Programlama Labaratuvarı 4.Proje Raporu

Sadık Gölpek
sadikgolpek@gmail.com

Ali Kılınç
aliklnc4104@gmail.com

I.Özet

Bu proje, kullanıcıların haritadan başlangıç ve bitiş noktalarını seçerek en uygun ulaşım rotasını öğrenebilecekleri bir Java tabanlı masaüstü uygulamasıdır. Rota hesaplamaları, SOLID prensipleri doğrultusunda nesne yönelimli programlama yapısıyla gerçekleştirilmiştir. Uygulama; kullanıcı tipleri, ödeme yöntemleri ve araç bilgilerini esnek biçimde yönetebilmektedir. Ödeme yöntemi seçimine bağlı olarak, kullanıcıya bilgilendirme amacıyla ses efekti oynatılmakta ve bu işlev de, tek sorumluluk prensibine uygun şekilde ayrı bir sınıf aracılığıyla sağlanmaktadır. Arayüz, kullanıcı dostu olacak şekilde Swing kütüphanesiyle geliştirilmiş, etkileşimli butonlar ve ipuçları (tooltips) eklenmiştir.

II.Giriş

Ulaşım sistemlerinde doğru ve kişiselleştirilmiş rota önerileri, kullanıcı memnuniyeti açısından büyük önem taşımaktadır. Özellikle kent içi ulaşımda, bireylerin başlangıç ve bitiş noktalarına göre en uygun rotayı; mesafe, ücret ve süre gibi parametreleri dikkate alarak seçmeleri beklenir. Bu bağlamda geliştirilen masaüstü Java uygulaması, kullanıcıların harita üzerinde etkileşimli olarak başlangıç ve bitiş noktalarını seçerek farklı ulaşım senaryolarını deneyimlemelerine olanak tanımaktadır.

Proje ilk olarak Python diliyle geliştirilmeye çalışılmış, ancak nesne yönelimli programlamaya (OOP) dair soyutlama, çok biçimlilik ve sorumlulukların ayrımı gibi temel ilkeleri uygulamada zorluklar yaşanmıştır. Bu sebeple proje Java diline taşınarak, SOLID prensipleri temel alınarak yeniden yapılandırılmıştır. Bu prensipler sayesinde sistem, genişletilebilir ve sürdürülebilir bir mimariye kavuşturulmuştur.

Kod yapısı, yazılımın esnekliğini ve modülerliğini artırmak amacıyla mantıksal katmanlara ayrılmıştır. Örneğin:

- **Ödeme yöntemleri**, `PaymentMethod` isimli bir soyut sınıf temel alınarak modellenmiş ve `Nakit`, `KentKart`, `Kredi Kartı` gibi farklı sınıflar bu yapıdan türetilerek sistemin genişletilmesine olanak tanınmıştır.
- **Araçlar**, `Arac` isimli soyut sınıftan türetilmiş olup, örneğin `Taksi` sınıfı bu yapıdan geliştirilmiştir.
- **Yolcu tipleri**, hem soyut sınıf hem de arayüz kullanılarak esnek ve yeniden kullanılabilir biçimde tasarlanmıştır (`IYolcu` gibi). Bu yapı sayesinde öğrenci, öğretmen, yaşlı vb. yolcu tiplerinin sisteme kolaylıkla entegre edilmesi mümkün hale gelmiştir.

Projenin kullanıcı etkileşimli kısmı Swing tabanlı bir arayüz ile sağlanmıştır. Kullanıcılar, harita üzerinde başlangıç ve bitiş konumlarını seçerek rota hesaplama işlemlerini gerçekleştirebilmekte, bu sırada çeşitli butonlar ve **tooltip** destekli açıklamalarla yönlendirilmektedir.

Rota hesaplama sırasında sistem, arka planda JSON formatında saklanan verilerden yararlanmaktadır. Başlangıç ve bitiş durakları arasındaki tüm ulaşım seçenekleri taranmakta; bunlar içinden **en kısa mesafeli**, **en az ücretli** ve **en kısa süreli** olan rotalar kullanıcıya sunulmaktadır.

