YAZILIM LABORATUVARI 1
2. PROJE

Ersin ALPASLAN - Yunus Emre GÜL
Bilgisayar Mühendisliği Bölümü
Kocaeli Üniversitesi
ersinalpaslan123@gmail.com - yunemregul@gmail.com

Özet

Bu doküman Yazılım Laboratuvarı 1 dersi 2. Projesi için çözümümüzü açıklamaya yönelik oluşturulmuştur. Dökümanda projenin tanımı, çözüme yönelik yapılan araştırmalar, kullanılan yöntemler, proje sürecinde karşılaşılan problemler, proje hazırlanırken kullanılan geliştirme ortamı gibi programın oluşumunu açıklayan başlıklara yer verilmiştir. Doküman sonunda proje sonucu ve projemizi hazırlarken kullandığımız kaynaklar bulunmaktadır.

1. Proje Tanımı

1.1. Proje Tanımı

Projede bizden AVM asansör sistemini temsil eden bir program yazmamız beklenmektedir. Asansörlere gelen isteklerdeki yoğunluğun bir çok threada bölünerek hafifletilmesi beklenmektedir.

Projenin amacı multithreading konseptini deneyimlememizdir.

Projede herhangi bir programlama dili kullanılabilir. Uygulama masaüstü uygulaması şeklinde olmalıdır.

1.2. İsterler

Projede 5 adet temel bileşen bulunmaktadır. Bileşenler aşağıdaki gibidir:

• **AVM Giriş (Login) Thread:** 500 ms zaman aralıklarıyla [1-10] arasında rastgele sayıda müşterinin AVM'ye giriş yapmasını sağlamaktadır (Zemin Kat). Giren müşterileri rastgele bir kata (1-4) gitmek için asansör kuyruğuna alır.

• **AVM Çıkış (Exit) Thread:** 1000 ms zaman aralıklarıyla [1-5] arasında rastgele sayıda müşterinin AVM'den çıkış yapmasını sağlamaktadır (Zemin Kat). Çıkmak isteyen müşterileri rastgele bir kattan (1-4), zemin kata gitmek için asansör kuyruğuna alır.

• **Asansör Thread:** Katlardaki kuyrukları kontrol eder. Maksimum kapasiteyi aşmayacak şekilde kuyruktaki müşterilerin talep ettikleri katlarda taşınabilmesini sağlar. Bu thread asansör sayısı kadar (5 adet) olmalıdır.

NOT: Zemin kattan diğer katlara (AVM'ye) giriş yapmak isteyenler, ya da diğer katlardan (AVM'den) çıkış yapmak isteyenler kuyruk oluştururlar.

• **AVM Çıkış (Exit) Thread:** 1000 ms zaman aralıklarıyla [1-5] arasında rastgele sayıda müşterinin AVM'den çıkış yapmasını sağlamaktadır (Zemin Kat). Çıkmak isteyen müşterileri rastgele bir kattan (1-4), zemin kata gitmek için asansör kuyruğuna alır.

AVM de toplam beş kat ve beş asansör bulunmalıdır. Asansörlerden bir tanesi sürekli çalışmalı, diğerleri gerektikçe açılmalı ve gerekmedikçe kapanmalıdır. Asansörlerin maksimum kapasitesi 10'dur. Asansörlerin kat arası geçiş hızı 200ms'dir.

2. Araştırmalar ve Yöntem

Projemize hangi dili kullanacağımızı seçmekle başladık. Önceden de tecrübemiz olduğu için Java dilini kullanmayı seçtik.

Java ile yapmaya karar verince Java da Thread yapısı üzerine araştırmalar yaptık ve öğrendik. Thread bir programın altında çalışan birbiriyle ilişkili ya da ilişkisiz işlemler yapabilen, işlem yaparken birbirini duraksatmaya gerek olmayan paralel çalışan yapılara denir.

Projede istenen bileşenlerin her biri kendi threadında çalışmalıdır. Böylece yaptıkları işlemler birbirlerinin devam etmesini engellemeyecektir.

