YILDIZ TEKNİK ÜNİVERSİTESİ

ELEKTRİK-ELEKTRONİK FAKÜLTESİ

BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ

BLM2012 - Nesneye Yönelik Programlama

Kubilay Kayra Kıvrak - 24011079 (Grup 3)

kayra.kivrak@std.yildiz.edu.tr

Hasan Altan Turan – 23011042 (Grup 1)

altan.turan@std.yildiz.edu.tr

Proje Grubu 26

Prof. Dr. Mehmet Sıddık AKTAŞ

Dr. Öğr. Üyesi Furkan ÇAKMAK

1. Giriş ve Ön Bilgi

Bu projede kapsamında bir havayolu şirketi için kullanıcı arayüzüne sahip program tasarlanmıştır. Program Java programlama dili kullanılarak yazılmıştır. GUI elementleri için javafx kütüphanesi, unit test yürütebilmek için JUnit kütüphanesi kullanılmıştır. Veri güvenliğine ve edge case handlinge önem verilmiştir. Proje yönergesinde verilen senaryolar gerçekleştirilmiş, yönergede istenen sınıflar ve programın çalışması gereken ek sınıflar uygulanmış, kullanıcı arayüzünde sonuçlar görselleştirilmiştir. User, Staff ve Admin tipi kullanıcılar için işlevler sağlanmıştır. Sonucunda tamamen işlevsel bir kullanıcı arayüzü ve uçuş yönetim sistemi elde edilmiştir.

Elde edilen sisteme ait UML grafları ilgili bölümlerde ve son sayfada verilmiştir, ayrıca UML tablosu çözünürlüğün bazı kısımlar için düşük olması sebebiyle teslim edilen .zip dosyasında .pdf formatında paylaşılmıştır. Projeye ait video hem .zip dosyasında .mp4 formatında teslim edilmiştir hem de Youtube platformundan paylaşılmıştır.

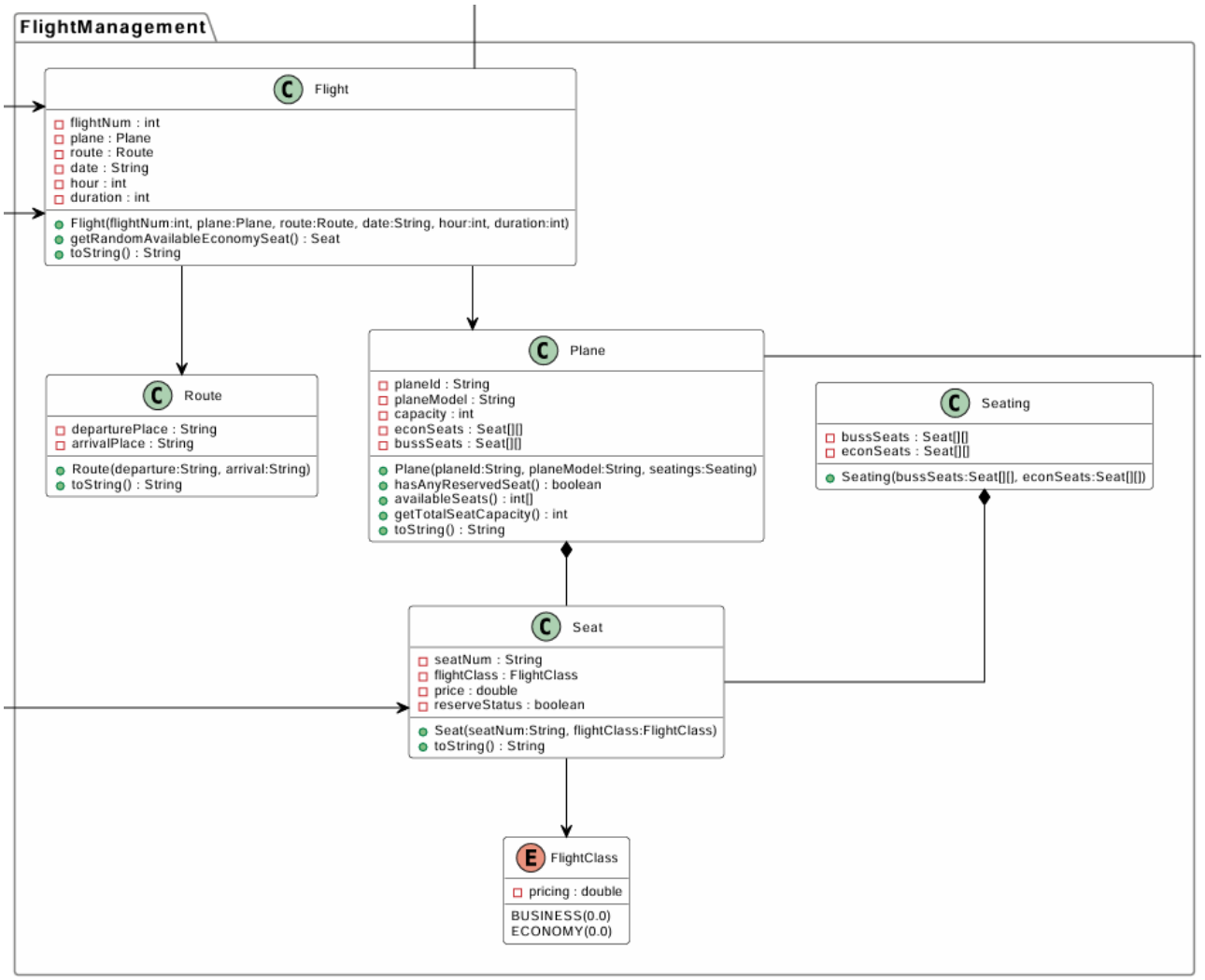
Video Linki: https://youtu.be/JNUBreurMBg
Yedek Link: https://youtu.be/pkAuV3jQkqM

2. Sistem Mimarisi

Bu bölümde uçuş yönetim sisteminin çalışma mantığı için gerekli olan sınıflardan bahsedilmiştir.

2.1. FlightManagement Paketi

FlightManagement paketi, uçuş yönetim sisteminin temel sınıflarını barındırır. Bu paket; uçuş süreçlerinin yönetilebilmesi için gerekli olan nesnelerin tanımlarını içerir. Sistemin diğer katmanları (Manager, GUI, Service), veri taşıma ve işleme süreçlerinde bu paketteki sınıfları temel veri yapıları olarak kullanır.



2.1.1. Flight Class

Flight sınıfı, oluşturulan bir uçuşu temsil eder ve içinde uçuşla alakalı bilgileri tutar. Management ve diğer ilgili sınıflar uçuşla alakalı bilgilere sadece buradan ulaşabilirler.

Functions

1. **getRandomAvailableEconomySeat** : Uçağın içindeki ekonomi sınıfı olan bir koltuğun rastgele seçilmesini sağlar. Senaryo 1'de kullanmak için oluşturulmuştur.

2.1.2. FlightClass Enum

Seat oluştururken bu koltuğun hangi sınıf olduğunu yani economy / business ayrımını belirten enum'dur.

2.1.3. Plane Class

Oluşturulan uçakları temsil eden ve uçakla ilgili bilgileri tutan class'tır. Management ve diğer ilgili classlar uçakla alakalı bilgilere sadece buradan ulaşabilirler.

Functions

1. **hasAnyReservedSeat**: boolean döndüren ve uçağın içindeki Seat[][] arraylerinde gezinerek , bu uçağın herhangi bir koltuğunun alınıp alınmadığını döndüren metod.
2. **availableSeats**: Uçağın içerisindeki koltukların alınıp alınmamasını ekonomi ve business için ayrı ayrı inceler ve boş olan koltuk sayısını int arreyde döndürür. İnt arreyde döndürmesinin sebebi ekonomi ve business koltuk sayılarını ayrı ayrı verebilmesidir.

3. **getTotalSeatCapacity** : Uçaktaki business ve economy koltuklarının dolu olup olmadığı farketmeksizin sayısına bakarak toplam koltuk kapasitesini döndüren fonksiyondur.

2.1.4. Route Class

Uçuşların oluşturulabilmesi için gereken uçak kalkış ve iniş yerlerini (departure , arrival) saklayan classtır.

2.1.5. Seat Class

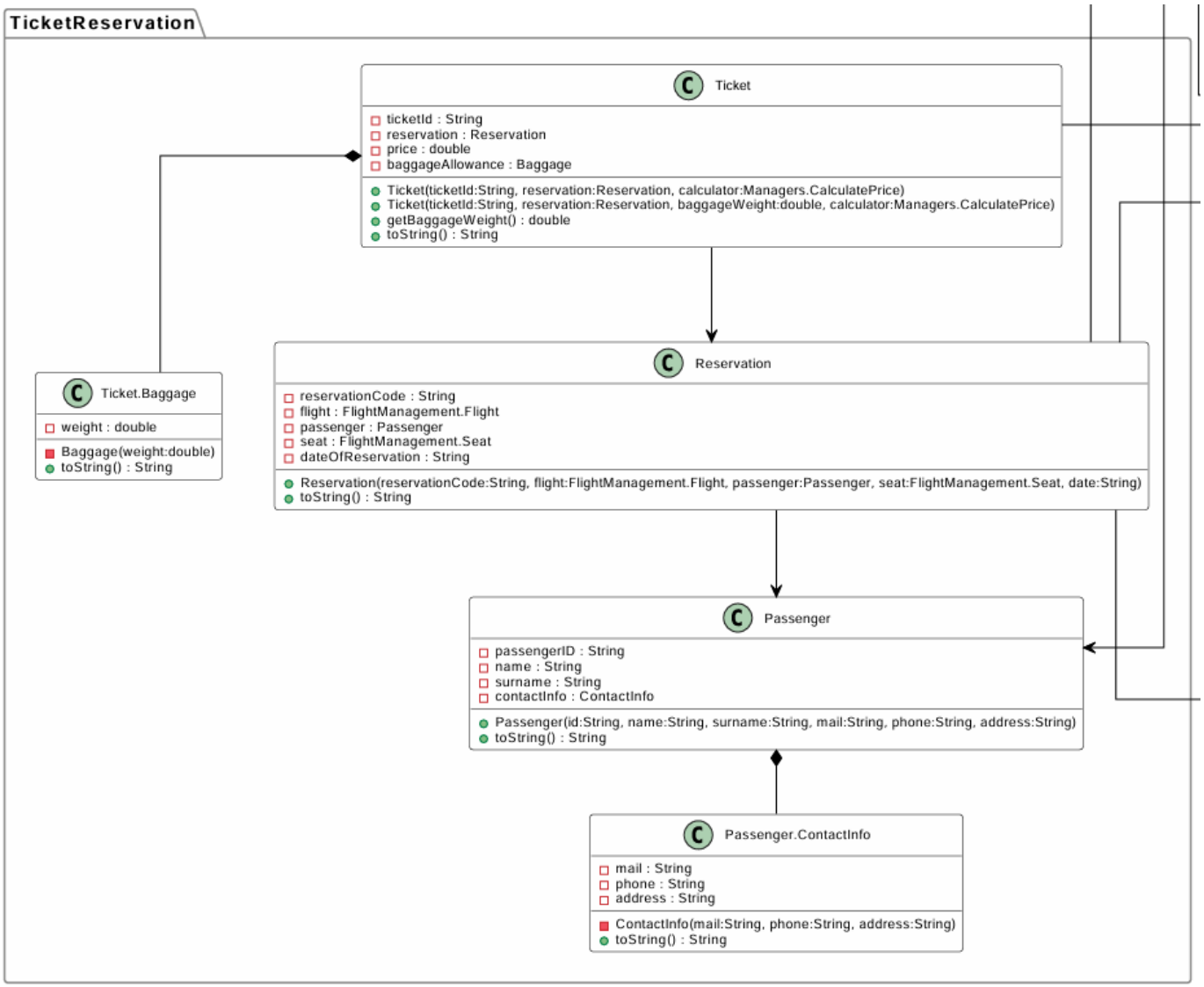
Oluşturulan bir koltuğu ifade eden ve onunla alakalı bilgileri tutan classtır. Rezervasyon durumu, business / ekonomi sınıfından hangisi olduğunu, fiyatını ve koltuk numarasını saklar.

2.1.6. Seating Class

Uçakların içindeki koltuk düzeninin oluşturulup uçağın içine yazılabilmesi için SeatManager ve PlaneManager arasında bir taşıyıcı görevi görür ve business / ekonomi koltuk düzenini Seat[][] matriksleri içerisinde saklar.

2.2. TicketReservation Paketi

TicketReservation paketi, havayolu sisteminin müşteri ve satış süreçlerini yöneten temel sınıfları içerir. Bu paket; yolcu kimlik bilgilerini, oluşturulan rezervasyon kayıtlarını ve ödeme/bagaj detaylarını içeren bilet nesnelerini kapsar. FlightManagement paketindeki verileri yolcularla ilişkilendirerek, bir uçuşu 'satılabilir' ürüne dönüştürür.



2.2.1. Passenger Class

Oluşturulan bir yolcuyu ifade eden ve onunla alakalı bilgileri tutan classtır. Yolcunun ID numarasını , isim soyismini , iletişim bilgilerini (mail , adres , telefon numarası) saklar. Management ve diğer ilgili classlar yolcuyla alakalı bilgilere sadece buradan ulaşabilirler.

2.2.2. ContactInfo Inner Class

Passenger Class'ın inner classı olan bu class , yolcuların iletişim bilgilerini tutmayı sağlar.

2.2.3. Reservation Class

Bir uçuşta yolcunun yaptığı rezervasyonu ifade eden ve onunla alakalı bilgileri tutan classtır. Rezervasyon numarasını , rezervasyonun yapıldığı uçuşu , rezervasyonu yapan yolcuyu, rezervasyon yapılan koltuğu ve rezervasyon tarihini içerisinde saklar. Management ve diğer ilgili classlar rezervasyonla alakalı bilgilere sadece buradan ulaşabilirler.

2.2.4. Ticket Class

Bir yolcunun rezervasyonunu yaptığı uçuşa aldığı bileti ifade eden ve onunla ilgili bilgileri tutan classtır. Bilet numarasını , bileti alınmış rezervasyonu , bilet fiyatını ve bagaj bilgisini içinde saklar.