Eğer başlangıç ve bitiş konumu duraklardan 3 kilometre ya da daha fazla uzaklıktaysa, **otomatik olarak taksi** kullanımı devreye girmektedir. Taksi ücretlendirmesi sabit bir açılış ücreti ve 40 km/s sabit hız esas alınarak hesaplanmaktadır. Ayrıca seçilen **ödeme yöntemi** ve **yolcu tipi** gibi parametrelere göre toplam ücrette artış veya azalma meydana gelebilmektedir. Örneğin yaşlı yolcular için otobüs/tramvay ücretleri sistem tarafından sıfırlanmakta; buna karşın taksi ücreti tam olarak uygulanmaktadır. Ödeme yöntemi olarak **Kredi Kartı** seçildiğinde ise sistem, komisyon yansıtmakta ve ücreti yükseltmektedir.

Tüm bu işlemler, kullanıcı arayüzünde sade ve sezgisel bir biçimde yönetilebilirken; arka planda görev paylaşımı net, **tek sorumluluk ilkesine uygun** sınıflar üzerinden gerçekleştirilmekte, sistemin bakımı ve genişletilmesi kolaylaştırılmaktadır.

III. Yöntem

3.1 Programlama Dili ve Kullanılan Araçlar

Bu proje, tamamen Java programlama dili kullanılarak geliştirilmiştir. Geliştirme sürecinde aşağıdaki kütüphane ve araçlardan yararlanılmıştır:

- **Java Swing**: Kullanıcı arayüzünün oluşturulmasında Swing kütüphanesi tercih edilmiştir. Arayüzde butonlar, paneller, etiketler ve kullanıcı etkileşimi için tooltip'ler kullanılmıştır.
- **org.json**: JSON tabanlı veri setinden durak ve bağlantı verilerinin çekilmesi için `org.json` kütüphanesi kullanılmıştır.
- **.bat (Batch) Dosyası**: Uygulamanın daha kullanıcı dostu çalıştırılabilmesi amacıyla, Java dosyalarının doğrudan başlatılmasını sağlayan bir `.bat` dosyası oluşturulmuştur. Bu sayede komut satırı kullanmadan uygulama kolayca çalıştırılabilmektedir.

3.2 Yazılım Mimarisi ve SOLID Prensipleri

Projenin yazılım mimarisi, SOLID prensiplerine uygun şekilde yapılandırılmıştır. Bu prensiplerin uygulanışı aşağıda detaylandırılmıştır:

- **S - Single Responsibility (Tek Sorumluluk Prensibi):**
  Her sınıf yalnızca tek bir sorumluluğu yerine getirir:
  - `UlasimArayuzu`: Arayüz ve kullanıcı etkileşimi.
  - `HaritaPanel`: Harita çizimi, durakların görselleştirilmesi ve tıklanarak seçilmesi.
  - `RouteService`: Rota hesaplamasıyla ilgili iş mantığını yürütür.
  - `RouteCalculator`: Belirli algoritmalarla (DFS/BFS) rota üretimini yapar.
  - `SoundPlayer`: Ses dosyalarının çalınması işlemini izole şekilde gerçekleştirir.
- **O - Open/Closed (Açık/Kapalı Prensibi):**
  Kod yapısı genişletilmeye açık, ancak

mevcut kodlara dokunmadan yeni işlevlerin eklenmesine izin verecek şekilde tasarlanmıştır:

- `PaymentMethod` soyut sınıfı üzerinden yeni ödeme yöntemleri (örneğin `MobilOdeme`) kolaylıkla eklenebilir.
- `Arac` sınıfından türeyerek yeni araç türleri (örneğin `Minibus`) entegre edilebilir.

- **L - Liskov Substitution (Yerine Geçme Prensibi):**
  Türeyen sınıflar, üst sınıfların yerine sorunsuz şekilde geçebilir:
  - Örneğin `Taksi` nesnesi, `Arac` referansı ile sorunsuzca kullanılabilir.

- **I - Interface Segregation (Arayüz Ayrımı Prensibi):**
  Gereksiz metod bağımlılıkları azaltılmıştır:
  - `IYolcu` arayüzü, yalnızca yolculara özgü işlevleri içermektedir. Yolcu türleri (`GenelYolcu`, `Ogrenci`, `Yasli`, vs.) bu arayüzü implement ederek ihtiyaç duyulan işlevselliği sağlar.

