ASANSÖRLERDEKİ TALEP YOĞUNLUĞUNUN MULTITHREAD İLE KONTROLÜ

Cumali TOPRAK
Mühendislik Fakültesi, Bilgisayar Mühendisliği
Kocaeli Üniversitesi
Kocaeli, Türkiye
cumalitoprakk@gmail.com

Berkay Efe ÖZCAN
Mühendislik Fakültesi, Bilgisayar Mühendisliği
Kocaeli Üniversitesi
Kocaeli, Türkiye
berkayefeozcan@gmail.com

***Özetçe*— Bu projenin amacı bir AVM'deki asansörlere gelen isteklerdeki yoğunluğu, multithread kullanarak diğer asansörlerle birlikte azaltmaktır. Bu projedeki senaryoda alışveriş merkezindeki kat sayısı beştir. Toplamda beş adet asansör bulunmaktadır. Bu asansörlerin biri sürekli çalışmaktadır, geriye kalanlar yoğunluk durumuna göre aktif veya pasif durumuna geçmektedir. Asansörlerin maksimum kapasitesi 10'dur. Asansörlerdeki katlar arası geçiş 200'ms dir. Müşterilerin alışveriş merkezine girmesini ve giriş katta asansör kuyruğunda beklemesini kontrol eden bir login thread, diğer katlardan alışveriş merkezinden çıkmak için rasgele sayıda müşteriyi o katın asansör kuyruğuna ekleyen exit thread, ve yoğunluk durumuna göre yeni bir asansör threadini aktif veya pasif hale getiren kontrol threadi yer almaktadır. Bu projede java dili kullanılmış olup intellij idea editöründen yararlanılmıştır. Proje sonunda öngörülen kazanımlar elde edilmiş olup, multi threading kavramı öğrenilmiştir.**

Anahtar Kelimeler—Çoklu İş Parçacığı, Java, Asansör Uygulaması, Senkronizasyon.

***Abstract* — The aim of this project is to reduce the density of requests to elevators in a shopping mall together with other elevators by using multithread. In the scenario in this project, the number of floors in the shopping center is five. There are five elevators in total. One of these elevators works continuously, the rest switch to active or passive state depending on the density. The maximum capacity of the elevators is 10. The transition between floors in elevators is 200ms. There is a login thread that controls customers entering the shopping center and waiting in the elevator queue at the entrance floor, an exit thread that adds random numbers of customers to the elevator queue of that floor to exit the shopping center from other floors, and a control thread that activates or deactivates a new elevator thread according to the density. . In this project java language was used and intellij idea editor was used. At the end of the project, the predicted gains were obtained and the concept of multithreading was learned.**

Keywords—Multhreading, Java, Elevator Application, Synchronization .

I. GİRİŞ

Bu projede multithreading kavramının gerçek bir proje üzerinden uygulanması ve anlaşılması amaçlanmıştır. Projede sanal bir asansör uygulama örneğinin multi thread yapısıyla gerçekleştirilmesi amaçlanmıştır. Özetle projemizde giriş, çıkış, asansör ve kontrol threadi olmak üzere 4 adet thread bulunmaktadır. Giriş threadi 500 ms aralıklarla sürekli olarak 1-10 arasındaki sayıda müşteriyi giriş kuyruğuna ekler. Asansör kuyruğu katlar arası geçiş süresi 200 ms olmak üzere her kattaki bekleme kuyruğundaki müşterileri gitmek istedikleri kata götürmeyi amaçlar. Asansör en fazla kapasitesi olan 10 kişiyi aynı anda taşıyabilir. Çıkış threadi 1-4 arasındaki katlardan müşteri olmayan katları hariç tutarak diğer katlardan birini rasgele seçer, seçtiği bu kattaki müşterilerden 1-10 arasında müşteriyi o kata ait asansör bekleme kuyruğuna ekler. Eğer rasgele seçilen bu katta müşteri sayısı 10'dan küçük ise ilgili kattaki müşteri sayısını sınır değer olarak alıp rasgele sayıda müşteriyi asansör bekleme kuyruğuna dahil eder. Kontrol threadi ise sürekli olarak giriş katı dahil tüm katlarda asansör bekleme kuyruğundaki müşterilerin sayısını kontrol eder. Eğer bu sayı toplam asansör kapasitesinin 2 katını geçerse yeni bir asansörü pasif halden aktif hale geçirir. Bu işlem kuyruktaki müşteri sayısı 10'un altına her düşüşünde ise bir asansör pasif hale getirilir. Ama bir asansör her zaman aktif olarak çalışır. Projede ortak kullanılan objelerin senkronizasyonu sağlanmıştır. Bu isterlere göre gerekli threadler oluşturulmuş testleri yapılmıştır. Projenin detayları ilerleyen bölümlerde anlatılmıştır.