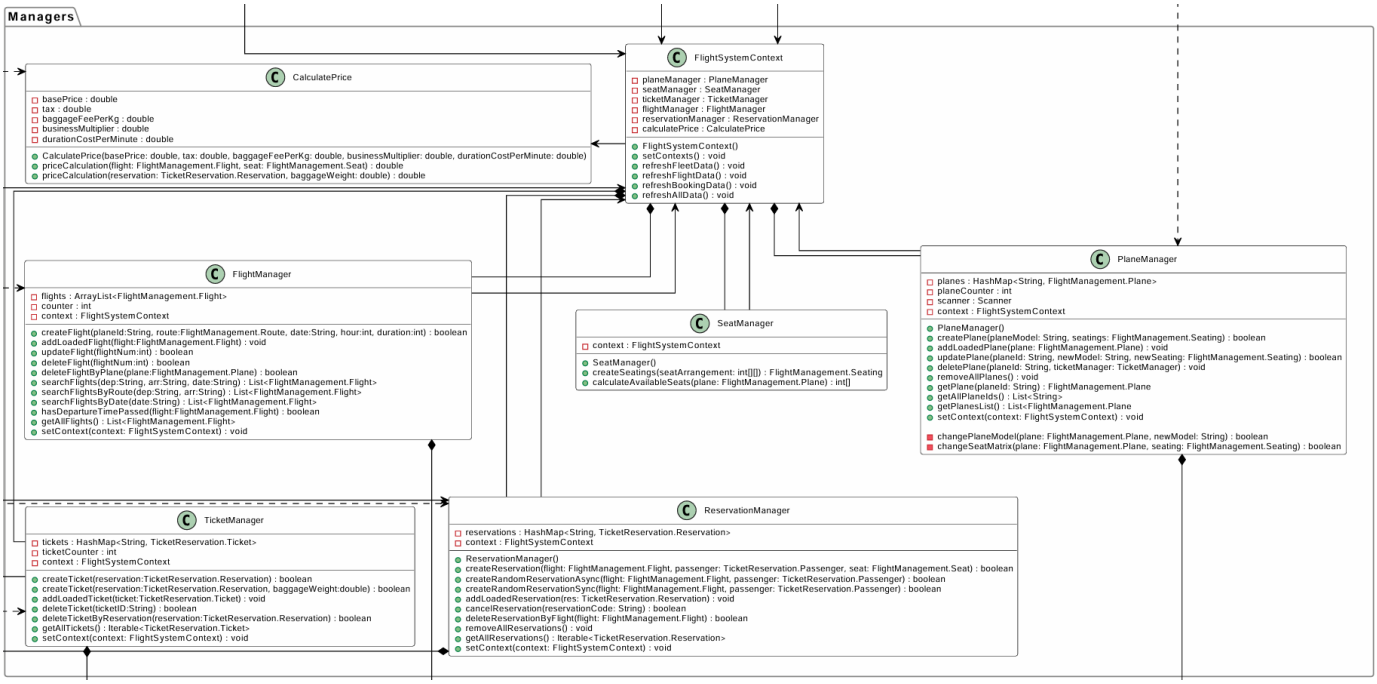
Ayrıca rezervasyon ve bagaj bilgisine göre bu class ‘ dan yeni bir tane üretilirken fiyatı CalculatePrice classıyla otomatik hesaplanarak içerisindeki fiyat bilgisi üretildiği an oluşturulur. 2 adet constructor içermesinin sebebi, bagajlı veya bagajsız biletlerin üretilebilmesini sağlamaktır.

2.2.5. Baggage Inner Class

Ticket Class’ın inner classı olan bu class, biletteki bagajın ağırlığını saklamak için kullanılır.

2.3. Manager Paketi

Manager paketi, sistemdeki tüm operasyonların yönetildiği merkezdir. Nesneler üzerindeki hesaplamalar, kontroller ve veri akışı bu paketteki yöneticiler aracılığıyla sağlanır.

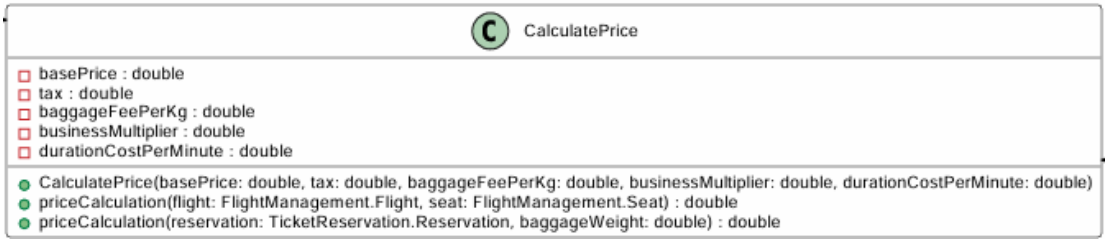


2.3.1. Calculate Price Class

Biletlerin fiyatının hesaplanmasını ve bu fiyat hesaplama parametrelerini tutan classtır.

Functions

1. **priceCalculation**: Rezervasyon ve bagaj ağırlığını alarak bu rezervasyondaki koltuk durumu, başlangıç fiyatı, bu rezervasyondaki uçuşun süresini ve bagaj ağırlığı parametrelerine göre bilet fiyatını belirler. İki adet price Calculation fonksiyonunun kullanılmasının sebebi bagaj olup olmama durumlarına göre fiyatı doğru belirleyebilmektir.

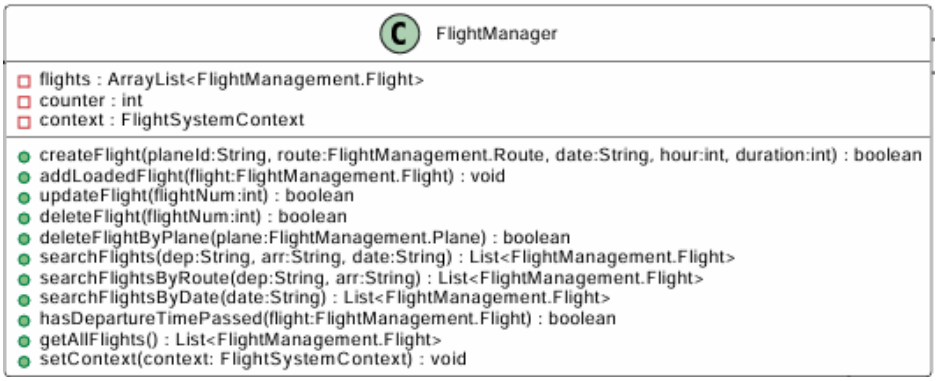


2.3.2. FlightManager Class

Uçuşların oluşturulmasını , bu uçuşların bir ArrayList de tutulmasını , uçuşlar üzerinde değişiklikler yapılmasını, uçuşların silinmesini, belirli parametrelere göre uçuş aranmasını, uçuşların süresinin geçip geçmediğinin belirlenmesini sağlayan classtır.

Functions

1. **createFlight**: Bir uçuşun oluşturulmasını ve ArrayList yapısının içine eklenmesini sağlar.
2. **deleteFlight**: Bir uçuşu Array List'in içinden çıkarır ve bu uçuşla bağlantılı bir rezervasyon varsa ayrıca onun da silinmesi için çağrı yapar.
3. **searchFlights**: Verilen iniş, kalkış ve tarih parametrelerine göre uçuş Array Listinin içinden bu parametrelerle eşit olan uçuşları bulur ve bir liste şeklinde döndürür.
4. **hasDepartureTimePassed**: Anlık olarak sistem tarihine bakar ve bunu içine gönderilen uçuşla karşılaştırarak bu uçuşun tarihinin geçip geçmediğine karar verir.
5. **addLoadedFlight**: Dosya işlemlerinden gelen uçuş objelerinin ArrayList'e kaydedilmesini sağlar.
6. **setContext**: Managerlar arası iletişim için kullanılır , FlightSystemContext'e bağlanılır.
7. **getAllFlights** : Tüm uçuşları bir liste olarak döndürür.

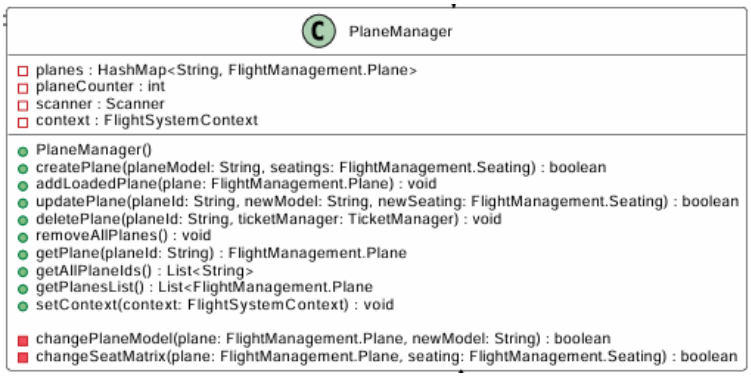


2.3.3. PlaneManager Class

Uçakların oluşturulmasını , bu uçakların bir HashMap üzerinde tutulmasını , uçaklar üzerinde değişiklik yapılmasını ve uçakların silinmesini sağlayan classtır.

Functions

1. **createPlane**: Bir uçağın oluşturulmasını ve HashMap yapısının içine eklenmesini sağlar.
2. **deletePlane**: Bir uçağın silinmesini sağlar ve ayrıca o uçağın yapacağı uçuşların da silinmesini tetikler.
3. **updatePlane**: Uçak modelini ve koltuk düzeninin değiştirilmesine olanak sağlar.
4. **addLoadedPlane**: Dosya işlemlerinden gelen uçak objelerinin HashMap'e kaydedilmesini sağlar.
5. **setContext**: Managerlar arası iletişim için kullanılır , FlightSystemContext'e bağlanılır.
6. **getPlane** : Id'si verilen uçağı HashMap üzerinden bularak döndürür.
7. **getAllPlaneIds** : Tüm uçakların id'lerini string listesi olarak döndürür.
8. **getPlaneList** : Tüm uçakları liste olarak döndürür.

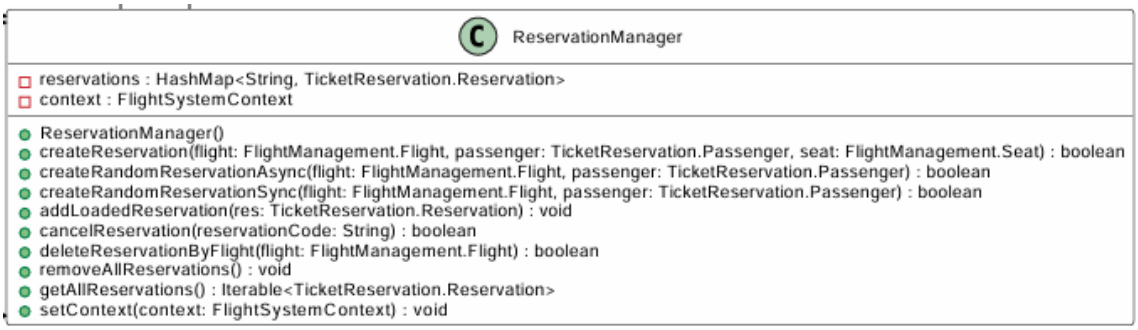


2.3.4. ReservationManager Class

Rezervasyonların oluşturulmasını , verilen senaryolar için bir uçaktan rastgele rezervasyonlar alınmasını ve rezervasyonların silinmesini sağlayan classtır.

Functions

1. **createReservation** : Verilen uçuş , yolcu ve koltuk için girilen yolcuya bir rezervasyon tanımlayan ve bunu HashMap içine kaydeden fonksiyondur. Ayrıca girilen koltuğun gerçekten uçuşun yapılacağı uçakta olup olmadığını ve bu koltuğun rezerve olup olmadığını da kontrol eder.
2. **createRandomReservationSync/Async** : Senaryolar için verilen uçuşta rastgele rezervasyonlar oluşturan ve bunları senkron ya da asenkron şekilde yapan fonksiyondur.
3. **cancelReservation**: Verilen rezervasyon koduna göre rezervasyonu siler ve bu rezervasyon için bilet tanımlandıysa o biletlerin de silinmesini tetikleyen fonksiyondur.
4. **addLoadedReservation**: Dosya işlemlerinden gelen rezervasyon objelerinin HashMap'e kaydedilmesini sağlar.
5. **setContext** : Managerlar arası iletişim için kullanılır , FlightSystemContext'e bağlanılır.
6. **deleteReservationByFlight:** Belirli bir uçuş iptal edildiğinde veya silindiğinde, o uçuşa bağlı olan tüm rezervasyonların toplu olarak sistemden silinmesini sağlayan fonksiyondur.
7. **removeAllReservations:** Sistemde kayıtlı bulunan tüm rezervasyon verilerini temizleyen (hashmap yapısını boşaltan) fonksiyondur.
8. **getAllReservations:** HashMap içerisinde tutulan tüm aktif rezervasyon nesnelerine erişilmesini sağlayan ve bunları bir liste olarak döndüren fonksiyondur.



2.3.5. SeatManager Class

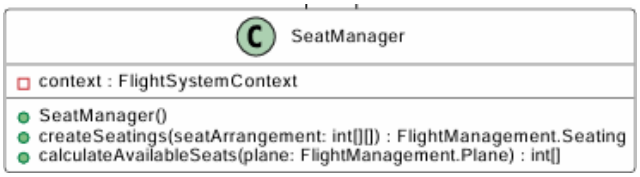
Uçaklar için koltuk düzenleri oluşturulmasını ve bir uçaktaki boş koltuk sayısını hesaplayan classtır.

Functions

1. **createSeatings**: Verilen int[][] 'e göre Seating classından üretir. Bu int[][] mekanizması şöyle işler: int[0] kısmı business koltuklarını ifade ederken int[0][0] bu koltuklardan kaç sütun ,

int[0][1] bunlardan kaç satır olduğunu ifade eder. Aynısı int[1] kısmında economy koltukları için geçerlidir. Bu mekanizmaya göre bir koltuk düzeni oluşturulur ve Seating classı ile kaydedilir.

2. **calculateAvailableSeats** : Verilen uçak için bu uçağın içindeki boş koltuk sayısını business ve ekonomi için ayrı ayrı döndürür. Business ve ekonomi sayısını ayırt edebilmek için int[] kullanılır.

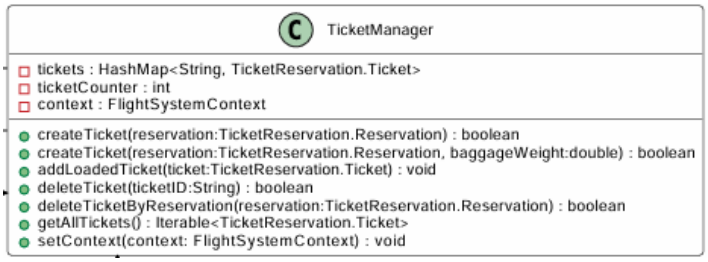


2.3.6. TicketManager Class

Biletlerin oluşturulup silinmesini sağlayan classtır.

Functions

1. **createTicket**: Verilen parametrelere göre bilet oluşturur ve bunu HashMap yapısına kaydeder.
2. **deleteTicket**: Bilet numarasına göre bir bileti HashMap yapısından çıkartır.
3. **addLoadedTicket** : Dosya işlemlerinden gelen bilet objelerinin HashMap yapısına kaydedilmesini sağlar.
4. **deleteTicketByReservation:** Bir rezervasyon iptal edildiğinde, o rezervasyona ait olan bileti bularak sistemden silen fonksiyondur.
5. **getAllTickets:** HashMap içerisinde tutulan tüm bilet nesnelerine erişilmesini sağlayan ve bunları döndüren fonksiyondur.
6. **setContext:** Managerlar arası iletişim için kullanılır, FlightSystemContext'e bağlanılmasını sağlar.



