ASANSÖRLERDEKİ TALEP YOĞUNLUĞUNUN MULTITHREAD İLE KONTROLÜ

Furkan Aydoğan
Bilgisayar Mühendisliği, Kocaeli Üniversitesi
180202085

Sefa Berke Kara
Bilgisayar Mühendisliği, Kocaeli Üniversitesi
180202086

GİRİŞ

Bu projenin amacı bir AVM'deki asansörlere gelen isteklerdeki yoğunluğu, multithread kullanarak diğer asansörlerle birlikte azaltmaktır.

Özet

Asansör multithread projesi 5 adet asansör thread ,1 adet kontrol thread ve giriş-çıkış threadlerinden oluşmaktadır. Asansör threadlerdeki yoğunluğu azaltmak için eş zamanlı olarak program çalışmaktadır. Katlardaki yoğunluk Kontrol_Thread classında kontrol edilir. Toplam katlardaki yoğunluk sayısı asansör sayısının 20 katını geçtiğinde yeni bir asansör aktif edilir. Her 500 ms de bir random (1-10 arası)yolcu AVM'ye giriş yapar ve her 1000 ms. de bir AVM'den belli sayıda random (1-5 arası) yolcu çıkış yapar. Bu sayede threadler eş zamanlı çalışarak katlardaki yoğunluğu engelleyerek yolcuların talepleri yerine getirilmiş olur.

Temel Bilgiler

Windows 10 işletim sistemine sahip bilgisayar ile geliştirme yapılmıştır. Program Java programlama dilinde geliştirilmiş olup, tümleşik geliştirme ortamı olarak "Netbeans" kullanılmıştı.

Tasarım

Asansörlerdeki talep yoğunluğunun multithread ile kontrolü projesi programlanma aşamaları altta belirtilen başlıklar altında açıklanmıştır.

Algoritma

Asansör multithread projesi ilk olarak her katın kendine özgü tutacağı kuyruk bilgisi için kat_bilgi[] dizisi oluşturur ve kuyruktaki sayı belirli bir sayı olmadığından dolayı bu bir Asansor_kuyruk tipinde ArrayLise eşitlenir. Her kattaki kat isteklerini kontrol etmesi için kat_yönetici sınıfından nesne oluşturulur Asansor_Thread classından asansörlerin bilgilerin tutulması için 5 tane dizi oluşturulur. Ardından for döngüsü asansör sayısı kadar döndürülür. Bu For döngüsü içerisinde her bir asansör için nesne oluşturulur ve parametreli constructora asansörler için değerler atanır. Bu oluşturulan nesne Thread dizisine ve asansörlerin bilgileri için oluşturulan asansörler[] dizisine eklenir. kat_yöneticisi classındaki asasor_al metodu ile asansörlerin bilgilerini tutar. Th3 (Kontrol thread) adındaki Kontrol_thread classından oluşturulan thread katlardaki yoğunluğu kontrol etmesi için oluşturulur. Giriş asansörü yani asansör-0 her zaman devamlı olarak çalışması için "working" durumuna getirilir Diğer asansörler de ""idle durumuna getirilerek bütün asansörler threadleri başlatılır. Giriş ve Çıkış threadleri oluşturulur ve başlatılır ardından kontrol threadi ile ara yüzde başlatılarak uygulama