II. TEMEL BİLGİLER

Projeye başlarken öncelikle temel classlarımızı yazarak başladık. Bunlar sırasıyla login işlemlerini gerçekleştiren LoginThread, çıkış işlemlerini gerçekleştirini gerçekleştiren ExitThread, müşterilerin taşınma işlemini gerçekleştiren ElevatorThread ve asansörleri yoğunluk durumuna göre aktif veya pasif hale ControlThread'ini yazdık. Bunların birer thread olması için Thread sınıfını inherit ettik. Bunlar için istenilen işlemleri gerçeklemelerini sağlayan kodları her bir thread için ayrı ayrı yazdık. Bu threadleri daha sonra tek bir merkezden çağırdık. Asansör threadi için sadece bir sınıf üretip bu sınıftan 5 elemanlı thread dizisi oluşturduk. Bu elevator thread'ini kontrol thread'inde çağırdık. Kontrol threadinde sonsuz döngü içerisinde her seferinde main

içerisinde statik olarak concurrent elevatorQueue'yu çağırıp güncel asansör kuyruğunda bekleyen toplam müşteri sayısını hesapladık. Bu sayı eğer 20'den büyük ise yeni bir asansör aktif hale gelir. Eğer 10'dan küçük ise bir asansör pasif hale getirilir. Burada önemli nokta ise asansör threadimizi yok etmeyiz sadece suspend eder wait durumunda bekletiriz ve sonrasında yoğunluğa bağlı olarak en son suspend edilen asansörü resume ederek aktif hale getiririz.

III. YÖNTEM

A. *Lock Kullanımı Ve Synchronized Blok*

Lock kullanımının temel amacı lock ismi verilen bir object keywordu ile synchronized blok içerisindeki kodlara sadece tek bir thread tarafından erişim sağlanmasını kontrol etmektir. Biz ortak kullanılan verileri(örneğin arraylist) lock keywordu ile birlikte synchronized kod bloğu içerisinde yazarak veri uyuşmazlığınında önüne geçmiş oluruz. Çünkü bu presiple o kod bloğuna sadece bir thread erişebilecektir. İlgili thread o kod bloğunda işini bitirdiği zaman lock ismi verilen anahtarı bırakır, serbest kalan bu lock'u kapan ilk thread o kod bloğuna erişir. Biz kodumuzda ElevatorThread ve Exit Thread tarafından ortak kullanılan elevatorQueue 'yu, aynı zamanda birçok asansör tarafından erişilmesi muhtemel olan exitCount değişkenini bu şekilde tanımladık.

```
private static Object lock1 = new Object();
private static Object lock2 = new Object();
private static Object lock3 = new Object();
private static Object lock4 = new Object();
private static Object lock5 = new Object();
private static Object lock6 = new Object();
```

Şekil 1 : Javada lock tanımlama

```
public static  Queue<CustomerTransformationInfo>
getFloorsQueue(int index ){
    synchronized (lock2){
     return elevatorQueue[index];
    }
}
```

Şekil 2 : Synchronized bir kod parçası örneği

```
public static void increaseExitCount(int amount){
    synchronized (lock3){
        exitCount+= amount;
    }
}
```