- **D - Dependency Inversion (Bağımlılığı Tersine Çevirme Prensibi):**
  Üst düzey modüller, alt düzey modüllere doğrudan bağlı değildir:
  - `IMesafeHesaplayici` arayüzü sayesinde hem `Haversine` hem `Manhattan` algoritmaları dışarıdan sisteme entegre edilebilir. Bu sayede rota hesaplamalarında strateji deseni kullanılarak esneklik sağlanmıştır.

3.3 Öne Çıkan Özellikler

- **Harita Tabanlı Seçim:**
  Kullanıcılar, başlangıç ve bitiş noktalarını harita üzerinden fareyle seçebilmektedir. Seçilen noktalar, en yakın duraklarla eşleştirilmektedir.

- **Gelişmiş Rota Hesaplaması:**
  Sistem, başlangıç ve bitiş noktaları arasındaki tüm rota alternatiflerini BFS/DFS ile hesaplayarak en kısa mesafe, en ucuz ve en hızlı rotaları ayrı ayrı kullanıcıya sunar.

- **Akıllı Taksi Entegrasyonu:**
  Eğer seçilen konum ile durak arasındaki mesafe 3 km'den fazla ise sistem otomatik olarak taksi ücretini rota başına veya sonuna dahil eder. Taksi sabit hızla (40 km/s) çalışmakta ve ücretlendirme mesafeye göre yapılmaktadır.

- **Kapsamlı Ücret Hesaplama:**
  Toplu taşıma ücretleri; yolcu tipi (öğrenci, yaşlı, öğretmen vs.) ve ödeme yöntemine (Nakit, KentKart, Kredi Kartı) göre dinamik olarak değişir. Yaşlı yolcular toplu taşıma için ücretsiz seyahat ederken, taksi ücretlerinden muaf değildir.

- **Kullanıcı Dostu Arayüz:**
  Arayüzde tüm butonlara **tooltip** ipuçları eklenmiştir. Kullanıcılar, fareyi butonların üzerine getirdiklerinde ne işe yaradıklarına dair kısa açıklamalar görebilirler.

- **Sesli Bildirim Sistemi:**
  Rota hesaplandığında seçilen ödeme yöntemine göre sistem sesli uyarı verir. Örneğin kredi kartı veya KentKart seçildiğinde uygun ses dosyası çalıştırılır. Bu ses oynatma işlemi `SoundPlayer` sınıfı ile ayrıştırılmıştır.

- **SSS ve Ayarlar:**
  Kullanıcı arayüzü içinde Sık Sorulan Sorular, profil ve ödeme ayarları gibi bölümler yer almaktadır. Kullanıcılar bu sayede uygulamayı özelleştirebilmektedir.

IV.Deneysel Sonuçlar

Bu bölümde, projenin geliştirme sürecinde karşılaşılan teknik zorluklar, yapılan iyileştirmeler ve ortaya çıkan yazılım mimarisi değerlendirilmektedir.

4.1 Süreç Yönetimi ve Dil Tercihi

Proje başlangıçta Python diliyle geliştirilmiştir. Ancak zamanla nesne yönelimli programlamaya (OOP) dair soyutlama, arayüz kullanımı ve modüler yapıların oluşturulmasında sorunlar yaşanmıştır. Özellikle:

- Python'da arayüz (interface) yapılarının olmayışı,
- SOLID prensiplerini tam olarak uygulayamamak,
- JSON verisinden veri çekerken `KeyError`, `NoneType` gibi hatalarla sık karşılaşmak,
- Bileşenler arası sorumlulukların net ayrılmaması gibi problemler nedeniyle proje mimarisi karmaşık bir hâl almıştır.

Bu sebeple proje, 1 gün gibi kısa sürede Java diline taşınarak yeniden yapılandırılmıştır. Bu geçiş süreci, sınıf yapılarının net tanımlanabilmesi ve katı tip güvenliği sayesinde mimariyi sağlamlaştırmış, sürdürülebilirliği artırmıştır.