Projemizde AVM yapısını nasıl temsil edeceğimizi düşünüp karar verdik. Projede istenen her temel birim için ayrı bir sınıf oluşturmayı uygun bulduk. Örneğin AVM sınıfı, Kat sınıfı, Müşteri sınıfı, Asansör sınıfı gibi. Böylece sınıflarımızı gerçek hayatı temsil edecek şekilde oluşturmuş olduk. Nesneye yönelik programlamayı birebir canlandırmış olduk.

AVM sınıfımız üzerine statik özellikler olarak *asansorler* ve *katlar* adında özellikler tanımladık. Böylece herhangi bir sınıf AVM üzerindeki asansörler ve katlara direk ulaşabildi. Bu da işlerimizi kolaylaştırdı.

Her *kat* nesnesi içine *müşteri* nesnesi alan iki bağlı listeden oluşuyor. Bu listelerden biri o kattaki müşterileri, diğeri de o kattan çıkıp zemin kata gitmek isteyen müşterileri tutuyor. Kattaki müşterileri ya da çıkmak isteyen müşterileri öğrenmek isteyenler tanımladığımız *getMusteriler, getCikacaklar* isimli senkronize fonksiyonları kullanarak öğrenebiliyor. Bu fonksiyonları senkronize yapmamız sayesinde Thread larımız arasında bilgi kaybı olmasını da önlemiş olduk.

AVM sınıfı üzerinde önceden de belirttiğimiz gibi statik *asansorler* ve *katlar* özelliklerine sahip. Katlar özelliği beş büyüklüğünde *Kat* nesnesi dizisinden oluşuyor. AVM nin katlarını temsil ediyor. Asansorler özelliği de beş büyüklüğünde *Asansor* nesnesi dizisinden oluşuyor. AVM nin asansörlerini temsil ediyor.

Müşteri sınıfımız AVM deki bir müşteriyi temsil ediyor, üzerinde müşterinin hangi kata gitmek istediğini belirten *hedefKat* özelliği tanımlı.

Giriş sınıfımız proje tanımında belirtildiği şekilde belirli aralıklarla belirli sayıda *Müşteri* nesnesi türeterek AVM nin zemin katına girişlerini sağlıyor. Bu sınıf bir Threaddan türüyor ve bu Thread da uygulama çalıştığı sürece durmadan çalışıyor.

Çıkış sınıfımız yine proje tanımında belirtildiği şekilde belirli aralıklarla belirli katlardan Müşterileri o katın çıkış kuyruğuna ekliyor. Böylece bir asansör geldiğinde o kattan çıkmak isteyenleri ayırabiliyor. Bu sınıf bir Threaddan türüyor ve bu Thread da uygulama çalıştığı sürece durmadan çalışıyor.

Asansör sınıfımız üzerinde asansördeki müşterileri temsilen bir *ArrayList* bulunuyor. Ayrıca asansörün olduğu katı tutan bir özellik, asansörün çalışıp çalışmadığını tutan bir özellik, asansörün gideceği hedef katı tutan bir özellik ve asansörün şu anda durma durumunda olup olmadığını tutan bir özellik bulunuyor. Asansörün şu anda hangi kata gideceğini belirleyen bir *gidilecekKatBelirle* fonksiyonumuz bulunuyor. Bu fonksiyon eğer asansör şu an zemin katta ise en üst kata, en üst katta ise de zemin kata yönlendiriyor. Böylece asansörümüz sürekli yukarı aşağı giderek tüm müşterilerin alınmasını sağlıyor. Asansör sınıfımız bir Thread sınıftan türüyor.

Kontrol sınıfımız proje tanımında belirtildiği üzere gerekli asansör varsa açmaya, gereksiz asansör varsa da kapatmaya yarıyor. Eğer tüm bekleyen kişi sayısı 20'den fazla ise yeni bir asansör açıyor, belirli bir süre uyuduktan sonra bekleyen kişi sayısı hâla 20 den fazla ise yeni bir asansör daha açıyor ve bu şekilde ilerliyor. Toplam bekleyen kişi sayısı 10 un altına düştüğü durumda da gereksiz bir asansörü kapatıyor ve yine belirli bir süre bu sınıf uyutuluyor. Bu sınıf bir Threaddan türüyor ve bu Thread da uygulama çalıştığı sürece durmadan çalışıyor.