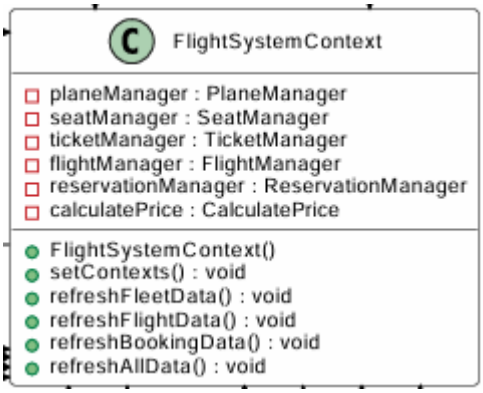
2.3.7. FlightSystemContext Class

Tüm manager classlarının birbirleriyle olan iletişimini sağlayan ve sistemin veri bütünlüğünü yöneten merkezi sınıftır.

Functions

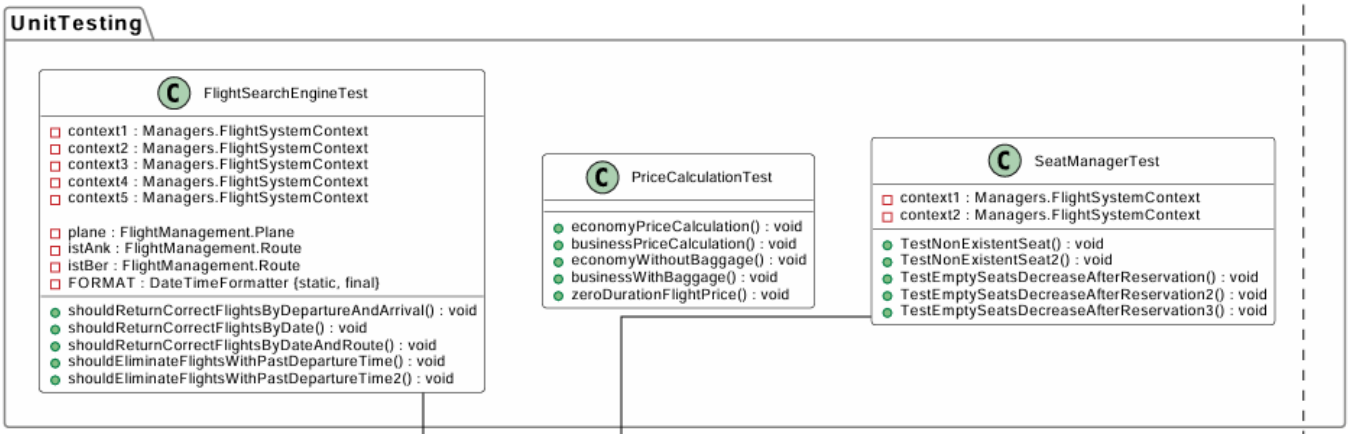
1. **setContexts:** Oluşturulan tüm Manager sınıflarına, merkezi iletişim noktası olan Context yapısının (kendisinin) tanıtılmasını sağlar. Böylece Manager'lar birbirlerine bu yapı üzerinden erişebilir.
2. **setCalculatePrice:** Fiyat hesaplama modülünün veya parametrelerinin güncellenmesini sağlar.
3. **refreshFleetData:** Hafızadaki uçak verilerini temizler ve FleetService aracılığıyla dosyadan güncel uçak ve koltuk verilerini tekrar yükler.
4. **refreshFlightData:** Hafızadaki uçuş verilerini temizler ve FlightService aracılığıyla dosyadan güncel uçuş verilerini tekrar yükler.
5. **refreshBookingData:** Rezervasyon ve bilet verilerini temizler, ardından ReservationService ve TicketService aracılığıyla dosyadan güncel verileri yükler.

6. **refreshAllData:** Filo, uçuş ve rezervasyon verilerinin tamamını sırasıyla yenileyerek sistemin dosya sistemiyle tam senkronize hale gelmesini sağlar.
7. **get[ManagerName]:** İlgili Manager sınıfının (PlaneManager, FlightManager vb.) örneğini döndürerek diğer sınıfların kullanımına sunar.



3. Unit Testing

Yazılan sistemlerin doğru çalışıp çalışmadığını test edebilmek için Unit Test uygulanmıştır.



3.1. Flight Search Engine Test

Flight Manager içerisinde yer alan uçuş arama sistemlerinin doğruluğunu test etmek için bu testler uygulanmıştır. Rota ve tarihe göre yapılan aramalarda doğru sonuçların dönüp dönmediğine, tarihi geçen uçuşların sistem tarafından gösterilmediğine yönelik testler yapılmıştır.

```
istAnk = new Route("Istanbul", "Ankara");
istBer = new Route("Istanbul", "Berlin");
planeManager.createPlane("F35", seating);

flightManager.createFlight("PL-1", istAnk, "01/06/2026", 12, 90);
    flightManager.createFlight("PL-1", istBer, "01/06/2026", 10, 60);


    List<Flight> results =
        flightManager.searchFlightsByRoute("Istanbul", "Ankara");

    assertEquals(1, results.size());
    assertEquals("Ankara" , results.get(0).getRoute().getArrivalPlace());
```

3.1.1. shouldReturnCorrectFlightsByDepartureAndArrival

Rotaya göre arama yapıldığında sadece aranan rotadaki uçuşların döndüğü kontrol edilmiştir.

```
istAnk = new Route("Istanbul", "Ankara");
istBer = new Route("Istanbul", "Berlin");
planeManager.createPlane("F35", seating);

flightManager.createFlight("PL-1", istAnk, "01/06/2026", 12, 90);
    flightManager.createFlight("PL-1", istBer, "05/10/2026", 10, 60);


    List<Flight> results =
        flightManager.searchFlightsByDate("05/10/2026");
    assertEquals(1, results.size());
    assertEquals("Berlin" , results.get(0).getRoute().getArrivalPlace());
```

3.1.2. shouldReturnCorrectFlightsByDate

Tarihe göre arama yapıldığında sadece o tarihteki uçuşlar liste olarak geri alındığından bu örnekte sadece İstanbul-Berlin uçuşu dönmüştür.

```
istAnk = new Route("Istanbul", "Ankara");
istBer = new Route("Istanbul", "Berlin");
planeManager.createPlane("F35", seating);

flightManager.createFlight("PL-1", istAnk, "01/06/2026", 12, 90);
flightManager.createFlight("PL-1", istAnk, "04/06/2026", 12, 90);
    flightManager.createFlight("PL-1", istBer, "05/10/2026", 10, 60);


    List<Flight> results =
        flightManager.searchFlights("Istanbul", "Ankara", "01/06/2026");
    assertEquals(1, results.size());
    assertEquals("Ankara" , results.get(0).getRoute().getArrivalPlace());
    assertEquals("01/06/2026" , results.get(0).getDate());
```

3.1.3. shouldReturnCorrectFlightsByDateAndRoute

Tarih ve rotaya göre arama yapıldığında aynı rotada birden farklı uçuş olsa da sadece aranan tarihteki uçuş sonuç olarak gelmiştir.

```
istAnk = new Route("Istanbul", "Ankara");
istBer = new Route("Istanbul", "Berlin");
planeManager.createPlane("F35", seating);

flightManager.createFlight("PL-1", istAnk, "01/06/2024", 12, 90);
flightManager.createFlight("PL-1", istAnk, "04/06/2023", 12, 90);
    flightManager.createFlight("PL-1", istBer, "05/10/2026", 10, 60);


    List<Flight> results =
        flightManager.searchFlightsByRoute("Istanbul", "Ankara");
    assertEquals(0, results.size());
```

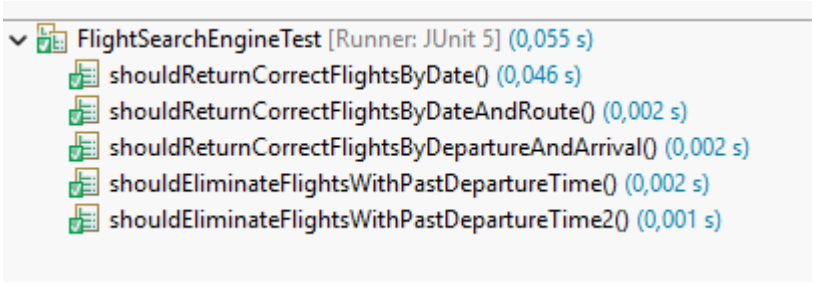
3.1.4. shouldEliminateFlightsWithPastDepartureTime

Oluşturulan uçuşların rotaları arananla eşleşse bile tarihleri güncel tarihten eski yani uçuşların süresi geçtiği için bunlar sonuç olarak gelmemiştir. Yani sonuç boyutu 0'dır.

```
flightManager.createFlight("PL-1", istAnk, "01/06/2024", 12, 90);
flightManager.createFlight("PL-1", istAnk, "04/06/2023", 12, 90);
    flightManager.createFlight("PL-1", istBer, "05/10/2002", 10, 60);


    List<Flight> results =
        flightManager.searchFlightsByRoute("Istanbul", "Berlin");
    assertEquals(0, results.size());
```

3.1.5. shouldEliminateFlightsWithPastDepartureTime2

Aynı sonuç burada da görülebilmektedir.



Yapılan JUNIT testlerinin sonuçlarında beklenen ve gerçek sonuçların aynı olduğu görülmektedir.

3.2. PriceCalculation Test

CalculatePrice sınıfında yazılan parametrelere göre rezervasyon ve bilet fiyatlarının doğru hesaplanıp hesaplanmadığına yönelik testler uygulanmıştır.

```
Seat seat = new Seat("A1", FlightClass.ECONOMY);

Plane plane = new Plane("PL1", "A320", new Seating(null, new Seat[][]{{seat}}));

Flight flight = new Flight(
    1, plane, new Route("IST", "ANK"), "2026-01-01", 10, 60 // 60 mins duration
);

Passenger p = new Passenger("P1", "A", "B", "m", "p", "c");
Reservation r = new Reservation("R1", flight, p, seat, "2026");

CalculatePrice calculatePrice = new CalculatePrice(500, 50, 15, 2.5, 1.5);
double price = calculatePrice.priceCalculation(r, 10);

double expected =
      500.0
    + (60 * 1.5)
    + (10 * 15.0)
    + 50.0;

assertEquals(expected, price, 0.01);
```

3.2.1. economyPriceCalculation

Verilen başlangıç fiyatı , vergi fiyatı , oluşturulan ekonomi koltuğunun fiyatı gibi etmenler göz önüne alınarak rezervasyonun fiyatı hesaplandığında beklenen değerle aynı olup olmadığı kontrol edilmiştir

3.2.2. BusinessPriceCalculation

Aynı örnek business koltuk ile tekrarlanmıştır.

3.2.3. EconomyWithoutBaggage

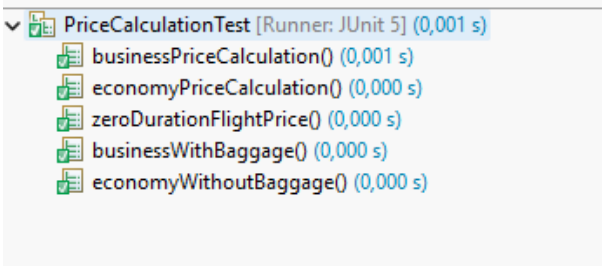
Aynı örnek bir ekonomi koltuğunda bagaj ağırlığı 0 yani bagajsız şekilde test edilmiştir.

3.2.4. BusinessWithBaggage

Bagajlı bir business koltuk için fiyat hesaplaması test edilmiştir.

3.2.5. ZeroDurationFlightPrice

Süre olarak 0 girilen bir uçuşun fiyatı test edilmiştir.



Yapılan JUNIT testlerinin sonuçlarında beklenen ve gerçek sonuçların aynı olduğu görülmektedir.

3.3. SeatManager Test

Rezervasyon oluştururken koltukların durum değişikliklerinin uygulanıp uygulanmadığına ve olmayan bir koltuğun rezerve edilip edilemediğine yönelik testler uygulanmaktadır.

```
    int[][] seatPlan = {{2,2},{2,2}};
    Seating seating = seatManager.createSeatings(seatPlan);

    Plane plane = new Plane("PL1", "Boeing", seating);

    Flight flight = new Flight(
        1,
        plane,
        new Route("IST", "ANK"),
        "2026/01/01",
        10,
        60
    );

    Passenger passenger = new Passenger(
        "P1", "A", "B", "m", "p", "c"
    );

    Seat fakeSeat = new Seat("Z99", FlightClass.BUSINESS);

    assertThrows(
            IllegalArgumentException.class,
        () -> reservationManager.createReservation(
            flight,
            passenger,
            fakeSeat
        )
    );
```

3.3.1. TestNonExistentSeat

Olmayan bir koltuk alınmaya çalışıldığında Illegal Argument Exception hatasının geleceği tahmin edilip testi yapılmıştır.

3.3.2. TestNonExistentSeat2

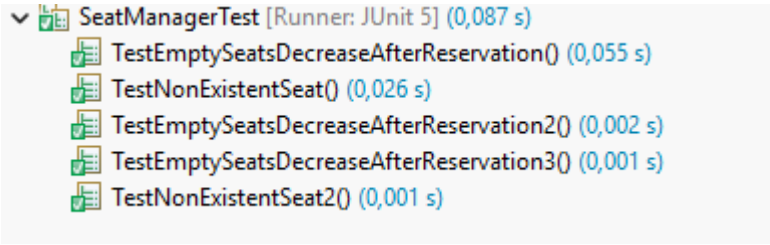
Aynı test farklı bir olmayan koltuk numarası ile tekrar edilmiştir.

```
    int[][] seatPlan = {{0,0},{2,2}}; // TOPLAM 4 YER VAR
    Seating seating = seatManager.createSeatings(seatPlan);

    Plane plane = new Plane("PL2", "Airbus", seating);

    Flight flight = new Flight(
        2,
        plane,
        new Route("IST", "IZM"),
        "2026/01/02",
        12,
        90
    );

    Passenger passenger = new Passenger("P2","X","Y","m","p","c");

    int[] beforeArray = seatManager.calculateAvailableSeats(plane);
    int before = beforeArray[0] + beforeArray[1];

    Seat seatToReserve = flight.getRandomAvailableEconomySeat();

    reservationManager.createReservation(
        flight,
        passenger,
        seatToReserve
    ); // TOPLAM 4 YER VARDI 1 TANESİ REZERVE OLDU 3 KALDI


    int[] afterArray = seatManager.calculateAvailableSeats(plane);
    int after = afterArray[0] + afterArray[1];

    assertEquals(3, after);
```

3.3.3. TestEmptySeatsDecreaseAfterReservation: Toplam kapasitesi 4 olan bir uçak oluşturulup 1 rezervasyon yapıldığında 3 yer kaldığı tahmin edilmiş ve bunun testi yapılmıştır.

