ASANSÖRLERDEKİ TALEP YOĞUNLUĞUNUN MULTİTHREAD İLE KONTROLÜ

KOCAELİ ÜNİVERSİTESİ BİLGİSAYAR MÜHENDİSLİĞİ

YAZILIM LABORATUVARI PROJESİ

YASİN ÖMER KARA-FEYZA DEMİREL

180201077-190201110

ÖZET

Projemizde bir AVM'deki asansörlere gelen isteklerdeki yoğunluğu,multithread kullanarak diğer asansörlerle birlikte azaltmaya çalıştık. Thread kavramını , algoritma yönetimi ile çözümler üreterek kat sayısı 5 olan AVM'mizdeki toplamda 5 adet asansörümüzü , ilk asansörümüz sürekli aktif olacak şekilde kullanım durumlarına göre thread yönetimi ile aktif-pasif durumlarına getirdik.AVM giriş-çıkış işlemlerimizin kontrolünü delaylar ile sağlayıp bizden istenilen şekilde asansörler arası geçişleri ve yoğunluk azaltımını gerçekleştirdik.

1.GİRİŞ

Öncelikle kavramamız gereken temel konu olan Thread kavramı nedir ve nasıl kullanılır sorusunu cevaplandırdık.

Thread Nedir?

Thread bir diğer adıyla İş Parçacığı, aynı process ortamında birden fazla iş yürütme imkanı sağlar. Bir Process'in çalışmaya başlaması ile birlikte bir thread (main thread) oluşturulur ve process içerisinde birden fazla (multi-thread) oluşturulabilir. Yaratılan threadler aynı networkde ve farklı networkte işlem yapabilirler. Threadler genel olarak yazılım testleri yapılırken simülasyon olarak kullanılabilir ancak bilişim sistemlerinde cpu üzerinden işlem yapan thread fiziki olarak işlem yaptığı için yazılım test senaryosundaki thread gruplardan farklı davranış gösterebilir. Threadler kullanıldığı alanlara göre farklılık gösterebilir.

Özet olarak: Thread (iş parçacığı) kullanımı, birden fazla işlemin tek bir akışı paylaşarak neredeyse eşzamanlı bir şekilde gerçekleşmesini sağlar.

2.YÖNTEM

Projemizin Verilenleri ve İsterleri:

• AVM'deki kat sayısı 5'tir.

• Toplamda 5 adet asansör bulunmaktadır.

• Asansörlerin biri sürekli çalışmaktadır. Geriye kalanlar, yoğunluk durumuna göre aktif veya pasif durumdadır.

• Asansörlerin maksimum kapasitesi 10'dur.

• Asansörlerdeki kat arası geçiş 200 ms'dir.

Proje Bileşenlerinin Özellikleri ve Kendi Tanımlamalarımız :

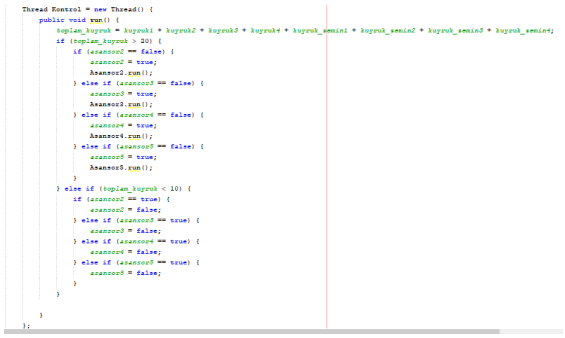
1) AVM Giriş (Login) Thread: 500 ms zaman aralıklarıyla [1-10] arasında rastgele sayıda müşterinin AVM'ye giriş yapmasını sağlamaktadır (Zemin Kat). Giren müşterileri rastgele bir kata (1-4) gitmek için asansör kuyruğuna alır.

```
Random random = new Random();
Thread Giris = new Thread() {
    public void run() {

        int giren_kisi_sayisi = 1 + random.nextInt(9);
        int kat = 1 + random.nextInt(4);
        if (kat == 1) {
            kuyruk_zemin1 += giren_kisi_sayisi;
        }
        if (kat == 2) {
            kuyruk_zemin2 += giren_kisi_sayisi;
        }
        if (kat == 3) {
            kuyruk_zemin3 += giren_kisi_sayisi;
        }
        if (kat == 4) {
            kuyruk_zemin4 += giren_kisi_sayisi;
        }

    }
};
```

2) 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.