ekrana bastırılır. Bu işlemlerin ardından "while" döngüsünde program devamlı olarak çalışarak ara yüzdeki güncellenen değerleri her 1 ms. de bir tekrar ederek günceller. Asansörlerin threadleri başladığında asansor_Thread classındaki run fonksiyonu çalışır. Burada "while" döngüsüne girerek bütün threadlerin sürekli çalışması sağlanır. isKontrol_Asansor methodunda false değerine eşit olan asansörler 100 ms. uyutulur , True değerine eşit olanlar ise katlara yönlendirilerek taşıma işlemine başlar. İf koşulu içindeki this.asansor_durum == true olduğunda Asansör durumu da true olarak set edilerek çalışıyor durumuna getirilir. Ardından kontrol işlemlerine başlanır. Yolcu sayısı sıfır ise ve hareket etmiyorsa if koşuluna girer. Burada asansör_thread classındaki getKontrol() metoduna çağırarak dönen değer ile kat_yönetici sınıfına gider. Kat_Yoneticisi sınıfında istek_Varmi metoduna parametreleri yollayarak katlardaki istekleri kontrol eder. Bu metot altında asansörün bulunduğu kat sıfırıncı veya birinci kat ise if koşuluna girer. Asansörün bulunduğu pozisyondaki kat bilgilerini Kat_Bilgi tipinde kat değişkeni tutar. İlk önce asansör bulunduğu katta istek var mı kontrol eder. Eğer var ise asansör_hareket sınıfından nesne oluşturularak return eder. Eğer bulunduğu katta herhangi bir istek yok ise 0. Kattan 5. Kata kadar istekleri kontrol eder. Hangi katta istek var ise asansör o kata doğru yol alır. Asansörün bulunduğu konumundan aşağıda bir istek var ise asansör yönü aşağıya, yukarı yönden istek gelirse de yukarı doğru tanımlanır. İstek yapılan kat ve o kata kaç ms de gidileceği değerleri parametre olarak Asansör_hareket parametreli constructoruna gönderilir ve yeni nesne oluşturularak return edilir. Eğer else koşuluna girerse yani asansör konumu 2-3-4 veya 5 .katta ise if koşuluna benzer yolları takip ederek devam eder. Eğer hiçbir yerde istek yok ise de null değeri döndürülür. Return edilen değer null dan farklı ise Asansor_thread sınıfındaki if koşuluna girer. Burada varacağı kat hedef_kat değişkenine eşitlenir. Dönen değer hareket metodu ile asansörün kendisine ait hareket ArrayListine eklenir.Eğer varacağı kat bulunduğu kata eşit ise "Asansör kapıyı açıyor " uyarısını bastırır. Eğer değil ise "kata almak için gidiyor" uyarısını bastırır. Eğer istek var ise else if koşuluna girer. Asansörün varacağı kat hedef_kat değişkenine eşitlenir. Asansörün varış süresi program çalışma süresinden kısa veya eşitse ilk if koşuluna gir asansörün varacağı katı bulunduğu_kat değişkenine eşitler. Eğer yolcu sayısı 0 ise ve bulunduğu kata varmış