Uygulamada birden çok thread olduğunda ve bunlar ortak değişkenler üzerinde değişiklik yaptığında bu değişiklikler threadlara anında gitmemesi gibi bir sorun oluşuyor. Bu sorunu çözmek için araştırmalar yaptık ve *synchronized* deyimini öğrendik. Bu deyim threadlar içinde belli kısımların senkronize olarak çalışmasını zorunlu kılıyor. Senkronize olarak çalışmak demek bir thread belirli bir değişken üzerinde değişiklik yaparken diğer threadların beklemesi anlamına geliyor. Böylece olası çakışmalar ve bilgi kayıpları önlenmiş oluyor.

2.1. Karşılaşılan Problemler

Proje genelinde çok fazla sorunla karşılaşmadık ancak karşılaştığımız ufak problemler bu bölümde bulunmaktadır.

Karşılaştığımız bir problem Java da thread yapısını kullanırken thread ların kapatıldığında aynı thread nesnesinin tekrar açılamaması durumuydu. Bunun çaresinin yeniden bir Thread nesnesi oluşturmak olduğunu öğrendik. Bu sorunla karşılaştığımız kısım ekstra asansör gerektiğinde açma, gereksiz asansör olduğunda da kapatma kısmıydı. Bir asansörü açarken sorun yoktu ama asansörü kapattıktan sonra tekrar açarken sorun vardı. Java buna izin vermiyordu. Bunu da yeni bir asansörü açarken her seferinde yeniden asansör nesnesi oluşturarak çözdük.

Karşılaştığımız bir diğer problem de ikinci bölümde de belirttiğimiz gibi thread lar arası senkronizasyon durumuydu. Bunu da thread ların ortak düzenlediği değişkenlerin düzenlendiği kısımları *synchronized* bloğuna alarak çözdük.

2.2. Kazanımlar

Bu projeyle birlikte grup çalışmasını deneyimlemiş olduk.

Yine bu projeyle birlikte birden çok Thread ın birbiri ile iletişiminin nasıl olacağını da deneyimlemiş olduk.

3. Geliştirme Ortamı

Projemizi Windows işletim sisteminde, Intellij IDE üzerinde geliştirdik. Projemizin gelişimini ve versiyonlarını takip edebilmek için de Git versiyon kontrol sistemi kullandık.

4. Kod Bilgisi

4.1. Akış Diyagramı

Kısım ektedir. [1](Akış Diyagramı)

4.2 İstatistik

Program kodu boşluksuz ve yorumsuz toplam 452 Java satırından oluşmaktadır. Kod düzenini sağlamak için toplamda 79 boş satır kullanılmıştır.

4.4. Projenin Çalıştırılması

Proje JAR dosyası java ile açılarak çalıştırılabilir. Çalıştırılmasına dair detaylar ve görüntüler ekte bulunmaktadır. [3](Proje Görüntüleri)