4.2 JSON Verisi ve Veri Doğruluğu

İlk veri setinde durakların enlem-boylam bilgileri hatalıydı. Rota hesaplamalarında ve harita çiziminde bu durum sorun yarattı. Bu nedenle JSON dosyasındaki koordinatlar, **Google Maps** üzerinden manuel olarak doğrulanmış ve her durağa ait gerçek enlem-boylam bilgileriyle güncellenmiştir. Bu iyileştirme, rota önerilerinin doğruluğunu önemli ölçüde artırmıştır.

4.3 Java'nın Sağladığı Avantajlar

Java'ya geçişle birlikte şu kazanımlar elde edilmiştir:

- Soyut sınıf ve arayüz yapılarıyla sistem esnek ve genişletilebilir hale gelmiştir.
- Kodlar SOLID prensiplerine uygun olarak ayrıştırılmıştır:
  - `PaymentMethod` sınıfı ile farklı ödeme türleri (`Nakit`, `KentKart`, `KrediKarti`) kolayca tanımlanmıştır.
  - `IYolcu` arayüzü ile kullanıcı tipi bazlı ücretlendirme yönetilmiştir.
  - `SoundPlayer`, `RouteCalculator`, `HaritaPanel` gibi sınıflar tek sorumluluk prensibine uygun şekilde izole edilmiştir.
- `org.json` kütüphanesiyle JSON verisi

kontrollü bir şekilde parse edilmiş ve `Durak`, `DurakBaglanti` nesnelerine dönüştürülerek sistemle bütünleştirilmiştir.

- GUI tarafında `Swing` ile kullanıcı dostu ve etkileşimli bir arayüz geliştirilmiştir.

4.4 Geliştirme Hızı ve Proje Yönetimi

Projenin Java'ya aktarılması sonrası sınıflar arası sorumluluklar netleştirilmiş, bileşenler bağımsız olarak geliştirilebilir hale getirilmiştir. Bu, hem geliştirme süresini hızlandırmış hem de ileride yapılacak değişikliklerin daha az hata riskiyle yapılabilmesini sağlamıştır. Küçük bir ekip ile sürdürülen bu proje, **doğru dil ve yapı seçiminin** yazılım başarısında ne kadar kritik olduğunu göstermiştir.

4.5 Fonksiyonel Test ve Gözlemler

Testler sonucunda sistemin:

- Başlangıç ve bitiş noktalarını doğru eşleyebildiği,
- En kısa mesafe, en az ücret ve en kısa süreye göre alternatif rota seçenekleri sunduğu,
- Kullanıcı tipine ve ödeme yöntemine göre dinamik ücretlendirme yaptığı (örneğin yaşlı bireyler toplu taşıma için ücret ödemezken, taksi ücretinin uygulandığı),
- 3 km'den uzak konumlarda otomatik olarak taksi önerisi yaptığı,
- Harita üzerindeki durakları doğru şekilde çizdiği ve kullanıcı etkileşimini başarılı şekilde sağladığı gözlemlenmiştir.

V.Sonuçlar

Bu proje kapsamında, kullanıcıların ulaşım senaryolarını harita üzerinde interaktif biçimde simüle edebileceği, kullanıcı tipi ve ödeme şekline göre dinamik hesaplamalar yapabilen, modüler ve sürdürülebilir bir masaüstü uygulama geliştirilmiştir.

Projenin Java diliyle yapılması, özellikle nesne yönelimli programlamayı etkin biçimde uygulama, sınıflar arası sorumlulukları net ayırma ve çoklu geliştirici ortamında iş bölümünü kolaylaştırma açısından önemli avantajlar sağlamıştır. SOLID prensipleri doğrultusunda geliştirilen yapı sayesinde yeni özellikler eklemek oldukça kolaylaşmıştır.

Proje süresince GPT teknolojisinden de faydalanılmış; kod yapısının iyileştirilmesi, mimari düzenlemeler ve teknik öneriler gibi alanlarda destek alınarak daha verimli ve hatasız bir geliştirme süreci yürütülmüştür.

Bu çalışma, özellikle yazılım mimarisi ve süreç yönetimi açısından da değerli bir deneyim sağlamış; gerçek dünya problemlerinin modüler ve esnek yapılarla nasıl çözülebileceğini göstermiştir.