ise asansörün bir önceki hareketi ArrayListten kaldırılır. Bu işlemin ardından Asansor_Thread sınıfındaki yolcu_yukle2() metoduna asansörün bulunduğu_kat asansörün ID'si ve asansörün maximum kapasitesi parametreleri gönderilir. Bu fonksiyon içinde Asansor_kuyruk tipinde inside adında bir ArrayList oluşturulur. Eğer kattaki kuyruk boş değil ise if koşuluna girer. Burada do-while döngüsü kattaki kuyruğun sonuna kadar döndürülür. Kattaki kuyruk en baştan başlayarak ve asansörün kapasitesinden küçük olduğu sürece kattaki kuyruğu asansörün içine alır. Eğer asansörün kapasitesi kuyruktaki toplam kişi sayısından daha az ise fakat kuyrukta asansörün içine alınabilecek kadar gruplu halde kişiler var ise asansörün kapasitesi yettiği sürece asansör içine alınır. Bu işlemler yapıldıktan sonra içeriye alınan kişiler return edilir ve Asansor_thread sınıfındaki inside ArrayListine tüm kuyruk eklenir. Eğer asansör boş değil ise ekrana yolcuların alındığına dair mesaj bastırılır ve Asansor_thread sınıfındaki yolcu_yukle fonksiyonuna gidilir. Bu fonksiyon içerisinde asansör içindeki kişilerinin hangi katlara gideceğinin hareketleri ArrayListde tutulur ve aynı zamanda asansör içine alınan yolcu sayısı asansörün içine eklenir. Asansörün içindeki kuyruk ilk gidecek kişiden son gidecek kişiye kadar sıralanarak ardından inside ArrayListi güncellenir. En son olarak da ilk gidilecek katın hareketi return edilir. Return edilen değer Asansor_Hareket tipindeki first_event değişkeninde tutulur. Eğer asansörün içi boş ise ve istek gelen kata vardıysa oraya vardıktan sonra isteği false değerine eşitler .Eğer varacağı kat yolcuları aldıktan sonra hedeflediği kata geldiyse else koşuluna girerek yolcu_indir fonksiyonuna hedeflediği katı parametre olarak yollar. Varacağı kata vardıktan sonra önceki hareketini remove() eder ve indirdiği kişi sayısı kadar asansörün içindeki kişi sayısını çıkartır. Eğer hedeflenen kat sıfır ise Kat_Yonetici sınıfındaki setCikis_yapan metotuna ara yüze bastırmak için çıkış yapan yolcuları ekler. Eğer sıfırdan farklı bir kata gidiliyorsa o katta bulunan kişilerin sayısını ekleme yapar. Ardından asansör içindeki kişileri indirdikçe de asansör den remove eder. Yolcuları indirdikten sonra asansör_id sıfırdan farklı ise yolcu_sayisi sıfıra eşit ise this.setKontrol_asansor(false) yapılır. Yukarıdaki işlemler yapılırken aynı zamanda kontrol threadi başlatıldığı için Kontrol_thread sınfındaki işlemlerde 100 ms. bir eş zamanlı olarak Asansor_Kontrol metodu çağırılır. Burada Asansor_Kontrol metodunda her kattaki toplam kuyruk sayısı alınır. Eğer istek sayısı kadar yoğunluk oluştuğu durumda her 20 kişide bir diğer asansör aktif edilir. Bu durum 80 kişiye kadar devam eder. Yoğunluk azaldığı durumda da (yirminin katları şeklinde olması koşuluyla) aktif olan asansörler pasif hale getirilir. Eş zamanlı olarak giriş threadi de 500 ms de bir giriş üretir. Bu giriş üretimi şu şekilde yapılır.Asansor_kuyruk sınıfından bir nesne oluşturulur.1-10 arasında random bir sayı alınarak giriş_yolcu değişkenine eşitlenir .Ardından asansor_kuyruk'dan oluşturulan nesnenin yolcu sayısına set edilir.1-4 arasında sayı oluşturularak setUlasilacak_kat metoduna set edilir.Ardından sıfırıncı kata giriş yapanlar asansor_Kuyruk sınıfındaki kat_bilgi Arraylistine eklenir ve kattaki kuyruğa eklenebilmesi içinde kat_bilgi ArrayListine eklenen değişkenler setKuyruk metotu ile sıfırıncı kata eklenir ,kuyruk oluşturulur. Kuyruğa eklendikten sonra da o katta istek olduğuna dair istek_kontrol metotunda o katın değişkeni true yapılır. Eş zamanlı olarak çıkış threadi de 1000 ms de bir çıkış üretir. Bu çıkış üretimi şu şekilde yapılır. Asansor_kuyruk sınıfından kuyruk adında bir nesne oluşturulur. Hangi kattan çıkış yapılacağı 1-4 arasında bir sayı üretilerek a değişkenine eşitlenir. üretilen sayı için katta kuyruk olup olmadığı kontrol edilir. Eğer kattaki kuyrukta kimse yok ise tekrardan 1-4 arasında bir sayı üretilir eğer var ise if koşuluna girer. Burada Kat_Yonetici üzerinden katta bulunan kuyruktaki kişi sayısı c değişkenine eşitlenir. Eğer c sayısı beşten büyük ise beşe eşitle. Eğer değil ise yolcu sayısı kadar çıkış üretilip b değişkenine atılır. Ardından çıkış yapılacak yolcu sıfırıncı kattan çıkış yapacağı için kuyruk.setUlasilacak_kat(0) olarak atanır ,yolcu sayısı da atanır. Bu atanan değerlerde kat_Bilginin hangi kattan çıkış yapıldıysa o kata eklenir. Bu adımlar takip edildikten sonra break yaparak do-while döngüsü içinden çıkar. Bundan sonra kat_bilgi ArrayListinde tutulan kat bilgileri kat_yöneticisinden hangi kattan çıkış yapılacaksa o katın kuyruğuna eklenir. Kuyruğa eklendikten sonra da o katta istek olduğuna dair istek_kontrol metodunda o katın değişkeni true yapılır.