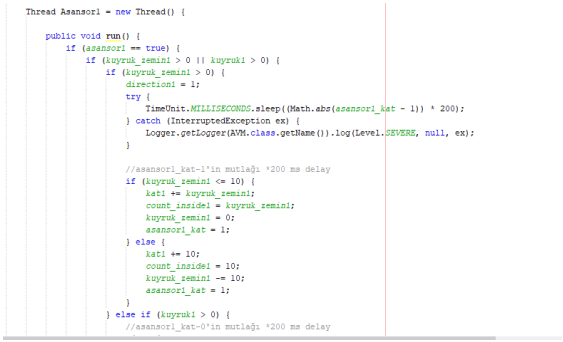
```
Thread Cikis = new Thread() {
    public void run() {
        int cikan_kisi_sayisi = 1 + random.nextInt(5);
        int cikilcak_kat = 1 + random.nextInt(4);

        if (cikilcak_kat == 1) {
            kuyruk1 += cikan_kisi_sayisi;
            // kat1 -= cikan_kisi_sayisi;
        }
        if (cikilcak_kat == 2) {
            kuyruk2 += cikan_kisi_sayisi;
            // kat2 -= cikan_kisi_sayisi;
        }
        if (cikilcak_kat == 3) {
            kuyruk3 += cikan_kisi_sayisi;
            // kat3 -= cikan_kisi_sayisi;
        }
        if (cikilcak_kat == 4) {
            kuyruk4 += cikan_kisi_sayisi;
            // kat4 -= cikan_kisi_sayisi;
        }

    }
};
```

3) Kontrol Thread: Katlardaki kuyrukları kontrol eder. Kuyrukta bekleyen kişilerin toplam sayısı asansörün kapasitesinin 2 katını aştığı durumda (20) yeni asansörü aktif hale getirir. Kuyrukta bekleyen kişilerin toplam sayısı asansör kapasitenin altına indiğinde asansörlerden biri pasif hale gelir. Bu işlem tek asansörün çalıştığı durumda geçerli değildir

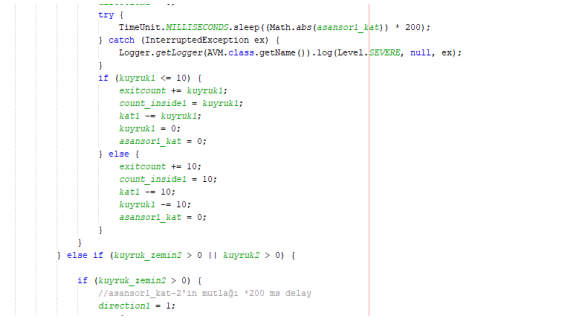


4) Asansör Thread : Katlardaki kuyrukları kontrol eder. Maksimum kapasiteyi aşmayacak şekilde kuyruktaki müşterilerin talep ettikleri katlarda taşınabilmesini sağlar. Bu thread asansör sayısı kadar (5 adet) olmalıdır.

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



```
Thread Asansor1 = new Thread() {

    public void run() {
        if (asansor1 == true) {
            if (kuyruk_zemin1 > 0 || kuyruk1 > 0) {
                if (kuyruk_zemin1 > 0) {
                    direction1 = 1;
                    try {
                        TimeUnit.MILLISECONDS.sleep((Math.abs(asansor1_kat - 1)) * 200);
                    } catch (InterruptedException ex) {
                        Logger.getLogger(AVM.class.getName()).log(Level.SEVERE, null, ex);
                    }

                    //asansor1_kat-1'in mutlağı *200 ms delay
                    if (kuyruk_zemin1 <= 10) {
                        kat1 += kuyruk_zemin1;
                        count_inside1 = kuyruk_zemin1;
                        kuyruk_zemin1 = 0;
                        asansor1_kat = 1;
                    } else {
                        kat1 += 10;
                        count_inside1 = 10;
                        kuyruk_zemin1 -= 10;
                        asansor1_kat = 1;
                    }
                } else if (kuyruk1 > 0) {
                    //asansor1_kat-0'in mutlağı *200 ms delay
```



```
                    try {
                        TimeUnit.MILLISECONDS.sleep((Math.abs(asansor1_kat)) * 200);
                    } catch (InterruptedException ex) {
                        Logger.getLogger(AVM.class.getName()).log(Level.SEVERE, null, ex);
                    }
                    if (kuyruk1 <= 10) {
                        exitcount += kuyruk1;
                        count_inside1 = kuyruk1;
                        kat1 -= kuyruk1;
                        kuyruk1 = 0;
                        asansor1_kat = 0;
                    } else {
                        exitcount += 10;
                        count_inside1 = 10;
                        kat1 -= 10;
                        kuyruk1 -= 10;
                        asansor1_kat = 0;
                    }
                }
            } else if (kuyruk_zemin2 > 0 || kuyruk2 > 0) {

                if (kuyruk_zemin2 > 0) {
                    //asansor1_kat-2'in mutlağı *200 ms delay
                    direction1 = 1;
```