VI.Alogoritmanın Kaba Kodu

Projemiz, SOLID prensiplerine uygun şekilde sınıflara ayrılmış modüler bir mimariye sahiptir. Her sınıf, yalnızca kendi sorumluluğunu yerine getirmek üzere tasarlanmıştır. Örneğin, arayüz işlemleri `UlasimArayuzu`, harita çizimi ve durak yönetimi `HaritaPanel`, rota hesaplamaları `RouteCalculator`, ödeme işlemleri `PaymentMethod` ve türevleri gibi ayrı sınıflarda toplanmıştır.

Bu yapı sayesinde projede çok sayıda sınıf ve

Java dosyası yer almaktadır. Tüm bu sınıfların algoritmalarını ayrı ayrı göstermek mümkün olmadığı için, sistemin ana kontrol akışını sağlayan `UlasimArayuzu` sınıfı üzerinden genel işleyişi temsil eden bir pseudocode hazırlanmıştır.

Ayrıca sistemde, kullanıcıya iki farklı mesafe hesaplama algoritması (Haversine veya Manhattan) arasından seçim yapma imkânı sunulmuştur. Bu sayede esnek ve strateji temelli bir yapı kurulmuş, `IMesafeHesaplayici` arayüzü ile SOLID'in "Dependency Inversion" prensibi uygulanmıştır.

Aşağıda `UlasimArayuzu` sınıfı üzerinden genel işleyişi temsil eden bir pseudocode hazırlanmıştır.

PROGRAM Başlat

1. Arayüzü oluştur (UlasimArayuzu sınıfı):
   - HaritaPanel merkezde (harita çizimi ve durak tespiti burada yapılır)
   - Sağ panelde: rota bilgisi, başlangıç/bitiş seçimi ve hesapla butonları
   - Menü çubuğu: profil, ödeme, tercih ayarları ve yardım içerir

2. JSON dosyasından durakları yükle (VeriYukleyici sınıfı)

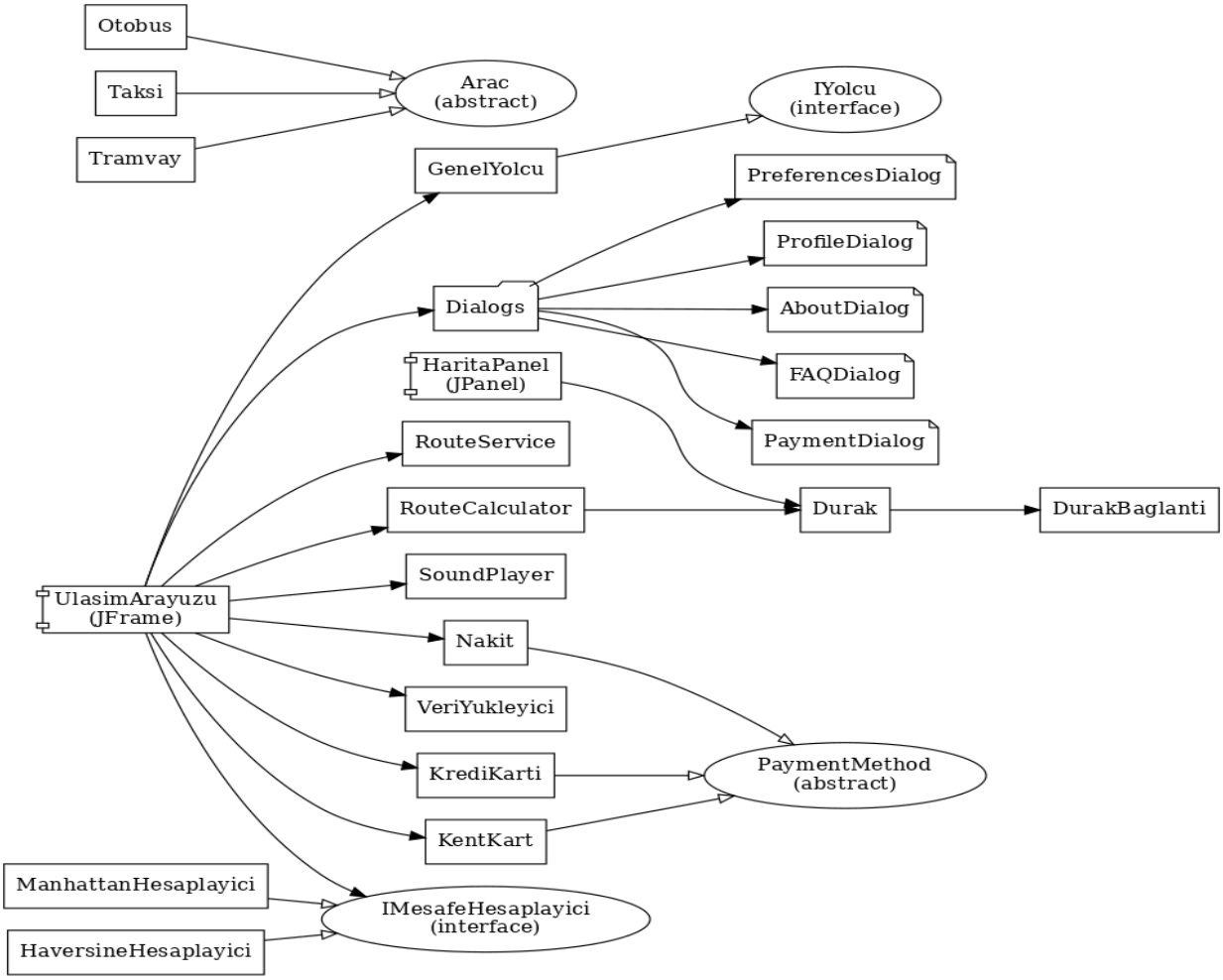
3. HaritaPanel nesnesini oluştur ve başlat:
   - Kullanıcının harita üzerinde tıklayarak nokta seçmesine izin ver
   - Seçilen noktaya en yakın durağı bul
   - 3 km'den uzaksa uyarı göster ve taksi kullanımını öner