Sözde Kod

1- kat_bilgi[] dizisi oluştur.
2- Bu diziyi Asansor_kuyruk tipinde ArrayListe eşitle.
3- kat_yönetici sınıfından nesne oluştur.
4- Asansor_Thread classından asansörlerin bilgilerin tutulması için 5 tane dizi oluştur.
5- Ardından for döngüsü asansör sayısı kadar döndür.
6- Bu For döngüsü içerisinde her bir asansör için nesne oluşturulur ve parametreli constructora asansörler için değerler ata.
7- Bu oluşturulan nesne Thread dizisine ve asansörlerin bilgileri için oluşturulan asansörler[] dizisine ekle.
8- kat_yöneticisi classındaki asansor_al metodu ile asansörlerin bilgilerini tut.
9- Th3 (Kontrol thread) adındaki Kontrol_thread classından oluşturulan thread katlardaki yoğunluğu kontrol etmesi için oluştur.
10- Giriş asansörü yani asansör-0 her zaman devamlı olarak çalışması için "working" durumuna getir.
11- Diğer asansörler de "idle" durumuna getirilerek bütün asansörler threadleri başlat.
12- Giriş ve Çıkış threadleri oluştur ve başlat.
13- Kontrol threadi ile ara yüzde başlatıp uygulama ekrana bastır.
14- Bu işlemlerin ardından "while" döngüsünde program devamlı olarak çalışarak ara yüzdeki güncellenen değerleri her 1 ms. de bir tekrar ederek güncelle.
15- Asansörlerin threadleri başladığında asansor_Thread classındaki run fonksiyonu çalıştır.
16- "while" döngüsüne girerek bütün threadlerin sürekli çalışmasını sağla.
17- isKontrol_Asansor methodunda false değerine eşit olan asansörler 100 ms. uyut,
18- True değerine eşit olanlar ise katlara yönlendirilerek taşıma işlemine başlat.
19- İf koşulu içindeki this.asansor_durum == true olduğunda Asansör durumu da true olarak set ederek çalışıyor durumuna getir.
20- Kontrol işlemlerine başla.
21- Yolcu sayısı sıfır ise ve hareket etmiyorsa if koşuluna gir.
22- Burada asansor_thread classındaki getKontrol() metoduna çağırarak dönen değer ile kat_yönetici sınıfına git.