3.3.4. TestEmptySeatsDecreaseAfterReservation2: Aynı örnek toplam kapasitesi 50 olan bir uçak ve 5 rezervasyon için tekrarlanmıştır.

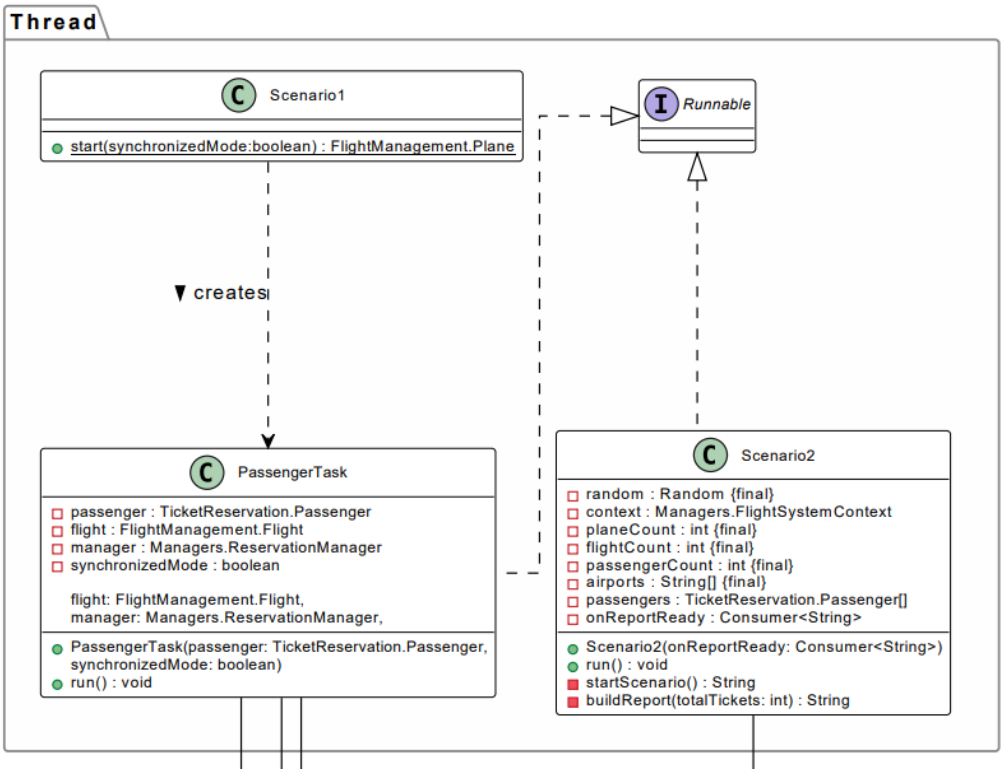
3.3.5. TestEmptySeatsDecreaseAfterReservation3: Aynı örnek toplam kapasitesi 50 olan bir uçak ve 10 rezervasyon için tekrarlanmıştır.



Yapılan JUNIT testlerinin sonuçlarıyla beklenen değerler aynıdır.

4. Senaryolar

Yapılması gereken senaryolar admin menüsünün içine entegre edilmiş ve GUI üzerinden başlatılabilir hale getirilmiştir.



4.1. Scenario1 Class:

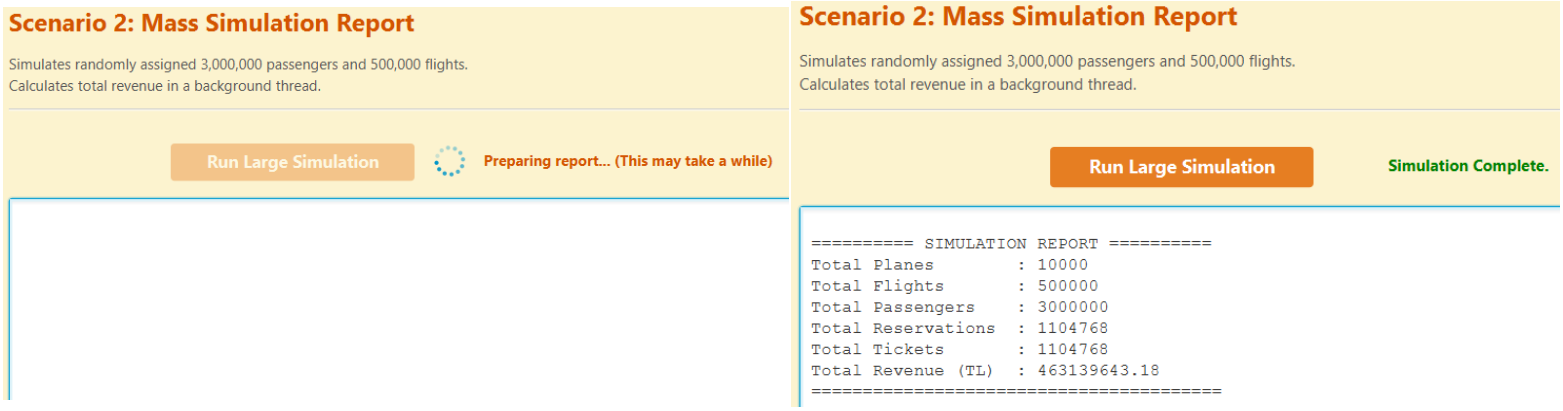
Bu class toplam kapasitesi 180 olan bir uçak ve 90 tane de yolcu üretmektedir. Bu yolcuları az sonra gösterilen PassengerTask adlı Runnable class'a atayarak Threadlar üretmekte ve bu threadları çalıştırdıktan sonra uçakta kalan boş yer miktarını belirlemektedir.

4.2. PassengerTask Class:

Bu class , kendisine gönderilen yolcunun gönderilen uçuştan rezervasyon almasını Runnable interface sayesinde run fonksiyonunun içinde yapmasını sağlar. Ayrıca Scenario 1 den buraya gönderilen senkron booleanı ile bu işlemin senkron mu yoksa asenkron mu olacağı burada belirlenir.

4.3. Scenario2 Class:

Uzun süren bir işlemi simüle edebilmek için 10000 uçak , 50000 uçuş ve 300000 yolcu rastgele biçimlerde üretilmektedir. Her uçağın kapasitesi rastgele belirlenmekte , her uçuşun rotası da belli rotalar arasında rastgele olarak seçilmektedir. Her yolcu bu uçuşlardaki koltuklardan rastgele bir tanesini rezerve etmekte ve rastgele bir bagaj ağırlığı ile bilet oluşturmaktadır. Bu işlemlerin hepsi düşünüldüğünde büyük bir işlem yükü çıkmaktadır. En sonunda da toplam istatistikler ekrana yazmakta ve toplam oluşturulan biletlerden kazanılan para hesaplanarak yazılmaktadır.

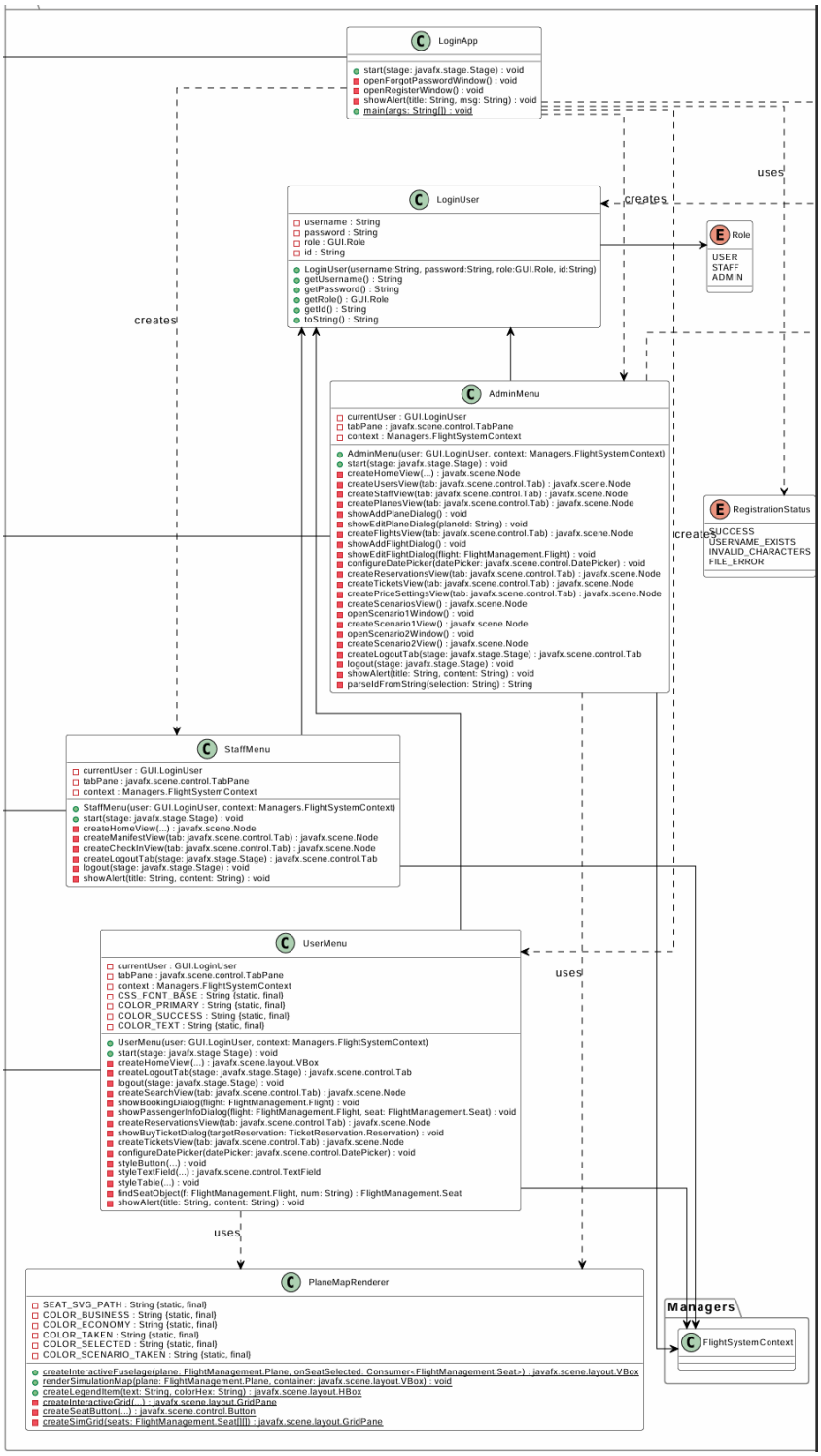


5. Kullanıcı Arayüzü

Bu bölüm, projenin içindeki sistemlerin kullanıcı tarafından etkileşime geçilebilir olmasını sağlayan sınıflardan oluşur. GUI paketi altındaki sınıflar, kullanıcıdan alınan girdileri doğrular ve işlenmek üzere ilgili Manager sınıflarına iletir. Sistem, giriş yapan kullanıcının rolüne göre tasarlanmış olup, her kullanıcı tipi (Yönetici, Personel, Müşteri) için ayrıştırılmış arayüzler sunar. Ayrıca Services paketi sayesinde yapılan işlemleri dosyalara yazarak sisteme işlenen verilerin kaydedilmesini sağlar ve işlem yapılacağı sırada verileri dosyadan yükleyerek veri kaybını veya çakışmalarını önler.

5.1. GUI Paketi

Projenin görsel arayüzünü oluşturan sınıfları içerir.



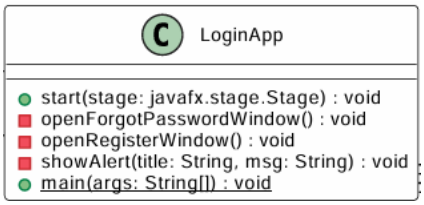
5.1.1. LoginApp Class

LoginApp class'ı, sistemin giriş (login), kayıt (register) ve şifre sıfırlama işlemlerini yöneten JavaFX tabanlı arayüz sınıfıdır.Kullanıcıdan alınan bilgiler AuthService aracılığıyla doğrulanır ve kullanıcı rolüne göre ilgili menüye yönlendirme yapılır.

Functions

1. **start**: Giriş ekranının ana arayüzünü oluşturur. Kullanıcı adı ve şifre alanlarını, giriş butonunu, kayıt olma butonunu ve şifremi unuttum bağlantısını tanımlar. Kullanıcının rolüne göre AdminMenu, StaffMenu veya UserMenu ekranına yönlendirme yapar.

2. **openForgotPasswordWindow**: Şifre sıfırlama penceresini açar. Kullanıcıdan önce kullanıcı adını alır, ardından ilgili güvenlik sorusunu gösterir. Girilen güvenlik cevabı doğruysa yeni şifrenin belirlenmesini sağlar ve AuthService üzerinden şifre güncellemesini gerçekleştirir.
3. **openRegisterWindow**: Yeni kullanıcı kayıt ekranını oluşturur. Kullanıcıdan kullanıcı adı, şifre, hesap türü (User/Staff) ve güvenlik sorusu bilgilerini alır. Girilen bilgilere göre yeni bir LoginUser oluşturur ve AuthService aracılığıyla sisteme kaydeder.
4. **showAlert**: Kullanıcıya bilgi veya hata mesajlarını göstermek için genel amaçlı uyarı penceresi oluşturur.
5. **main**: JavaFX uygulamasını başlatan ana metottur.



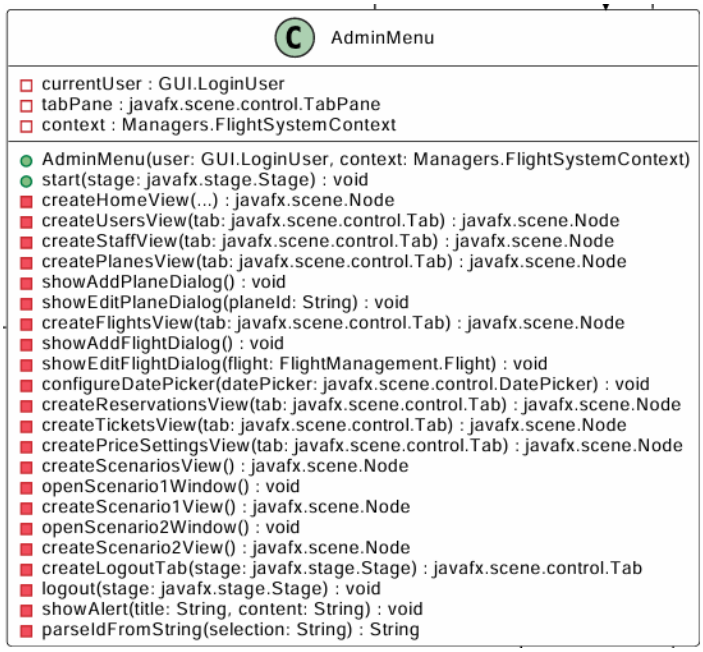
5.1.2. AdminMenu Class

AdminMenu class'ı, sistemin yönetici (admin) arayüzünü temsil eder. Uçak, uçuş, rezervasyon, bilet, kullanıcı, personel ve fiyatlandırma yönetiminin grafik arayüz üzerinden yapılmasını sağlar.Tüm işlemler ilgili Manager class'ları aracılığıyla gerçekleştirilir.