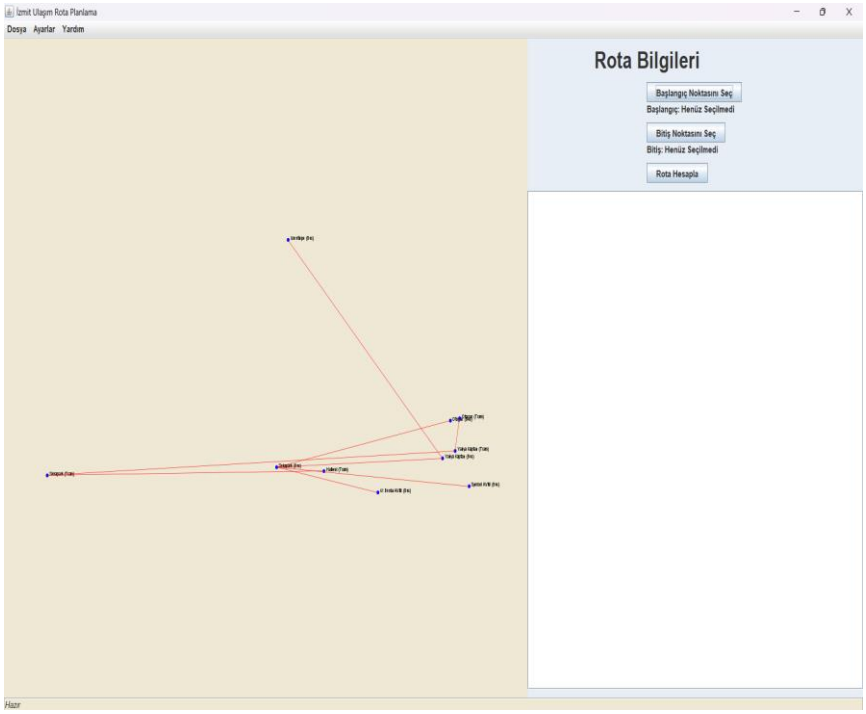
4. Mesafe hesaplama stratejisini ayarla (IMesafeHesaplayici interface'i):
   - Kullanıcı tercihlerine göre Haversine veya Manhattan algoritması seçilebilir
   - Bu seçim strateji deseni sayesinde kolayca uygulanır ve değiştirilebilir

