araçların eklenmesi daha kolay olurdu?*

Yeni bir sınıf oluşturulabilir ve bu sınıf her yaşlı yolcunun ücretsiz seyahat hakkını takip edebilir.

Bu sayede eldery classında değişiklik yapmak gerekmez ama BiletHesapla() classında değişiklik yapmak gerekebilir.

6. *Bu sınırlama kod üzerinde nasıl uygulanmalıdır? Hangi sınıf ve fonksiyonlar etkilenecektir?*

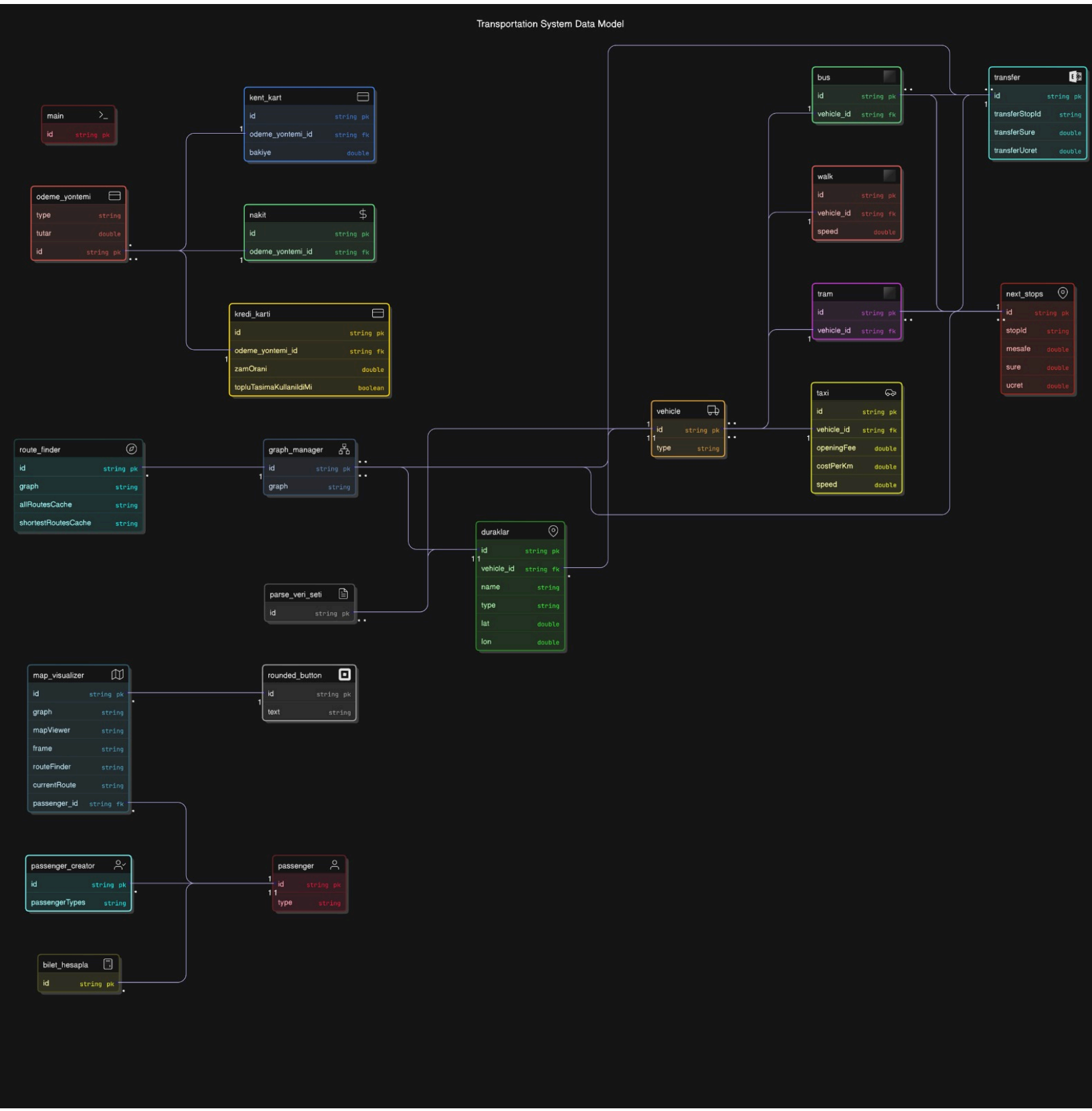
Yeni sınıf eklenebilir ve yeni sınıf aşağıdaki gibi olabilir:

```
public class ElderlyPassengerTracker {
   private static final int FREE_TRAVEL_LIMIT =
20;
   private Map<Elderly, Integer> travelCountMap =
new HashMap<>();

   public double getIndirimOrani(Elderly elderly) {
      int travelCount =
travelCountMap.getOrDefault(elderly, 0);

      if (travelCount < FREE_TRAVEL_LIMIT) {
         travelCountMap.put(elderly, travelCount +
1);
         return 1.0; // Ücretsiz seyahat
      } else {
         return 0.6; // 20 yolculuktan sonra %60
indirim
      }
   }
}
```

BiletHesapla() classında aşağıdaki gibi bir değişiklik yapılabilir:

```
public class BiletHesapla {
   private ElderlyPassengerTracker elderlyTracker =
new ElderlyPassengerTracker(); // Yaşlı yolcuları
takip eden sınıf

   public double calculatePrice(Passenger
passenger, double toplamUcret) {
      double indirimOrani;
```

```
        if (passenger instanceof Elderly) {

            indirimOrani =
elderlyTracker.getIndirimOrani((Elderly)
passenger);
        } else {

            indirimOrani = passenger.getIndirimOrani();
        }

        return toplamUcret - (toplamUcret *
indirimOrani);
    }
}
```


Transportation System Data Model
main
id string pk
kent_kart
id string pk
odeme_yontemi_id string fk
bakiye double
odeme_yontemi
type string
tutar double
id string pk
nakit
id string pk
odeme_yontemi_id string fk
kredi_karti
id string pk
odeme_yontemi_id string fk
zamOrani double
topluTasimaKullanildiMi boolean
bus
id string pk
vehicle_id string fk
transfer
id string pk
transferStopId string
transferSure double
transferUcret double
walk
id string pk
vehicle_id string fk
speed double
tram
id string pk
vehicle_id string fk
next_stops
id string pk
stopId string
mesafe double
sure double
ucret double
taxi
id string pk
vehicle_id string fk
openingFee double
costPerKm double
speed double
vehicle
id string pk
type string
route_finder
id string pk
graph string
allRoutesCache string
shortestRoutesCache string
graph_manager
id string pk
graph string
duraklar
id string pk
vehicle_id string fk
name string
type string
lat double
lon double
parse_veri_seti
id string pk
map_visualizer
id string pk
graph string
mapViewer string
frame string
routeFinder string
currentRoute string
passenger_id string fk
rounded_button
id string pk
text string
passenger_creator
id string pk
passengerTypes string
passenger
id string pk
type string
bilet_hesapla
id string pk