Functions

1. **AdminMenu:** Yönetici panelini başlatır.Gerekli tüm Manager class'larını oluşturur, birbirleriyle ilişkilendirir ve sistemde kayıtlı uçak, uçuş, rezervasyon ve bilet verilerini dosyalardan yükler.
2. **createHomeView:** Yönetici ana ekranını oluşturur. Admin'in sistemdeki ana modüllere (uçak, rezervasyon, bilet, kullanıcı, personel) hızlı erişim sağlaması için yönlendirme butonlarını içerir.
3. **createUsersView:** Sistemde kullanıcı hesaplarını listeleyen ekranı oluşturur. Admin'in kullanıcı hesaplarını görüntülemesini ve gerektiğinde kullanıcı silme işlemi yapmasını sağlar.
4. **createStaffView:** Sistemdeki admin ve staff kullanıcılarını listeleyen ekranı oluşturur. Admin'in personel hesaplarını görüntülemesini ve yetkili kullanıcılar için silme işlemi yapmasını sağlar. Kendi hesabının veya ana admin hesabının silinmesini engelleyen güvenlik kontrolleri içerir.
5. **createPlanesView:** Uçak filosunun yönetildiği ekranı oluşturur.Sistemdeki uçakları listeler; uçak ekleme, düzenleme ve silme işlemlerinin yapılmasını sağlar.Uçak silme işlemi sırasında bu uçağa bağlı uçuş, rezervasyon ve biletlerin de güncellenmesini tetikler.
6. **createFlightsView:** Uçuş planlarının yönetildiği ekranı oluşturur.Sistemde tanımlı uçuşları listeler; yeni uçuş ekleme, uçuş bilgilerini düzenleme ve uçuş iptal etme işlemlerini sağlar.
7. **createReservationsView:** Sistemde oluşturulmuş tüm rezervasyonları listeleyen ekranı oluşturur.Admin'in aktif rezervasyonları görüntülemesini ve seçilen rezervasyonları iptal etmesini sağlar.Rezervasyon iptali sırasında ilgili biletlerin de silinmesi tetiklenir.
8. **createTicketsView**: Sistemde oluşturulmuş tüm biletleri listeleyen ekranı oluşturur. Admin'in biletleri görüntülemesini ve gerektiğinde bilet iptal işlemi yapmasını sağlar.
9. **createPriceSettingsView:** Sistemin global fiyatlandırma parametrelerini yöneten ekranı oluşturur.Taban fiyat, uçuş süresi ücreti, bagaj ücreti, vergi ve business class çarpanı gibi değerlerin görüntülenmesini ve güncellenmesini sağlar.
10. **createScenariosView**: Simülasyon senaryolarının (Scenario 1 ve Scenario 2) çalıştırılacağı butonları içeren "Scenarios" sekmesini oluşturur.

11. **openScenario1Window**: 1. Senaryo (Eşzamanlı koltuk rezervasyonu simülasyonu) için yeni bir pencere (Stage) açar.
12. **createScenario1View**: 1. Senaryo penceresinin içeriğini oluşturur. Senkronize/Asenkron mod seçimini, uçak görselleştirmesini (PlaneMapRenderer) ve simülasyonu başlatma kontrollerini içerir.
13. **openScenario2Window**: 2. Senaryo (Büyük veri simülasyonu) için yeni bir pencere açar.
14. **createScenario2View**: 2. Senaryo penceresinin içeriğini oluşturur. Arka planda çalışan "Scenario2" thread'ini başlatır ve sonuç raporunu ekrana basar.

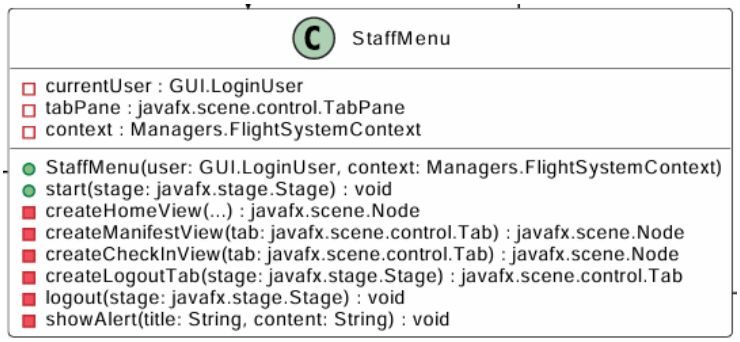


5.1.3. StaffMenu Class

StaffMenu class'ı, sistemde görev yapan personelin kullanabildiği JavaFX tabanlı arayüz sınıfıdır.Personelin yolcu listelerini görüntülemesini, bilet doğrulama ve boarding işlemlerini gerçekleştirmesini sağlar. Tüm işlemler ilgili Manager ve Service class'ları aracılığıyla dosya tabanlı verilerle senkron şekilde yürütülür.

Functions

1. **StaffMenu**: Giriş yapan personel kullanıcısını alır, gerekli Manager class'larını başlatır ve sistemdeki tüm verileri başlangıçta dosyalardan yükler.
2. **start**: Personel panelinin ana arayüzünü oluşturur. Dashboard, yolcu listeleri ve check-in sekmelerini tanımlar ve sahneyi ekranda gösterir.
3. **createHomeView**: Personel ana ekranını oluşturur. Yolcu listesi görüntüleme ve check-in ekranlarına yönlendiren butonları içerir.
4. **createManifestView**: Seçilen uçuşa ait yolcu manifestosunu görüntüleyen ekranı oluşturur. FlightManager üzerinden uçuşları listeler ve ReservationManager üzerinden ilgili uçuşa ait yolcu rezervasyonlarını tablo halinde gösterir.
5. **createCheckInView**: Bilet doğrulama ve boarding işlemlerinin yapıldığı ekranı oluşturur. Girilen bilet numarasını TicketManager üzerinden kontrol eder ve bilet geçerliyse yolcu, uçuş, koltuk ve bagaj bilgilerini gösterir.
6. **createLogoutTab:** TabPanel'e çıkış işlevini tetikleyen özel bir sekme ekler.
7. **logout:** Oturumu sonlandırıp giriş ekranına yönlendirir.
8. **showAlert:** Kullanıcıya bilgilendirme ve hata mesajları gösteren pop-up pencere oluşturur.

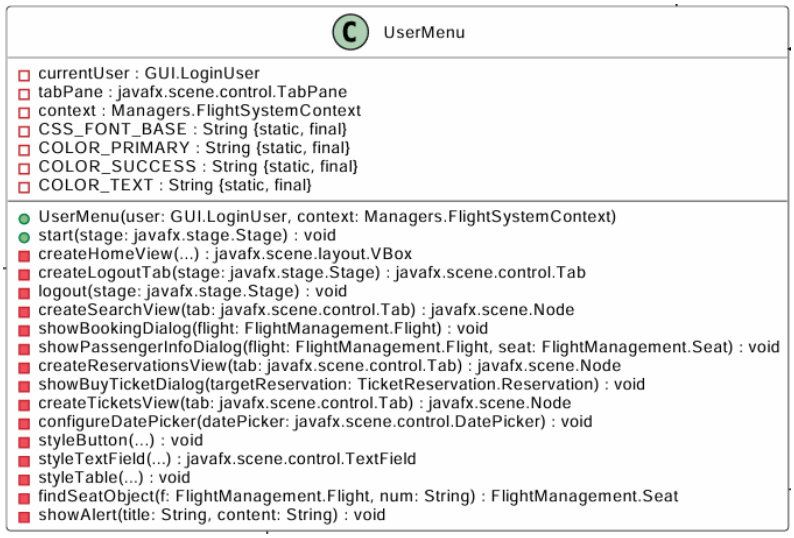


5.1.4. UserMenu Class

UserMenu class'ı, sisteme giriş yapan normal kullanıcıların uçuş arama, rezervasyon oluşturma, bilet satın alma ve kendi işlemlerini görüntüleme süreçlerini yöneten JavaFX tabanlı arayüz sınıfıdır.Kullanıcıya yalnızca kendisine ait rezervasyon ve bilet bilgilerine erişim yetkisi verir ve tüm işlemleri ilgili Manager ve Service class'ları üzerinden gerçekleştirir.

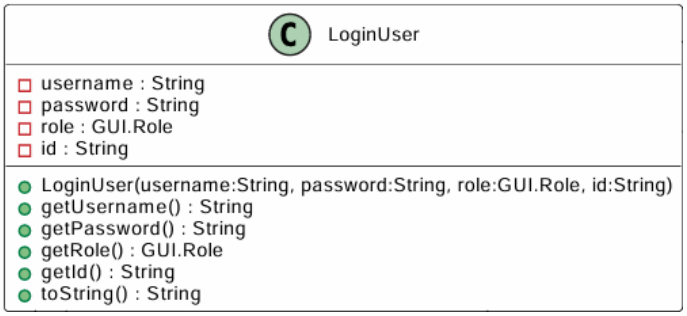
Functions

1. **UserMenu**: Sisteme giriş yapan kullanıcıyı alır, gerekli Manager class'larını başlatır, fiyat hesaplama bileşenini yükler ve tüm verileri başlangıçta dosyalardan sisteme aktarır.
2. **start**: Kullanıcı panelinin ana arayüzünü oluşturur. Ana sayfa, uçuş arama, rezervasyonlarım ve biletlerim sekmelerini tanımlar ve sahneyi ekranda gösterir.
3. **createHomeView**: Kullanıcı ana ekranını oluşturur. Uçuş arama, rezervasyon görüntüleme, bilet görüntüleme ve çıkış işlemleri için yönlendirme butonlarını içerir.
4. **createSearchView**: Uçuş arama ekranını oluşturur. Kullanıcının kalkış yeri, varış yeri ve tarih kriterlerine göre uçuşları aramasını sağlar. Arama sonuçlarını tablo halinde gösterir ve seçilen uçuş için koltuk seçimi ile rezervasyon oluşturma sürecini başlatır.
5. **showBookingDialog**: Seçilen uçuş için görsel koltuk haritası üzerinden koltuk seçilmesini sağlar. Koltuk sınıfına göre bilgilendirme yapar ve rezervasyon işlemini senkron kontrollerle güvenli şekilde gerçekleştirir.
6. **createReservationsView**: Kullanıcının kendisine ait aktif rezervasyonlarını listeleyen ekranı oluşturur. Kullanıcı bu ekrandan rezervasyon iptali yapabilir veya ilgili rezervasyon için bilet satın alma sürecini başlatabilir.
7. **showBuyTicketDialog**: Seçilen rezervasyon için bagaj ağırlığı bilgisini alır, bilet fiyatını önceden hesaplar ve kullanıcı onayı sonrasında bilet oluşturma işlemini gerçekleştirir.
8. **createTicketsView**: Kullanıcının satın aldığı tüm biletleri listeleyen ekranı oluşturur. Sadece kullanıcıya ait biletler görüntülenir.
9. **showPassengerInfoDialog:** Rezervasyon sürecinin ikinci adımıdır (showBookingDialog'dan sonra çalışır). Kullanıcıdan yolcu bilgilerini (isim, soyisim, e-posta, telefon, adres) alır. "Confirm & Book" butonuna basıldığında koltuğun hala boş olup olmadığını son kez kontrol eder (senkronizasyon) ve ReservationManager üzerinden rezervasyonu veritabanına kaydeder.
10. **createLogoutTab:** TabPane yapısının en sağına çıkış yapma işlevini tetikleyen "Logout" sekmesini ekler.
11. **logout:** Mevcut kullanıcı oturumunu kapatır, pencereyi sonlandırır ve LoginApp ekranını yeniden başlatarak güvenli çıkış sağlar.



5.1.5. LoginUser Class

LoginUser class'ı, sistemde oturum açan bir kullanıcıyı temsil eden veri sınıfıdır.
Kullanıcının kullanıcı adı, şifre bilgisi, rolü (ADMIN, STAFF, USER) ve kullanıcı numarasını saklar.
GUI ve AuthService katmanlarında kullanıcı bilgilerine erişim bu class üzerinden sağlanır.

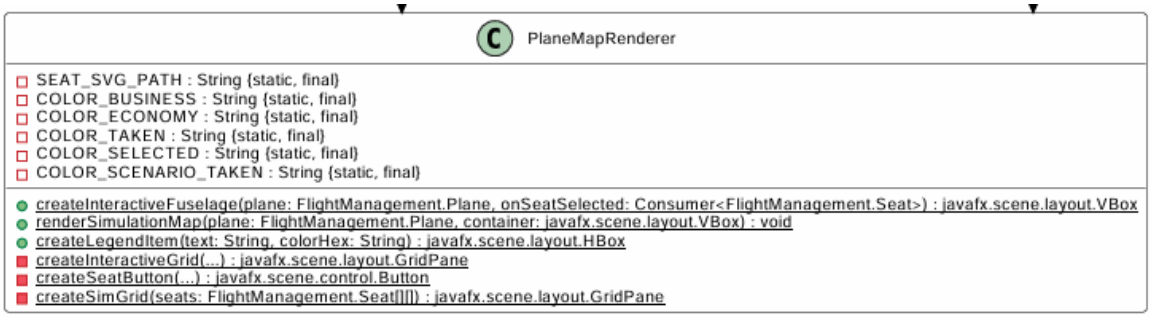


5.1.6. PlaneMapRenderer Class

Uygulamanın grafiksel arayüzünde uçak oturma düzenini dinamik olarak çizen ve görselleştiren yardımcı (utility) sınıftır. JavaFX kütüphanesi ve SVG (Scalable Vector Graphics) yollarını kullanarak, uçak nesnesinin koltuk matrisi yapısını etkileşimli bir koltuk haritasına dönüştürür. UserMenu için etkileşimli seçim ekranları ve Scenario1 için anlık simülasyon görüntüleri üretir.