Şekil 3 : Synchronized bir kod parçası örneği

```
public static void addElevatorQueue (int customerAmount,
                                     int currentFloor,
                                     int destinationFloor) {
    synchronized (lock1){

        elevatorQueue[currentFloor].
                add( new CustomerTransformationInfo
                        (customerAmount,destinationFloor));
    }

}
```

Şekil 4 : Synchronized bir kod parçası örneği

B. *ConcurrentLinkedQueue Ve CopyOnWriteArrayList Kullanımı*

Başta asansör kuyruğunu tanımlarken 5 elemanlı normal LinkedList tanımladık. Aynı zamanda asansör içerisindeki müşterileri tutmak için de normal bir arrayList tanımladık. Kodumuzun ilerleyen bölümlerinde gördük ki bu veri yapılarına birden fazla thread aynı zamanda erişmeye çalıştığı için ConcurrentModified hatası aldık. Bundan dolayı veri yapımızı başlıkta belirttiğimiz veri yapılarına çevirdik. Bu veri yapılarının faydası ise eş zamanlı olarak işlem yapmaya olanak sağlamasıdır.

Kod Örneği:

```
Public  ConcurrentLinkedQueue<Integer> destinationFloorQueue
= new ConcurrentLinkedQueue<Integer>();
public CopyOnWriteArrayList<CustomerTransformationInfo>
elevatorInside = new CopyOnWriteArrayList<>();
```

IV. GERÇEKLEŞTİRİLEN TESTLER VE DEĞERLENDİRME

Gerçekleştirdiğimiz testler ve çözülen sorunlar sonucunda multi threading olarak gerçekleştirdiğimiz proje sorunsuz olarak çalışmaktadır. Programın doğru çalıştığından emin olmak için birden çok bilgisayarda birden çok kez çalıştırdık. Bu proje sonunda multi threading kavramını anlamış, gerçek bir projeye uygulamış olduk. Kodumuzun çıktısı aşağıdaki gibi olmaktadır.

```
1. floor :all :3
2. floor :all :2
3. floor :all :0
4. floor :all :0
active  :True
         asansor : 0
         floor:0
         destinaton Amount:1
         direction:UP
         capacity:10
         countInside:5
         hedef kuyrugu : [1]
         inside:[(1,5)]

active :False
         asansor : 1
         floor:0
         destinaton Amount:0
         direction:N
         capacity:10
         countInside:0
         hedef kuyrugu : []
         inside:[]

active :False
         asansor : 2
         floor:0
         destinaton Amount:0
         direction:N
         capacity:10
```

Şekil 5 : Programın çıktısı

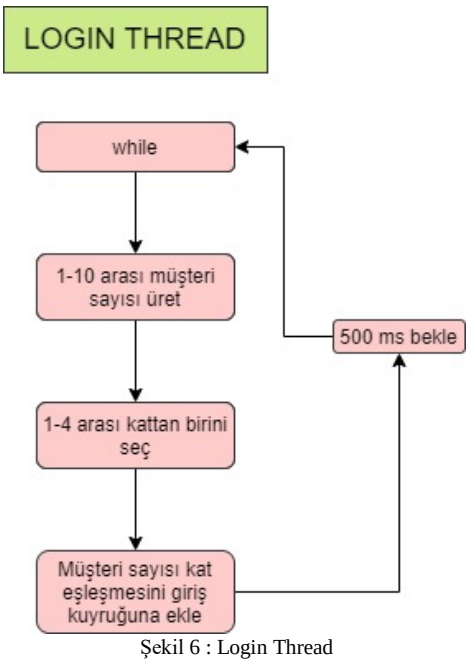
V. ÖZET VE SONUÇLAR

Yaptığımız bu proje sonucunda bizden istenilen tüm isterleri eksiksiz olarak uyguladık. Gerçekleştirilen testler sonucu da bunun hatasız olarak çalıştığını gördük. Bu projeyle birlikte multhreading kavramını daha iyi anlamış olduk. Bu proje an itibari ile gelişime açık durumdadır.