5. "Rota Hesapla" butonuna basıldığında:
   - Başlangıç/bitiş noktalarının seçildiğini kontrol et
   - RouteService sınıfını çağır:
     - Başlangıç ve bitiş noktasından durağa mesafe hesapla
     - Taksi gerekiyorsa ücret ve süreyi hesapla (Taksi sınıfı)
     - RouteCalculator ile toplu taşıma rotalarını DFS ile bul
     - Ödeme yöntemi ve yolcu tipine göre ücret dönüştür (PaymentMethod & IYolcu)
     - En kısa, en ucuz ve en hızlı rotaları analiz et
     - Tüm bu bilgileri HTML olarak JEditorPane'e yazdır

6. Ek özellikler:
   - Ses efekti çalınması (SoundPlayer sınıfı) ödeme türüne bağlı
   - ToolTip'ler ile kullanıcı dostu arayüz sunulur
   - .bat dosyası ile uygulama tek tıkla açılabilir

PROGRAM Biter


Otobus
Taksi
Tramvay
Arac
(abstract)
IYolcu
(interface)
GenelYolcu
PreferencesDialog
ProfileDialog
Dialogs
AboutDialog
FAQDialog
HaritaPanel
(JPanel)
PaymentDialog
RouteService
RouteCalculator
Durak
DurakBaglanti
SoundPlayer
UlasimArayuzu
(JFrame)
Nakit
VeriYukleyici
KrediKarti
PaymentMethod
(abstract)
KentKart
ManhattanHesaplayici
IMesafeHesaplayici
(interface)
HaversineHesaplayici



Rota Seçeneği 4:

Umuttepe (Bus) -> Yahya Kaptan (Bus): 5,00 km, 4,00 TL, 12,00 dk

Yahya Kaptan (Bus) -> Yahya Kaptan (Tram) (Transfer): 0,00 km, 0,00 TL, 2,00 dk

Yahya Kaptan (Tram) -> Otogar (Tram): 1,20 km, 2,50 TL, 5,00 dk

Otogar (Tram) -> Otogar (Bus) (Transfer): 0,00 km, 0,50 TL, 2,00 dk

Otogar (Bus) -> Sekapark (Bus): 3,50 km, 3,00 TL, 10,00 dk

Sekapark (Bus) -> 41 Burda AVM (Bus): 4,50 km, 3,50 TL, 14,00 dk

Toplam: 17,97 km, 13,50 TL, 45,00 dk

----- **EN OPTIMIZE ROTALAR** -----

En Kısa Süreli Yol: 34,00 dk

(bus_umuttepe->bus_yahyakaptan->bus_sekapark->bus_41burda)

En Ucuz Yol: 9,50 TL

(bus_umuttepe->bus_yahyakaptan->tram_yahyakaptan->tram_sekapark->bus_sekapark->bus_41burda)

En Kısa Mesafeli Yol: 15,27 km

(bus_umuttepe->bus_yahyakaptan->bus_sekapark->bus_41burda)

7. Yazar Katkıları

Bu proje, **Sadık Gölpek** ve **Ali Kılınç** tarafından ortaklaşa geliştirilmiştir. Görev dağılımı aşağıdaki şekilde gerçekleştirilmiştir:

Sadık Gölpek

- **Proje Fikri ve Yapılandırması:** Uygulamanın temel senaryosunun ve kullanıcı akışlarının oluşturulması.
- **Kod Geliştirme:** SOLID prensiplerine uygun şekilde sınıf tasarımı ve servis katmanlarının yazımı.
- **Veri İşleme:** JSON veri setinin düzenlenmesi, durak enlem-boylam bilgilerinin Google Maps üzerinden alınarak güncellenmesi.
- **Ses ve Arayüz Entegrasyonu:** Ödeme seslerinin ve isteğe bağlı müzik sisteminin entegre edilmesi.
- **Test ve Hata Ayıklama:** Uygulamanın farklı senaryolarla test edilmesi, Python'dan Java'ya geçişte oluşan hataların çözülmesi.
- **Dokümantasyon:** Rapor, pseudocode, UML diyagramı ve akış şemalarının oluşturulması.