1. **createInteractiveFuselage:** Verilen uçak nesnesinin (Plane) Business ve Ekonomi sınıfı koltuk matrislerini analiz eder. Uçağın gövdesini temsil eden dikey bir VBox oluşturur.
2. **createInteractiveGrid:** Koltuk matrisini (Seat[][]) tarayarak bir GridPanel oluşturur. Satır numaralarını, koridor boşluklarını hesaplar ve her bir hücreye etkileşimli koltuk butonlarını yerleştirir.
3. **createSeatButton:** Tek bir koltuk için özelleştirilmiş bir buton oluşturur. Koltuğun durumuna (Dolu/Boş) ve sınıfına (Business/Economy) göre SVG ikonunu renklendirir. Koltuk doluysa butonu pasif hale getirir, boşsa tıklama olayını (callback) dinler.
4. **renderSimulationMap:** Thread simülasyonu (Scenario 1) için uçağın anlık doluluk durumunu gösteren statik bir harita çizer. Etkileşim içermez, sadece görsel geri bildirim (Kırmızı: Dolu, Yeşil: Boş) sağlar.

5. **createSimGrid:** Simülasyon haritası yapısını oluşturur. Buton yerine sadece görsel ikonlar kullanarak performansı optimize eder.
6. **createLegendItem:** Haritanın altında yer alan renk kodlarını (Örn: Sarı=Business, Kırmızı=Dolu) açıklayan görsel lejant öğelerini oluşturur.

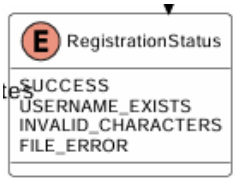


5.1.7. Role Enum

Uygulama içerisindeki kullanıcı yetkilendirme seviyelerini ve hesap türlerini tanımlayan enum yapısıdır. Sisteme giriş yapan bir LoginUser nesnesinin yetki kapsamını belirler ve LoginApp tarafından kullanıcının hangi arayüze (UserMenu, StaffMenu veya AdminMenu) yönlendirileceğine karar verilmesinde kullanılır.

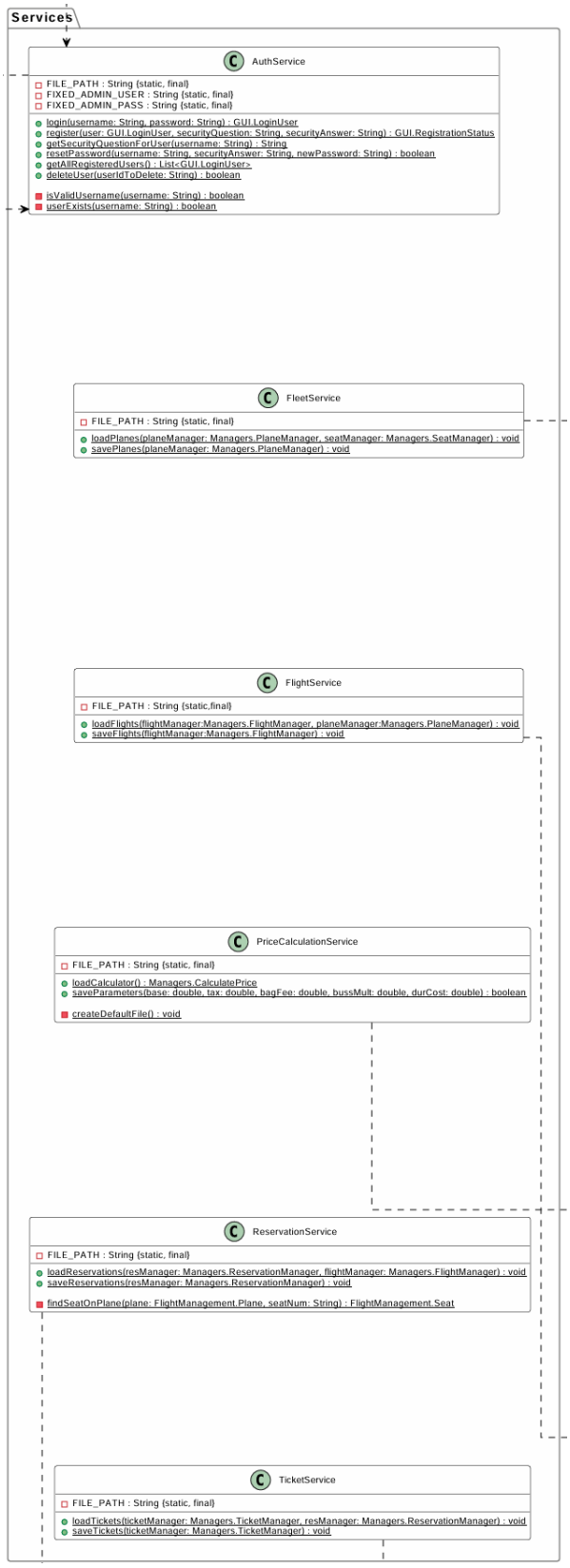
5.1.8. RegistrationStatus Enum

Uygulama içerisindeki yeni kullanıcı kayıt (register) işlemlerinin sonucunu ve durumunu temsil eden enum yapısıdır. AuthService sınıfı, kayıt işleminin başarısını veya başarısızlık nedenini bu yapı LoginApp'e bildirir. Bu sayede, kayıt sırasında oluşan hatanın türüne göre kullanıcıya özelleştirilmiş hata mesajları gösterilebilir.



5.2. Services Paketi

Services paketi, uçuş yönetim sisteminin veri saklamasını sağlayan ve dosya tabanlı veritabanı işlemlerini yürüten sınıfları barındırır. Bu katman, sistemdeki tüm operasyonel verilerin (kullanıcılar, uçaklar, uçuşlar, rezervasyonlar ve biletler) güvenli bir şekilde metin dosyalarına (.txt) kaydedilmesini ve sistem başlatıldığında bu dosyalar üzerinden verilerin tekrar yüklenmesini sağlar. Manager katmanıyla doğrudan entegre çalışan bu servisler, veri kaybını önler ve sistemin sürekliliğini garanti altına alır.



5.2.1. AuthService Class

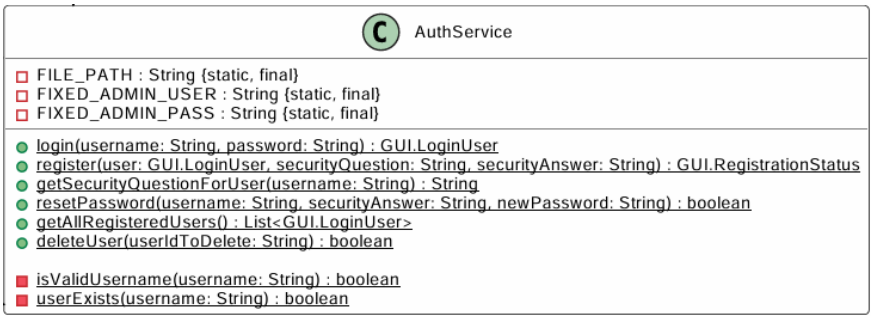
AuthService class'ı, sistemdeki kullanıcı girişi (login), kayıt (register), şifre sıfırlama ve kullanıcı yönetimi işlemlerini dosya tabanlı olarak yöneten servis sınıfıdır.

Kullanıcı bilgileri users.txt dosyasında saklanır ve tüm erişimler bu sınıf üzerinden yapılır.

Bu class statik metotlar içerir ve nesne oluşturulmadan kullanılır.

Functions

1. **login**: Girilen kullanıcı adı ve şifre bilgilerini kontrol eder. Öncelikle sabit tanımlı admin hesabını kontrol eder, eşleşme yoksa kullanıcı dosyasını okuyarak doğrulama yapar. Başarılı giriş durumunda LoginUser nesnesi döndürür, başarısız durumda null döndürür.
2. **register**: Yeni bir kullanıcıyı sisteme kaydeder. Kullanıcı adının geçerliliğini ve daha önce kayıtlı olup olmadığını kontrol eder. Güvenlik sorusu ve cevabı ile birlikte kullanıcıyı dosyaya yazar ve işlem sonucunu RegistrationStatus enum'u ile döndürür.
3. **getSecurityQuestionForUser**: Verilen kullanıcı adına ait güvenlik sorusunu dosyadan bularak döndürür. Kullanıcı bulunamazsa null döndürür. Şifre sıfırlama senaryosunda kullanılır.
4. **resetPassword**: Verilen kullanıcı adı için güvenlik cevabını kontrol eder. Cevap doğruysa kullanıcının şifresini günceller ve dosyayı yeniden yazar. İşlemin başarılı olup olmadığını boolean olarak döndürür.
5. **getAllRegisteredUsers**: Sistemde kayıtlı tüm kullanıcıları users.txt dosyasından okuyarak bir LoginUser listesi şeklinde döndürür. Admin panelinde kullanıcı ve personel listeleme işlemleri için kullanılır.
6. **deleteUser**: Verilen kullanıcı ID'sine sahip kullanıcıyı sistemden siler. Kullanıcı dosyasını yeniden yazarak ilgili kullanıcıyı kalıcı olarak kaldırır. Silme işleminin başarılı olup olmadığını boolean olarak döndürür.

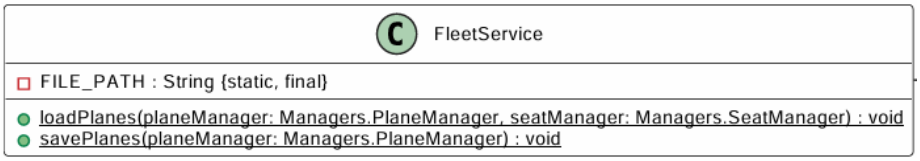


5.2.2. FleetService Class

FleetService class'ı, sistemdeki uçakların dosya tabanlı olarak okunmasını ve kaydedilmesini sağlayan servis sınıfıdır.Uçak verileri planes.txt dosyasında saklanır ve bu class, PlaneManager ile dosya sistemi arasında köprü görevi görür.Bu class statik metotlar içerir ve nesne oluşturulmadan kullanılır.

Functions

1. **loadPlanes**: planes.txt dosyasını satır satır okuyarak kayıtlı uçak bilgilerini sisteme yükler. Her satırdan uçak ID'si, model bilgisi ve business/economy koltuk düzeni bilgilerini alır. SeatManager aracılığıyla koltuk düzenini yeniden oluşturur ve oluşturulan Plane nesnesini PlaneManager'a ekler. Dosyada kayıtlı uçak ID'lerinin korunmasını sağlar.
2. **savePlanes**: PlaneManager içerisinde kayıtlı olan tüm uçakları alır ve mevcut uçak listesini planes.txt dosyasına yazar. Her uçak için model bilgisi ve business/economy koltuk matrislerinin satır–sütun boyutlarını dosya formatına uygun şekilde kaydeder. Dosyadaki eski veriler tamamen silinerek güncel uçak bilgileri yazılır.



5.2.3. FlightService Class

FlightService class'ı, sistemdeki uçuşların dosya tabanlı olarak okunmasını ve kaydedilmesini sağlayan servis sınıfıdır.Uçuş verileri flights.txt dosyasında saklanır ve bu class, FlightManager ile dosya sistemi arasında köprü görevi görür.Bu class statik metotlar içerir ve nesne oluşturulmadan kullanılır.

Functions

1. **loadFlights**: flights.txt dosyasını satır satır okuyarak kayıtlı uçuş bilgilerini sisteme yükler. Her satırdan uçuş numarası, uçak ID'si, kalkış–iniş bilgileri, tarih, saat ve uçuş süresi verilerini alır. PlaneManager üzerinden ilgili uçağı bulur, Route ve Flight nesnelerini yeniden oluşturur ve uçuşu FlightManager'a ekler. Uçağı bulunamayan uçuşlar sisteme dahil edilmez.
2. **saveFlights**: FlightManager içerisinde kayıtlı olan tüm uçuşları alır ve flights.txt dosyasına yazar. Her uçuş için uçuş numarası, bağlı olduğu uçak ID'si, rota bilgileri, tarih, saat ve süre bilgileri dosya formatına uygun şekilde kaydedilir. Dosyadaki eski veriler tamamen silinerek güncel uçuş bilgileri yazılır.

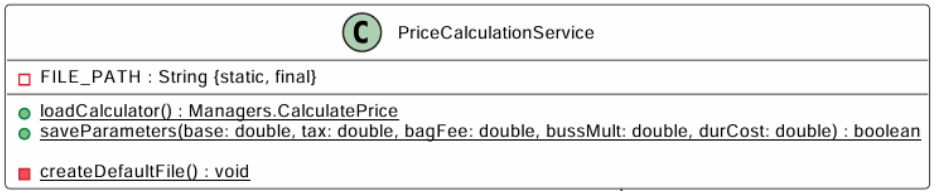
5.2.4. PriceCalculationService Class

PriceCalculationService class'ı, bilet fiyatlarının hesaplanmasında kullanılan fiyatlandırma parametrelerinin dosya tabanlı olarak okunmasını ve kaydedilmesini sağlayan servis sınıfıdır.

Fiyat parametreleri price_parameters.txt dosyasında saklanır ve CalculatePrice class'ının oluşturulması bu servis üzerinden gerçekleştirilir.Bu class statik metotlar içerir ve nesne oluşturulmadan kullanılır.

Functions

1. **loadCalculator**: Fiyat hesaplama için gerekli olan taban fiyat, vergi, bagaj ücreti, business class çarpanı ve uçuş süresi maliyeti parametrelerini dosyadan okur. Dosya mevcut değilse veya okuma hatası oluşursa varsayılan değerleri kullanır ve gerekli dosyayı oluşturur. Okunan veya varsayılan değerlerle yeni bir CalculatePrice nesnesi üretir.
2. **saveParameters**: Güncellenmiş fiyatlandırma parametrelerini price_parameters.txt dosyasına yazar. Taban fiyat, vergi, bagaj ücreti, business class çarpanı ve süre maliyeti değerlerini dosyada kalıcı olarak saklar. İşlemin başarılı olup olmadığını boolean olarak döndürür.



5.2.5. ReservationService Class

ReservationService class'ı, sistemdeki rezervasyonların dosya tabanlı olarak okunmasını ve kaydedilmesini sağlayan servis sınıfıdır. Rezervasyon verileri reservations.txt dosyasında saklanır ve bu class, ReservationManager ile dosya sistemi arasında köprü görevi görür.Bu class statik metotlar içerir ve nesne oluşturulmadan kullanılır.