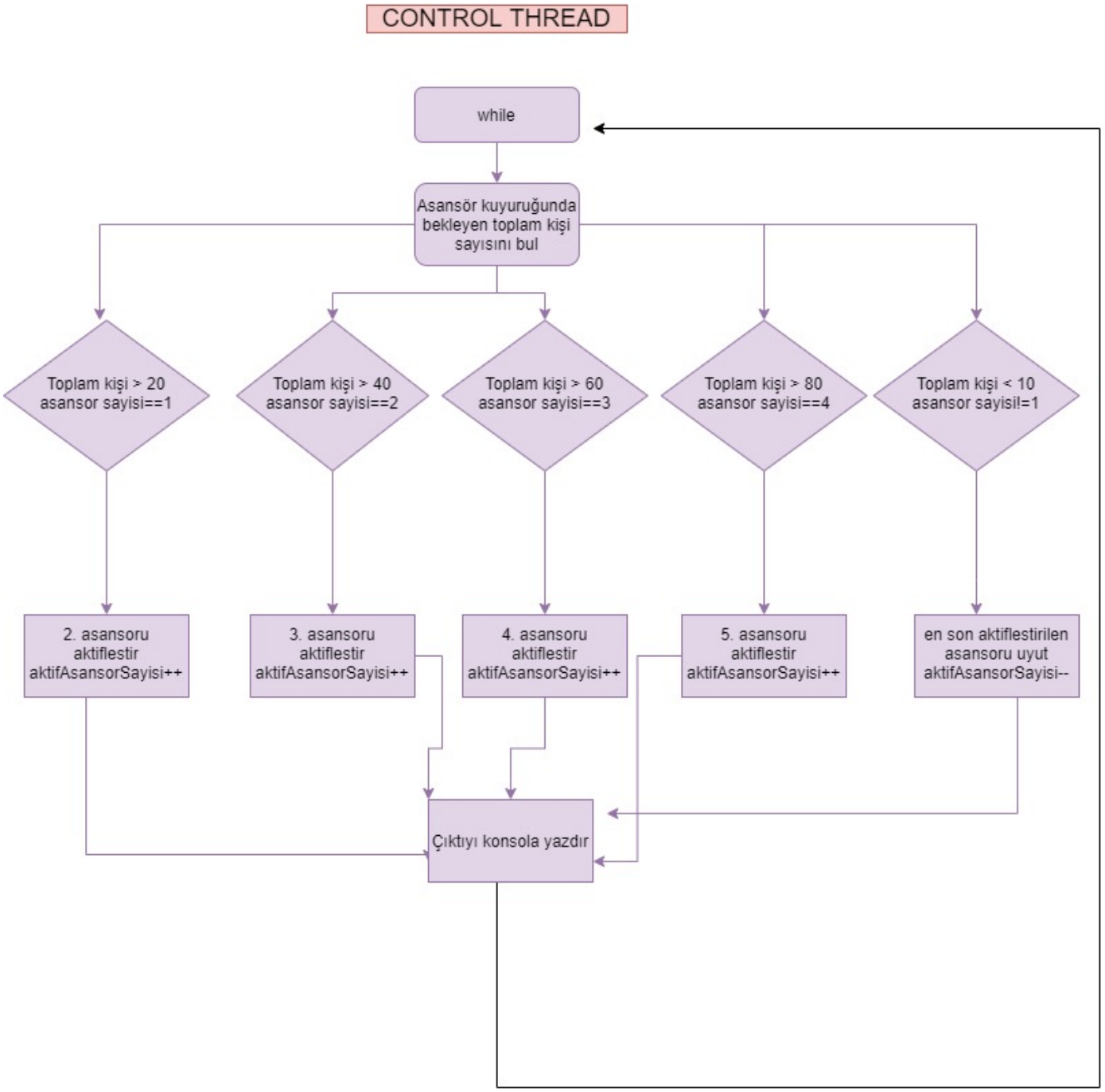
KAYNAKÇA

[1] https://stackoverflow.com/
[2] https://www.javatpoint.com/
[3] https://www.youtube.com/
[4] https://www.udemy.com/