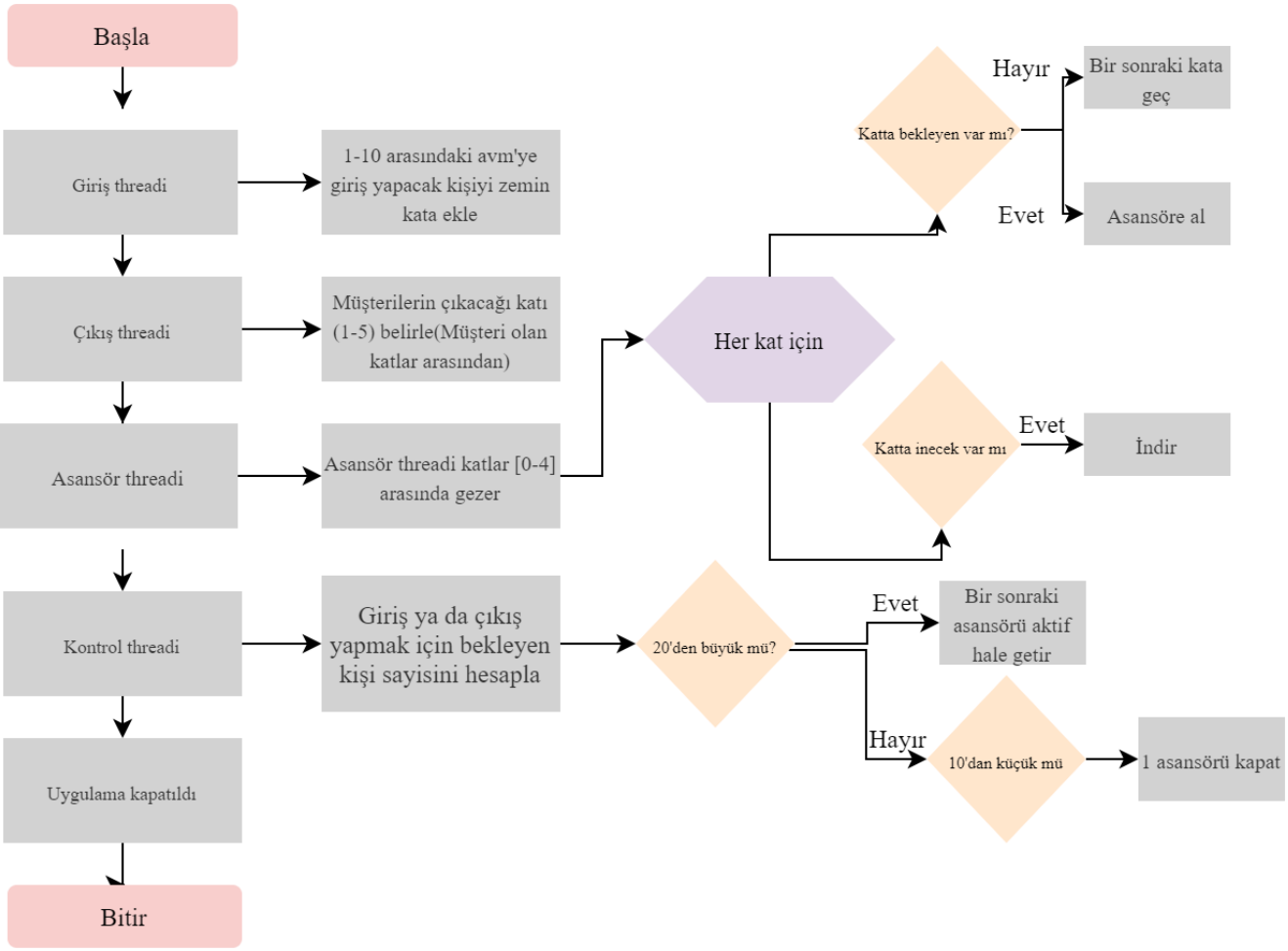
4.5. Sonuç

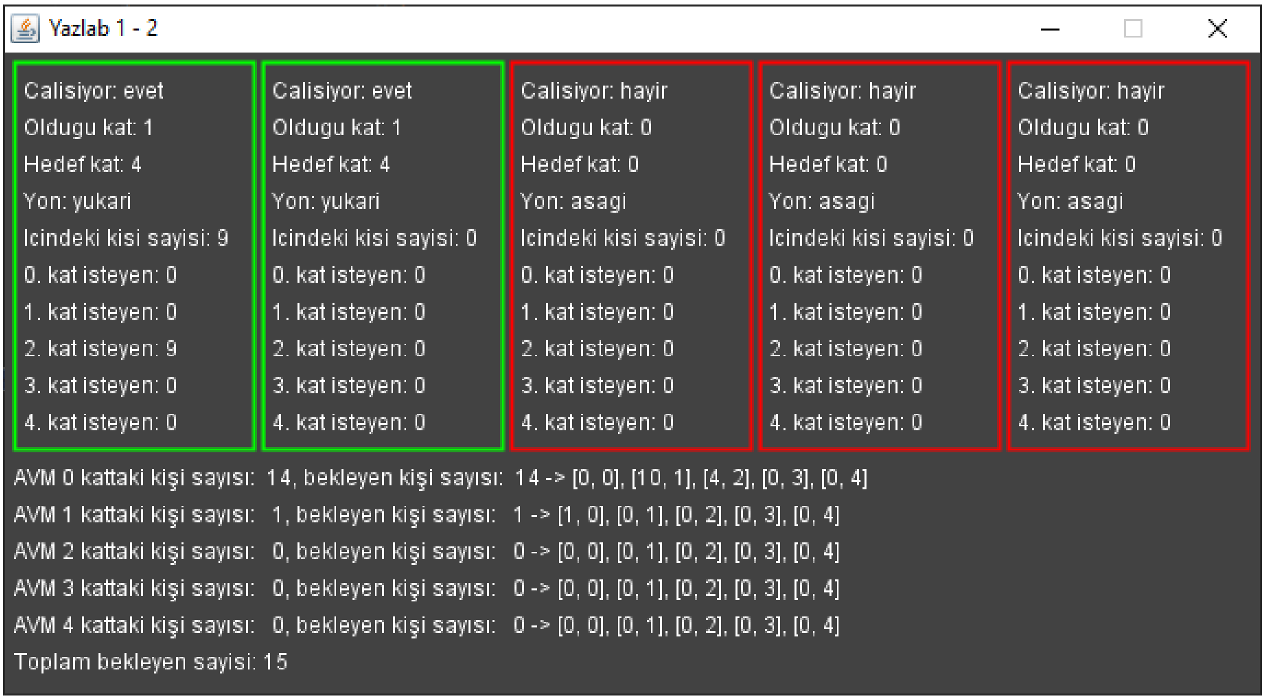
Projenin gerektirdiği tüm isterleri başarıyla tamamladık. Java ile multithreading işlemlerini öğrenmiş ve deneyimlemiş olduk.

Kaynakça

- https://www.w3schools.com/java/java_threads.asp (Access Date: 05.12.2020)
- https://stackoverflow.com/

EK 1 (Akış Diyagramı)




Yazlab 1 - 2
Calisiyor: evet
Oldugu kat: 1
Hedef kat: 4
Yon: yukari
Icindeki kisi sayisi: 9
0. kat isteyen: 0
1. kat isteyen: 0
2. kat isteyen: 9
3. kat isteyen: 0
4. kat isteyen: 0
Calisiyor: evet
Oldugu kat: 1
Hedef kat: 4
Yon: yukari
Icindeki kisi sayisi: 0
0. kat isteyen: 0
1. kat isteyen: 0
2. kat isteyen: 0
3. kat isteyen: 0
4. kat isteyen: 0
Calisiyor: hayir
Oldugu kat: 0
Hedef kat: 0
Yon: asagi
Icindeki kisi sayisi: 0
0. kat isteyen: 0
1. kat isteyen: 0
2. kat isteyen: 0
3. kat isteyen: 0