Functions

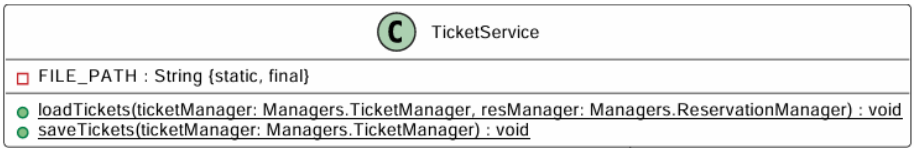
1. **loadReservations**: reservations.txt dosyasını satır satır okuyarak kayıtlı rezervasyon bilgilerini sisteme yükler. Her satırdan rezervasyon kodu, uçuş numarası, yolcu bilgileri, koltuk numarası ve rezervasyon tarihi verilerini alır. FlightManager üzerinden ilgili uçuşu bulur, uçağın içindeki doğru koltuğu tespit eder ve koltuğu rezerve edilmiş olarak işaretler. Yolcu ve rezervasyon nesneleri yeniden oluşturularak ReservationManager'a eklenir.
2. **saveReservations**: ReservationManager içerisinde kayıtlı olan tüm rezervasyonları alır ve reservations.txt dosyasına yazar. Her rezervasyon için rezervasyon kodu, uçuş numarası, yolcu bilgileri, koltuk numarası ve rezervasyon tarihi dosya formatına uygun şekilde kaydedilir. Dosyadaki eski veriler tamamen silinerek güncel rezervasyon bilgileri yazılır.

5.2.6. TicketService Class

TicketService class'ı, sistemdeki biletlerin dosya tabanlı olarak okunmasını ve kaydedilmesini sağlayan servis sınıfıdır.Bilet verileri tickets.txt dosyasında saklanır ve bu class, TicketManager ile dosya sistemi arasında köprü görevi görür. Bu class statik metotlar içerir ve nesne oluşturulmadan kullanılır.

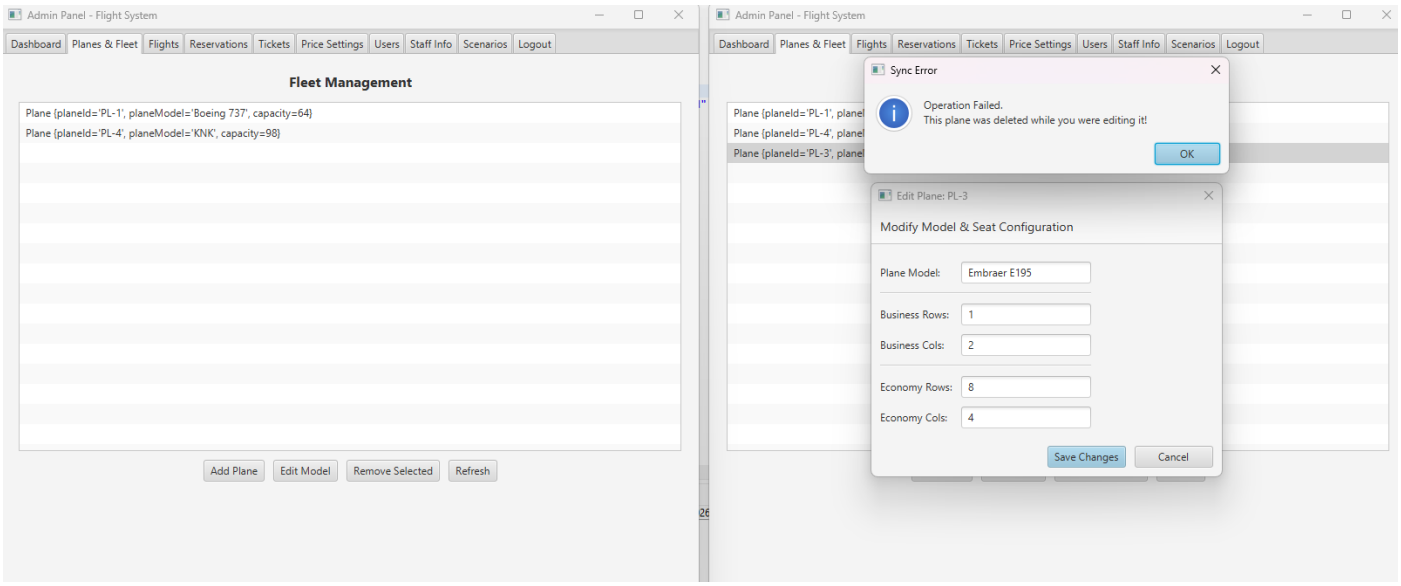
Functions

1. **loadTickets**: tickets.txt dosyasını satır satır okuyarak kayıtlı bilet bilgilerini sisteme yükler. Her satırdan bilet numarası, rezervasyon kodu, bilet fiyatı ve bagaj ağırlığı bilgilerini alır. ReservationManager üzerinden ilgili rezervasyonu bulur, Ticket nesnesini yeniden oluşturur ve TicketManager'a ekler.
2. **saveTickets**: TicketManager içerisinde kayıtlı olan tüm biletleri alır ve tickets.txt dosyasına yazar. Her bilet için bilet numarası, rezervasyon kodu, fiyat ve bagaj ağırlığı bilgileri dosya formatına uygun şekilde kaydedilir. Dosyadaki eski veriler tamamen silinerek güncel bilet bilgileri yazılır.

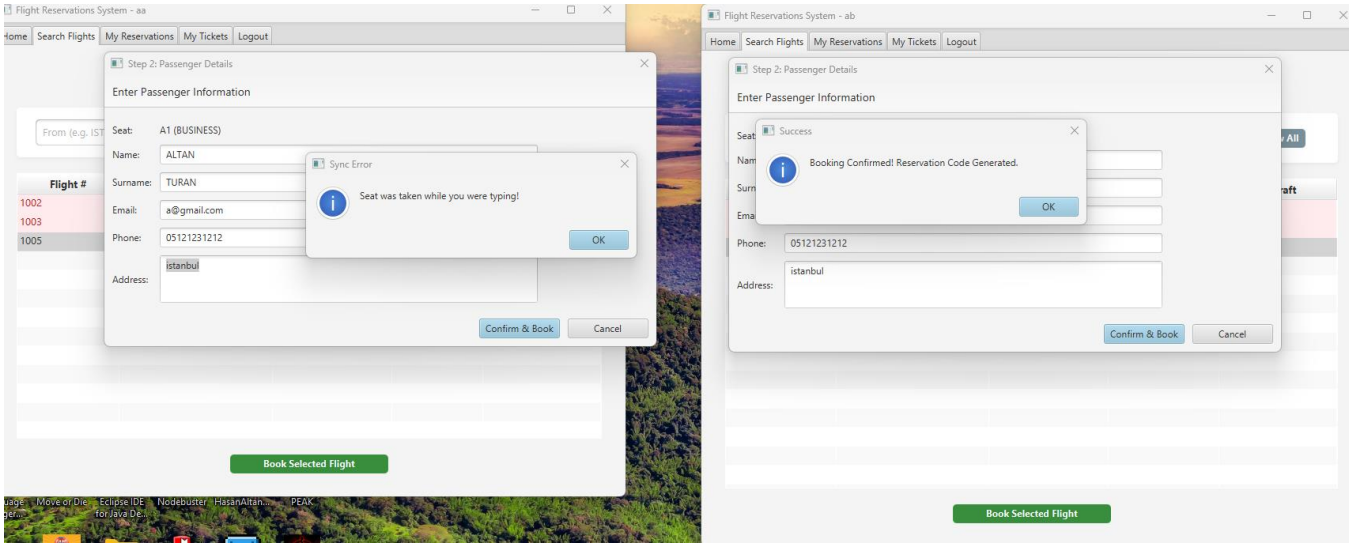


6. Edge Cases

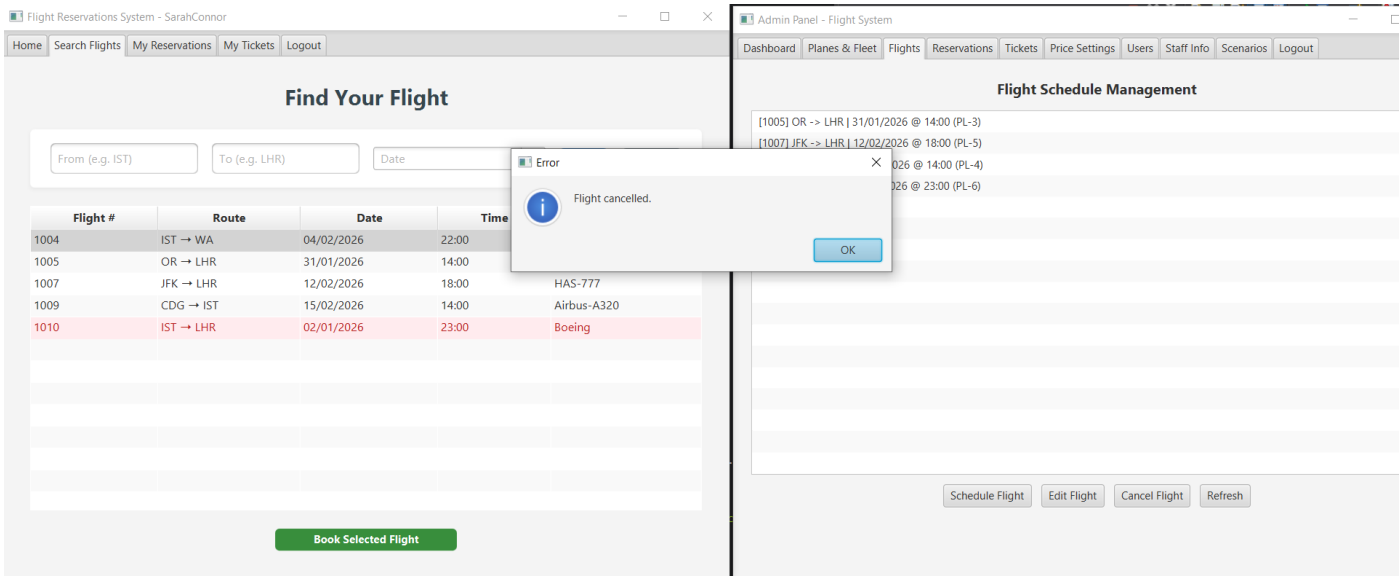
1) 2 Admin paneli açıksa ve bir admin uçağı düzenlemeye çalışırken diğeri silerse, düzenlemeye çalışan admin paneline uyarı olarak bu durum yansıtılır. Aynı durum aynı veriyi silmede, eklemede ve düzenlemede de geçerlidir.



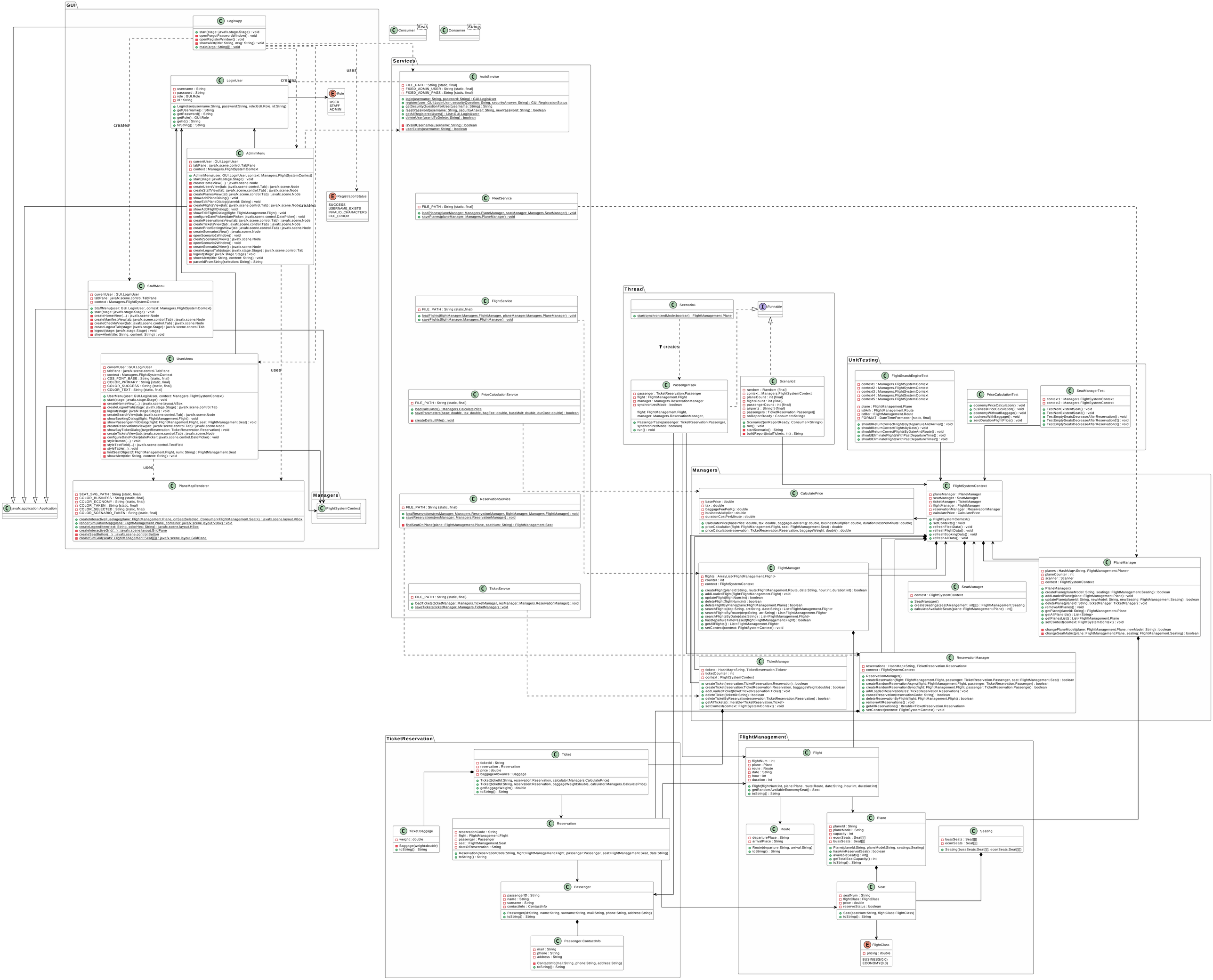
2) Aynı koltuğa 2 yolcu aynı anda rezervasyon yapmaya çalışırsa, ilk rezervasyon yapan yolcu koltuğu alabilirken sonradan deneyen yolcu rezervasyon yapmasına izin verilmeyecektir.



3) Eğer bir user uçuş listesini güncellemeden reservasyon yapmaya çalışırsa admin tarafından silinen bir uçuşu seçerse izin vermek yerine uçuşun iptal olduğunu söyleyip izin vermeyecektir.



4) Belirtilenlerden ayrıca birçok farklı durum test edilmiştir. Verilerin uyumluluğuna öncelikli önem verilmiştir. Test sürecinde fark edilen bütün hatalar giderilmeye çalışılmıştır.