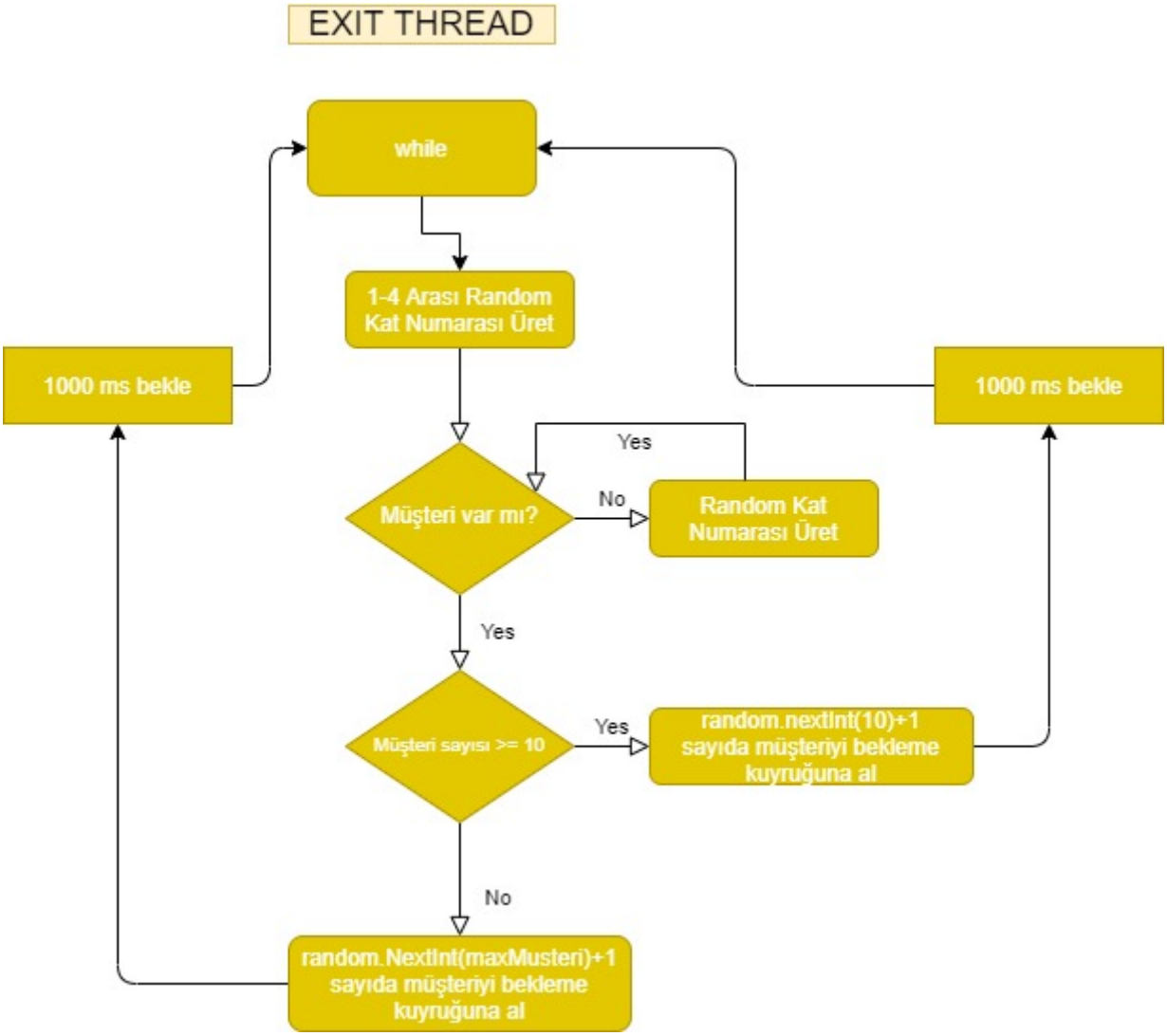
VI. Projenin Akış Diyagramları



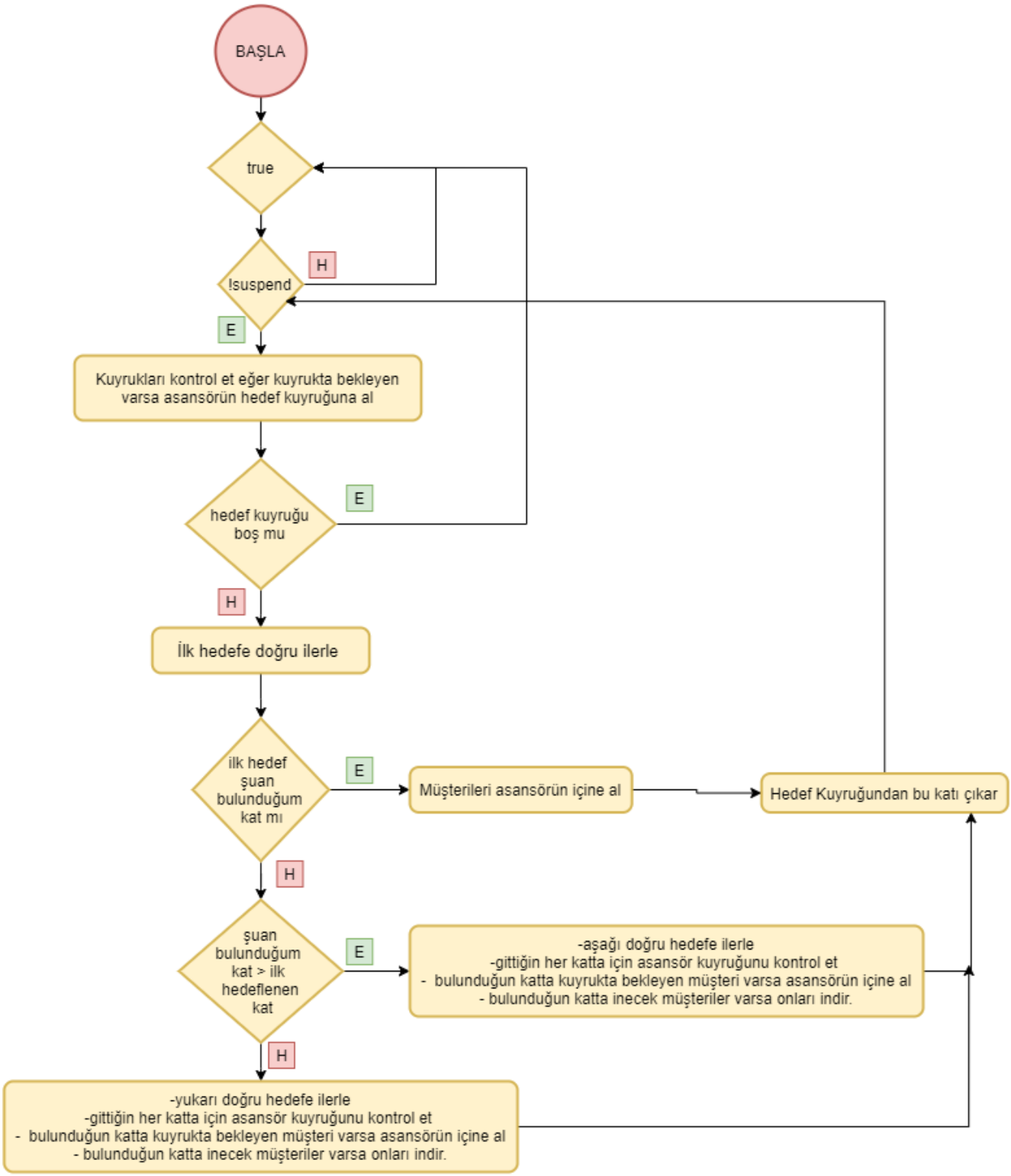
Şekil 6 : Login Thread



Şekil 7 : Control Thread



Şekil 8 : Exit Thread



ElevatorThread in Akış Şeması

Şekil 9 : Elevator Thread