Asansör Threadinin İçeriğinin Uzunluğundan Ötürü Bir Parçası Görüntülenmiştir.

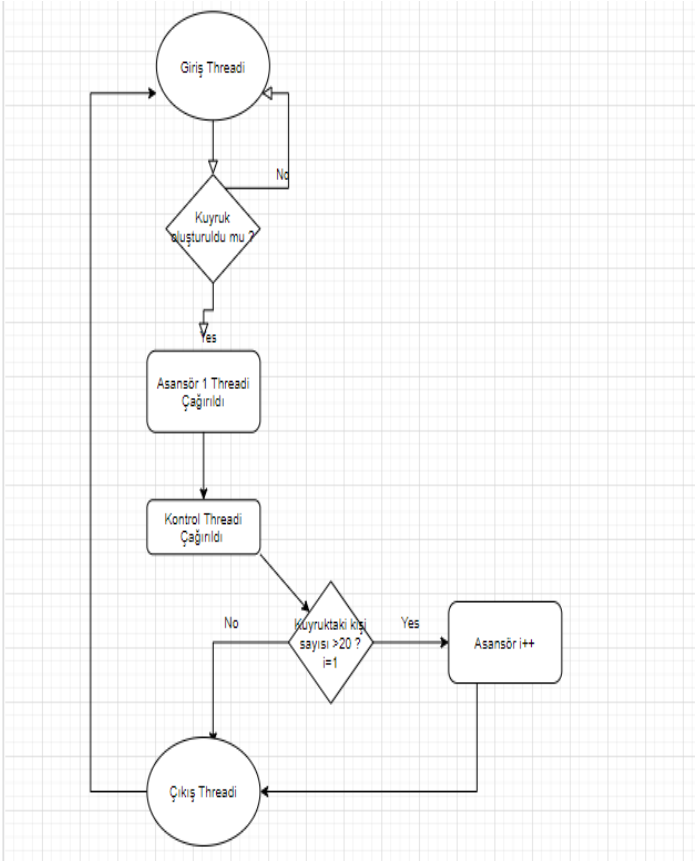
Genel Olarak:

Katlardaki kişi sayısını kontrol ederek kapasitesini aşmıyacak şekilde , kişileri katlara alır ve talep ettikleri katlara ulaştırır.Asansör katlara kişileri bıraktığında veya alındığında, katlardaki kişi sayısı düzenlenir.

3.SONUÇ

Biz bu projemizde thread kullanımını,multithread yönetimini, JAVA üzerinden gerçekleştirdiğimiz için JAVA'nın bize thread yönetiminde sağladığı kolaylıkları öğrenmiş olduk.
Threadler üzerindeki işlemleri, bunların kontrol yapılarını ve çıkış işlemlerinin işleyişlerini kavradık ve bizden istenen şekilde projemizi gerçekleştirdik.