GUI
LoginApp
start(stage: javafx.stage.Stage) : void
openForgotPasswordWindow() : void
openRegisterWindow() : void
showAlert(title: String, msg: String) : void
main(args: String[]) : void
Consumer
Seat
Consumer
String
uses
Services
LoginUser
username : String
password : String
role : GUI.Role
id : String
LoginUser(username:String, password:String, role:GUI.Role, id:String)
getUsername() : String
getPassword() : String
getRole() : GUI.Role
getId() : String
toString() : String
creates
Role
USER
STAFF
ADMIN
AuthService
FILE_PATH : String {static, final}
FIXED_ADMIN_USER : String {static, final}
FIXED_ADMIN_PASS : String {static, final}
login(username: String, password: String) : GUI.LoginUser
register(user: GUI.LoginUser, securityQuestion: String, securityAnswer: String) : GUI.RegistrationStatus
getSecurityQuestionForUser(username: String) : String
resetPassword(username: String, securityAnswer: String, newPassword: String) : boolean
getAllRegisteredUsers() : List<GUI.LoginUser>
deleteUser(userIdToDelete: String) : boolean
isValidUsername(username: String) : boolean
userExists(username: String) : boolean
AdminMenu
currentUser : GUI.LoginUser
tabPane : javafx.scene.control.TabPane
context : Managers.FlightSystemContext
AdminMenu(user: GUI.LoginUser, context: Managers.FlightSystemContext)
start(stage: javafx.stage.Stage) : void
createHomeView(...) : javafx.scene.Node
createUsersView(tab: javafx.scene.control.Tab) : javafx.scene.Node
createStaffView(tab: javafx.scene.control.Tab) : javafx.scene.Node
createPlanesView(tab: javafx.scene.control.Tab) : javafx.scene.Node
showAddPlaneDialog() : void
showEditPlaneDialog(planeId: String) : void
createFlightsView(tab: javafx.scene.control.Tab) : javafx.scene.Node
showAddFlightDialog() : void
showEditFlightDialog(flight: FlightManagement.Flight) : void
configureDatePicker(datePicker: javafx.scene.control.DatePicker) : void
createReservationsView(tab: javafx.scene.control.Tab) : javafx.scene.Node
createTicketsView(tab: javafx.scene.control.Tab) : javafx.scene.Node
createPriceSettingsView(tab: javafx.scene.control.Tab) : javafx.scene.Node
createScenariosView() : javafx.scene.Node
openScenario1Window() : void
createScenario1View() : javafx.scene.Node
openScenario2Window() : void
createScenario2View() : javafx.scene.Node
createLogoutTab(stage: javafx.stage.Stage) : javafx.scene.control.Tab
logout(stage: javafx.stage.Stage) : void
showAlert(title: String, content: String) : void
parseIdFromString(selection: String) : String
creates
RegistrationStatus
SUCCESS
USERNAME_EXISTS
INVALID_CHARACTERS
FILE_ERROR
FleetService
FILE_PATH : String {static, final}
loadPlanes(planeManager: Managers.PlaneManager, seatManager: Managers.SeatManager) : void
savePlanes(planeManager: Managers.PlaneManager) : void
StaffMenu
currentUser : GUI.LoginUser
tabPane : javafx.scene.control.TabPane
context : Managers.FlightSystemContext
StaffMenu(user: GUI.LoginUser, context: Managers.FlightSystemContext)
start(stage: javafx.stage.Stage) : void
createHomeView(...) : javafx.scene.Node
createManifestView(tab: javafx.scene.control.Tab) : javafx.scene.Node
createCheckInView(tab: javafx.scene.control.Tab) : javafx.scene.Node
createLogoutTab(stage: javafx.stage.Stage) : javafx.scene.control.Tab
logout(stage: javafx.stage.Stage) : void
showAlert(title: String, content: String) : void
FlightService
FILE_PATH : String {static, final}
loadFlights(flightManager:Managers.FlightManager, planeManager:Managers.PlaneManager) : void
saveFlights(flightManager:Managers.FlightManager) : void
Thread
Scenario1
start(synchronizedMode:boolean) : FlightManagement.Plane
Runnable
creates
PassengerTask
passenger : TicketReservation.Passenger
flight : FlightManagement.Flight
manager : Managers.ReservationManager
synchronizedMode : boolean
flight: FlightManagement.Flight,
manager: Managers.ReservationManager,
PassengerTask(passenger: TicketReservation.Passenger, synchronizedMode: boolean)
run() : void
Scenario2
random : Random {final}
context : Managers.FlightSystemContext
planeCount : int {final}
flightCount : int {final}
passengerCount : int {final}
airports : String[] {final}
passengers : TicketReservation.Passenger[]
onReportReady : Consumer<String>
Scenario2(onReportReady: Consumer<String>)
run() : void
startScenario() : String
buildReport(totalTickets: int) : String
UnitTesting
FlightSearchEngineTest
context1 : Managers.FlightSystemContext
context2 : Managers.FlightSystemContext
context3 : Managers.FlightSystemContext
context4 : Managers.FlightSystemContext
context5 : Managers.FlightSystemContext
plane : FlightManagement.Plane
istAnk : FlightManagement.Route
istBer : FlightManagement.Route
FORMAT : DateTimeFormatter {static, final}
shouldReturnCorrectFlightsByDepartureAndArrival() : void
shouldReturnCorrectFlightsByDate() : void
shouldReturnCorrectFlightsByDateAndRoute() : void
shouldEliminateFlightsWithPastDepartureTime() : void
shouldEliminateFlightsWithPastDepartureTime2() : void
PriceCalculationTest
economyPriceCalculation() : void
businessPriceCalculation() : void
economyWithoutBaggage() : void
businessWithBaggage() : void
zeroDurationFlightPrice() : void
SeatManagerTest
context1 : Managers.FlightSystemContext
context2 : Managers.FlightSystemContext
TestNonExistentSeat() : void
TestNonExistentSeat2() : void
TestEmptySeatsDecreaseAfterReservation() : void
TestEmptySeatsDecreaseAfterReservation2() : void
TestEmptySeatsDecreaseAfterReservation3() : void
UserMenu
currentUser : GUI.LoginUser
tabPane : javafx.scene.control.TabPane
context : Managers.FlightSystemContext
CSS_FONT_BASE : String {static, final}
COLOR_PRIMARY : String {static, final}
COLOR_SUCCESS : String {static, final}
COLOR_TEXT : String {static, final}
UserMenu(user: GUI.LoginUser, context: Managers.FlightSystemContext)
start(stage: javafx.stage.Stage) : void
createHomeView(...) : javafx.scene.layout.VBox
createLogoutTab(stage: javafx.stage.Stage) : javafx.scene.control.Tab
logout(stage: javafx.stage.Stage) : void
createSearchView(tab: javafx.scene.control.Tab) : javafx.scene.Node
showBookingDialog(flight: FlightManagement.Flight) : void
showPassengerInfoDialog(flight: FlightManagement.Flight, seat: FlightManagement.Seat) : void
createReservationsView(tab: javafx.scene.control.Tab) : javafx.scene.Node
showBuyTicketDialog(targetReservation: TicketReservation.Reservation) : void
createTicketsView(tab: javafx.scene.control.Tab) : javafx.scene.Node
configureDatePicker(datePicker: javafx.scene.control.DatePicker) : void
styleButton(...) : void
styleTextField(...) : javafx.scene.control.TextField
styleTable(...) : void
findSeatObject(fl: FlightManagement.Flight, num: String) : FlightManagement.Seat
showAlert(title: String, content: String) : void
uses
PriceCalculationService
FILE_PATH : String {static, final}
loadCalculator() : Managers.CalculatePrice
saveParameters(base: double, tax: double, bagFee: double, bussMult: double, durCost: double) : boolean
createDefaultFile() : void
PlaneMapRenderer
SEAT_SVG_PATH : String {static, final}
COLOR_BUSINESS : String {static, final}
COLOR_ECONOMY : String {static, final}
COLOR_TAKEN : String {static, final}
COLOR_SELECTED : String {static, final}
COLOR_SCENARIO_TAKEN : String {static, final}
createInteractiveFuselage(plane: FlightManagement.Plane, onSeatSelected: Consumer<FlightManagement.Seat>) : javafx.scene.layout.VBox
renderSimulationMap(plane: FlightManagement.Plane, container: javafx.scene.layout.VBox) : void
createLegendItem(text: String, colorHex: String) : javafx.scene.layout.HBox
createInteractiveGrid(...) : javafx.scene.layout.GridPane
createSeatButton(...) : javafx.scene.control.Button
createSimGrid(seats: FlightManagement.Seat[][]) : javafx.scene.layout.GridPane
Managers
FlightSystemContext
ReservationService
FILE_PATH : String {static, final}
loadReservations(resManager: Managers.ReservationManager, flightManager: Managers.FlightManager) : void
saveReservations(resManager: Managers.ReservationManager) : void
findSeatOnPlane(plane: FlightManagement.Plane, seatNum: String) : FlightManagement.Seat
CalculatePrice
basePrice : double
tax : double
baggageFeePerKg : double
businessMultiplier : double
durationCostPerMinute : double
CalculatePrice(basePrice: double, tax: double, baggageFeePerKg: double, businessMultiplier: double, durationCostPerMinute: double)
priceCalculation(flight: FlightManagement.Flight, seat: FlightManagement.Seat) : double
priceCalculation(reservation: TicketReservation.Reservation, baggageWeight: double) : double
FlightSystemContext
planeManager : PlaneManager
seatManager : SeatManager
ticketManager : TicketManager
flightManager : FlightManager
reservationManager : ReservationManager
calculatePrice : CalculatePrice
FlightSystemContext()
setContexts() : void
refreshFleetData() : void
refreshFlightData() : void
refreshBookingData() : void
refreshAllData() : void
FlightManager
flights : ArrayList<FlightManagement.Flight>
counter : int
context : FlightSystemContext
createFlight(planeId:String, route:FlightManagement.Route, date:String, hour:int, duration:int) : boolean
addLoadedFlight(flight:FlightManagement.Flight) : void
updateFlight(flightNum:int) : boolean
deleteFlight(flightNum:int) : boolean
deleteFlightByPlane(plane:FlightManagement.Plane) : boolean
searchFlights(dep:String, arr:String, date:String) : List<FlightManagement.Flight>
searchFlightsByRoute(dep:String, arr:String) : List<FlightManagement.Flight>
searchFlightsByDate(date:String) : List<FlightManagement.Flight>
hasDepartureTimePassed(flight:FlightManagement.Flight) : boolean
getAllFlights() : List<FlightManagement.Flight>
setContext(context: FlightSystemContext) : void
SeatManager
context : FlightSystemContext
SeatManager()
createSeatings(seatArrangement: int[][]) : FlightManagement.Seating
calculateAvailableSeats(plane: FlightManagement.Plane) : int[]
PlaneManager
planes : HashMap<String, FlightManagement.Plane>
planeCounter : int
scanner : Scanner
context : FlightSystemContext
PlaneManager()
createPlane(planeModel: String, seatings: FlightManagement.Seating) : boolean
addLoadedPlane(plane: FlightManagement.Plane) : void
updatePlane(planeId: String, newModel: String, newSeating: FlightManagement.Seating) : boolean
deletePlane(planeId: String, ticketManager: TicketManager) : void
removeAllPlanes() : void
getPlane(planeId: String) : FlightManagement.Plane
getAllPlaneIds() : List<String>
getPlanesList() : List<FlightManagement.Plane
setContext(context: FlightSystemContext) : void
changePlaneModel(plane: FlightManagement.Plane, newModel: String) : boolean
changeSeatMatrix(plane: FlightManagement.Plane, seating: FlightManagement.Seating) : boolean
TicketService
FILE_PATH : String {static, final}
loadTickets(ticketManager: Managers.TicketManager, resManager: Managers.ReservationManager) : void
saveTickets(ticketManager: Managers.TicketManager) : void
TicketManager
tickets : HashMap<String, TicketReservation.Ticket>
ticketCounter : int
context : FlightSystemContext
createTicket(reservation:TicketReservation.Reservation) : boolean
createTicket(reservation:TicketReservation.Reservation, baggageWeight:double) : boolean
addLoadedTicket(ticket:TicketReservation.Ticket) : void
deleteTicket(ticketID:String) : boolean
deleteTicketByReservation(reservation:TicketReservation.Reservation) : boolean
getAllTickets() : Iterable<TicketReservation.Ticket>
setContext(context: FlightSystemContext) : void
ReservationManager
reservations : HashMap<String, TicketReservation.Reservation>
context : FlightSystemContext
ReservationManager()
createReservation(flight: FlightManagement.Flight, passenger: TicketReservation.Passenger, seat: FlightManagement.Seat) : boolean
createRandomReservationAsync(flight: FlightManagement.Flight, passenger: TicketReservation.Passenger) : boolean
createRandomReservationSync(flight: FlightManagement.Flight, passenger: TicketReservation.Passenger) : boolean
addLoadedReservation(res: TicketReservation.Reservation) : void
cancelReservation(reservationCode: String) : boolean
deleteReservationByFlight(flight: FlightManagement.Flight) : boolean
removeAllReservations() : void
getAllReservations() : Iterable<TicketReservation.Reservation>
setContext(context: FlightSystemContext) : void
javafx.application.Application
TicketReservation
Ticket
ticketId : String
reservation : Reservation
price : double
baggageAllowance : Baggage
Ticket(ticketId:String, reservation:Reservation, calculator:Managers.CalculatePrice)
Ticket(ticketId:String, reservation:Reservation, baggageWeight:double, calculator:Managers.CalculatePrice)
getBaggageWeight() : double
toString() : String
Ticket.Baggage
weight : double
Baggage(weight:double)
toString() : String
Reservation
reservationCode : String
flight : FlightManagement.Flight
passenger : Passenger
seat : FlightManagement.Seat
dateOfReservation : String
Reservation(reservationCode:String, flight:FlightManagement.Flight, passenger:Passenger, seat:FlightManagement.Seat, date:String)
toString() : String
Passenger
passengerID : String
name : String
surname : String
contactInfo : ContactInfo
Passenger(id:String, name:String, surname:String, mail:String, phone:String, address:String)
toString() : String
Passenger.ContactInfo
mail : String
phone : String
address : String
ContactInfo(mail:String, phone:String, address:String)
toString() : String
FlightManagement
Flight
flightNum : int
plane : Plane
route : Route
date : String
hour : int
duration : int
Flight(flightNum:int, plane:Plane, route:Route, date:String, hour:int, duration:int)
getRandomAvailableEconomySeat() : Seat
toString() : String
Route
departurePlace : String
arrivalPlace : String
Route(departure:String, arrival:String)
toString() : String
Plane
planeId : String
planeModel : String
capacity : int
econSeats : Seat[][]
bussSeats : Seat[][]
Plane(planeId:String, planeModel:String, seatings:Seating)
hasAnyReservedSeat() : boolean
availableSeats() : int[]
getTotalSeatCapacity() : int
toString() : String
Seating
bussSeats : Seat[][]
econSeats : Seat[][]
Seating(bussSeats:Seat[][], econSeats:Seat[][])
Seat
seatNum : String
flightClass : FlightClass
price : double
reserveStatus : boolean
Seat(seatNum:String, flightClass:FlightClass)
toString() : String
FlightClass
pricing : double
BUSINESS(0.0)
ECONOMY(0.0)