Ali Kılınç

- **Arayüz Geliştirme:** Swing tabanlı kullanıcı arayüzünün geliştirilmesi; butonlar, menüler, ipuçları (tooltips) gibi etkileşimli bileşenlerin tasarımı.
- **Veri Yönetimi:** JSON dosyasından veri çekilmesini sağlayan `VeriYukleyici` sınıfının yazımı ve entegrasyonu.
- **OOP Yapılar:** Araç, yolcu ve ödeme sistemleri için soyut sınıfların ve interface yapıların oluşturulması.
- **Mimari Düzenleme:** Class'ların paketlere ayrılması ve projenin modüler bir yapıya kavuşturulması.
- **.bat Dosyası:** Projenin kolay başlatılabilmesi için çalıştırılabilir script'in hazırlanması.
- **Deneysel Süreç Yönetimi:** Python'dan Java'ya geçiş sürecinin analiz edilmesi, alternatiflerin değerlendirilmesi.

Ayrıca proje boyunca planlama, fikir alışverişi, hata giderme ve genel kod yapısı konularında eşit sorumluluk üstlenilmiş, GPT desteğinden verimli biçimde faydalanılmıştır.

8. Kaynakça

1. **Google Maps Koordinat Sistemi** – Harita üzerinden doğru enlem-boylam çekimi https://www.google.com/maps
2. **Yazılım Mimarisi ve OOP Yapıları Hakkında Genel Bilgi**
   a. GeeksForGeeks: https://www.geeksforgeeks.org/object-oriented-programming-oops-concept-in-java/
3. **ChatGPT Yardımı** – Kod yapılarının önerilmesi, rapor yazımı ve SOLID ilkeleriyle ilgili açıklamalar için OpenAI ChatGPT desteği alınmıştır.

9. Tartışma ve Geliştirme Önerileri

• Yeni Bir Ulaşım Türü (Örneğin: Elektrikli Scooter) Sisteme Nasıl Eklenir?

Projemiz SOLID prensiplerine uygun şekilde, özellikle **Açık/Kapalı (Open/Closed)** prensibi dikkate alınarak tasarlandığı için, yeni bir ulaşım türü eklemek mümkündür. Tüm araçlar `Arac` isimli soyut sınıftan türediği için, elektrikli scooter eklenmek istendiğinde yalnızca `Scooter` adında yeni bir sınıf oluşturulup `Arac` sınıfından kalıtım alması yeterlidir.

Bu sayede sistemdeki diğer kodlara dokunmadan yeni sınıf projeye entegre edilir.

• Otonom Taksi ve Benzeri Yeni Ulaşım Araçlarının Eklenmesi

Otonom taksi de tıpkı scooter gibi `Arac` soyut sınıfından türeyen yeni bir sınıf olarak tanımlanabilir.

• Hangi Fonksiyonlar Değişebilir?

Eğer ücret hesaplama algoritması çok özel bir duruma göre çalışacaksa, örneğin scooter sadece kısa mesafelerde çalışsın gibi bir kural varsa, `RouteService` sınıfındaki mantığa ufak koşul eklenebilir. Ancak tasarım doğru yapıldıysa bu gereklilik azalır.

• Open/Closed Prensibi Doğrultusunda Araçları Nasıl Ekleyebiliriz?

Başlangıçta ulaşım araçlarının hepsi bir **`Arac`** soyut sınıfında toplanmış ve her bir alt sınıf `hesaplaUcret()` metodunu kendine göre tanımlamıştır. Bu yapı sayesinde, yeni bir araç geldiğinde sadece bu sınıftan kalıtım alarak tanımlanır. Kodun geri kalanında bir değişiklik yapılmasına gerek kalmaz.

Bu da **“yeniye açık, eskiye kapalı”** prensibini doğrudan destekler.

• 65 Yaş Üstü Bireyler İçin Ücretsiz Seyahatin 20 Seyahat ile Sınırlandırılması

Bu kural `Yasli` sınıfında uygulanabilir. `Yasli` sınıfına bir sayaç (`int kalanUcretsizHakki = 20`) eklenerek her toplu taşıma kullanımında bu sayı azaltılır. Sayaç sıfıra ulaştığında, `indirimOrani()` metodu artık 1 değil, örneğin %50 gibi bir orana dönebilir.

• Bu Değişiklik Hangi Sınıfları Etkiler?

- `Yasli` sınıfı: yeni alan ve metotlar eklenir.
- `RouteService` sınıfı: yolcu tipine göre ücret indirimi uygulanırken bu sınıfın yeni durumu kontrol edebilir.