4.AKIŞ ŞEMASI



5.DENEYSEL SONUÇLAR

```
run:
0.floor : queue : 0
1.floor : all : 0 queue : 0
2.floor : all : 3 queue : 1
3.floor : all : 0 queue : 0
4.floor : all : 0 queue : 0
Asansör1
active :true
        mode : working
        floor : 2
        direction : up
        capacity : 10
        count_inside : 3

Asansör2
active :false
        mode : idle
        floor : 0
        direction : down
        capacity : 10
        count_inside : 0

Asansör3
active :false
        mode : idle
        floor : 0
        direction : down
        capacity : 10
        count_inside : 0

Asansör4
active :false
        mode : idle
        floor : 0
        direction : down
        capacity : 10
        count_inside : 0

Asansör5
active :false
        mode : idle
        floor : 0
        direction : down
        capacity : 10
        count_inside : 0
0.floor : [[0,1][0,2][0,3][0,4]]
1.floor : []
2.floor : [[1,0]]
3.floor : []
4.floor : []
Exit count: 0
++++++++++++++++++++++++
```

```
0.floor : queue : 9
1.floor : all : 0 queue : 0
2.floor : all : 6 queue : 0
3.floor : all : 0 queue : 4
4.floor : all : 0 queue : 4
Asansör1
active :true
        mode : working
        floor : 0
        direction : down
        capacity : 10
        count_inside : 3

Asansör2
active :true
        mode : working
        floor : 2
        direction : up
        capacity : 10
        count_inside : 6

Asansör3
active :false
        mode : idle
        floor : 0
        direction : down
        capacity : 10
        count_inside : 0

Asansör4
active :false
        mode : idle
        floor : 0
        direction : down
        capacity : 10
        count_inside : 0

Asansör5
active :false
        mode : idle
        floor : 0
        direction : down
        capacity : 10
        count_inside : 0
0.floor : [[0,1][0,2][9,3][0,4]]
1.floor : []
2.floor : []
3.floor : [[4,0]]
4.floor : [[4,0]]
Exit count: 7
```

- Bu projede Thread ve MultiThread yapısını kullanarak , birden fazla Threadin etkilediği durumun kontrolünü sağlamayı öğrendik.
- Javada TimeUnit kütüphanesini kullanılarak nasıl gecikme (zaman aralığı) eklenir, öğrendik.
- MultiThreadlerin hangi sırayla, hangi düzene göre nerede çağrılması gerektiğini öğrendik.
- Threadin belli zaman aralıkları ile nasıl ve hangi kurallara göre çağırılacağını öğrendik.

5.YAZILIM MİMARİSİ

- Değişken tanımlamaları yapıldı.

-Giriş Threadi yazıldı.Giriş Threadinin içerisinde 1-10 arasında random sayı üretildi ve random bir katın kuyruğuna bu sayı eklendi.

-Çıkış Threadi yazıldı.Çıkış Threadinin içerisinde 1-5 arasında random sayı üretildi ve zemin kat kuyruğuna bu sayı eklendi .

-Asansör1 Threadi yazıldı.Katlardaki kuyruklar kontrol edilerek AVMdeki kişilerin istedikleri katlara ulaşması if-else if yapısıyla sağlandı ve her kat arası geçiş için 200 milisaniye zaman aralığı verildi.

-Diğer Asansör Threadleri de Asansör1 gibi hazırlanmıştır;{

Asansör2 Threadi yazıldı.

Asansör3 Threadi yazıldı.

Asansör4 Threadi yazıldı.

Asansör5 Threadi yazıldı.}

-Kontrol Threadi yazıldı.Kontrol Threadinin içerisinde istenilen şartların sağlanma durumuna göre Asansör 2,Asansör 3,Asansör 4

ve Asansör 5 Threadleri if - else if yapılarına göre çağırıldı.

-Try-catch bloğu içerisinde 500 milisaniyede bir Giriş Threadi ve 1000 milisaniyede bir Çıkış Threadi çağrıldı.Ayrıca try bloğunun içerisinde kontrol Asansör 1 ve Kontrol Threadi çağırıldı.

-Projede Arayüz ekranında verilmesi belirtilen veriler , try bloğunun içerisinde istenilen düzene göre ekrana yazdırıldı.

6.KULLANDIĞIMIZ KÜTÜPHANELER

```
1. import java.io.IOException;
2. import java.util.Random;
3. import
   java.util.concurrent.TimeUn
   it;
4. import
   java.util.logging.Level;
5. import
   java.util.logging.Logger;
```

7.KAYNAKÇA

https://emrahmete.wordpress.com/2011/10/06/javada-thread-yapisi-ve-kullanimi-hakkinda-ipuclari/

https://ufukuzun.wordpress.com/2015/02/14/javada-multithreading-bolum-1-merhaba-thread/

http://bilgisayarkavramlari.com/2010/03/22/thread-iplik/

https://docs.oracle.com/javase/7/docs/api/java/util/concurrent/TimeUnit.html