4. kat isteyen: 0
Calisiyor: hayir
Oldugu kat: 0
Hedef kat: 0
Yon: asagi
Icindeki kisi sayisi: 0
0. kat isteyen: 0
1. kat isteyen: 0
2. kat isteyen: 0
3. kat isteyen: 0
4. kat isteyen: 0
Calisiyor: hayir
Oldugu kat: 0
Hedef kat: 0
Yon: asagi
Icindeki kisi sayisi: 0
0. kat isteyen: 0
1. kat isteyen: 0
2. kat isteyen: 0
3. kat isteyen: 0
4. kat isteyen: 0
AVM 0 kattaki kişi sayısı: 14, bekleyen kişi sayısı: 14 -> [0, 0], [10, 1], [4, 2], [0, 3], [0, 4]
AVM 1 kattaki kişi sayısı: 1, bekleyen kişi sayısı: 1 -> [1, 0], [0, 1], [0, 2], [0, 3], [0, 4]
AVM 2 kattaki kişi sayısı: 0, bekleyen kişi sayısı: 0 -> [0, 0], [0, 1], [0, 2], [0, 3], [0, 4]
AVM 3 kattaki kişi sayısı: 0, bekleyen kişi sayısı: 0 -> [0, 0], [0, 1], [0, 2], [0, 3], [0, 4]
AVM 4 kattaki kişi sayısı: 0, bekleyen kişi sayısı: 0 -> [0, 0], [0, 1], [0, 2], [0, 3], [0, 4]
Toplam bekleyen sayisi: 15