23- Kat_Yoneticisi sınıfında istek_Varmi metoduna parametreleri yollayarak katlardaki istekleri kontrol et.
24- Bu metot altında asansörün bulunduğu kat sıfırıncı veya birinci kat ise if koşuluna gir.
25- Asansörün bulunduğu pozisyondaki kat bilgilerini Kat_Bilgi tipinde kat değişkeni tut.
26- İlk önce asansör bulunduğu katta istek var mı diye kontrol et.
27- Eğer var ise asansör_hareket sınıfından nesne oluşturularak return et.
28- Eğer bulunduğu katta herhangi bir istek yok ise 0. Kattan 5. Kata kadar istekleri kontrol et.
29- Hangi katta istek var ise asansör o kata doğru yol alır.
30- Asansörün bulunduğu konumundan aşağıda bir istek var ise asansör yönü aşağıya ,yukarı yönden istek gelirse de yukarı doğru tanımla.
31- İstek yapılan kat ve o kata kaç ms de gidileceği değerleri parametre olarak Asansor_hareket parametreli constructoruna gönder
32- Yeni nesne oluşturularak return et.
33- Eğer else koşuluna girerse yani asansör konumu 2-3-4 veya 5.katta ise if koşuluna benzer yolları takip ederek devam et.
34- Eğer hiçbir yerde istek yok ise de null değeri döndür.
35- Return edilen değer null dan farklı ise Asansor_thread sınıfındaki if koşuluna gir.
36- Varacağı kat hedef_kat değişkenine eşitle.
37- Dönen değer hareket metodu ile asansörün kendisine ait hareket ArrayListine ekle.
38- Eğer istek var ise else if koşuluna gir.
39- Asansörün varacağı kat hedef_kat değişkenine eşitle.
40- Asansörün varış süresi program çalışma süresinden kısa veya eşitse ilk if koşuluna gir
41- Asansörün varacağı katı bulunduğu_kat değişkenine eşitle.
42- Eğer yolcu sayısı 0 ise ve bulunduğu kata varmış ise asansörün bir önceki hareketi ArrayListten kaldır.
43- Asansor_Thread sınıfındaki yolcu_yukle2() metoduna asansörün bulunduğu_kat asansörün ID'si ve asansörün maximum kapasitesi parametreleri gönder.
44- Bu fonksiyon içinde Asansor_kuyruk tipinde inside adında bir ArrayList oluştur.
45- Eğer kattaki kuyruk boş değil ise if koşuluna gir.
46- Burada do-while döngüsü kattaki kuyruğun sonuna kadar döndür.
47- Kattaki kuyruk en baştan başlayarak ve asansörün kapasitesinden küçük olduğu sürece kattaki kuyruğu asansörün içine al.
48- Eğer asansörün kapasitesi kuyruktaki toplam kişi sayısından daha az ise fakat kuyrukta asansörün içine alınabilecek kadar gruplu halde kişiler var ise asansörün kapasitesi yettiği sürece asansör içine al.
49- Bu işlemler yapıldıktan sonra içeriye alınan kişiler return et ve Asasnor_thread sınıfındaki inside ArrayListine tüm kuyruğu ekle.
50- Eğer asansör boş değil ise ekrana yolcuların alındığına dair mesaj bastı ve Asansor_thread sınıfındaki yolcu_yukle fonksiyonuna git.
51- Bu fonksiyon içerisinde asansör içindeki kişilerinin hangi katlara gideceğinin hareketleri ArrayListde tut ve aynı zamanda asansör içine alınan yolcu sayısı asansörün içine ekle
52- Asansörün içindeki kuyruk ilk gidecek kişiden son gidecek kişiye kadar sırala.
inside ArrayListi güncelle
53- En son olarak da ilk gidilecek katın hareketi return et.
54- Return edilen değer Asansor_Hareket tipindeki first_event değişkeninde tut .
55- Eğer asansörün içi boş ise ve istek gelen kata vardıysa oraya vardıktan sonra isteği false değerine eşitle.
56- Eğer varacağı kat yolcuları aldıktan sonra hedeflediği kata geldiyse else koşuluna girerek yolcu_indir fonksiyonuna hedeflediği katı parametre olarak yolla.
57- Varacağı kata vardıktan sonra önceki hareketini remove() eder ve indirdiği kişi sayısı kadar asansörün içindeki kişi sayısını çıkart.
58- Eğer hedeflenen kat sıfır ise Kat_Yonetici sınıfındaki setCikis_yapan metotuna arayüze bastırmak için çıkış yapan yolcuları ekle.
59- Eğer sıfırdan farklı bir kata gidiliyorsa o katta bulunan kişilerin sayısını ekleme yap.
60- Asansör içindeki kişileri indirdikçe de asansörden remove et.
61- Yolcuları indirdikten sonra asansör_id sıfırdan farklı ise yolcu_sayisi sıfıra eşit ise this.setKontrol_asansor(false) yap.
62- Yukarıdaki işlemler yapılırken aynı zamanda kontrol threadi başlatıldığı için Kontrol_thread sınıfındaki işlemlerde 100 ms. de bir eş zamanlı olarak Asansor_Kontrol metodunu çağır.
63- Asansor_Kontrol metodunda her kattaki toplam kuyruk sayısı al.
64- Eğer istek sayısı kadar yoğunluk oluştuğu durumda her 20 kişide bir diğer asansör aktif et.
65- Bu durum 80 kişiye kadar devam et.
66- Yoğunluk azaldığı durumda aktif olan asansörler pasif hale getir.
67- Eş zamanlı olarak giriş threadi de 500 ms de bir giriş üret.
68- Bu giriş üretimi şu şekilde yapılır.Asansor_kuyruk sınıfından bir nesne oluştur.
69- 1-10 arasında random bir sayı alınarak giriş_yolcu değişkenine eşitle.
70- asansör_kuyruk'dan oluşturulan nesnenin yolcu sayısına set et.
71- 1-4 arasında sayı oluşturularak setUlasilacak_kat metotuna set et.
72- Sıfırıncı kata giriş yapanlar asansör_Kuyruk sınıfındaki kat_bilgi Arraylistine ekle.
73- Kattaki kuyruğa eklenebilmesi içinde kat_bilgi ArrayListine eklenen değişkenler setKuyruk metodu ile sıfırıncı kata ekle ve kuyruk oluştur.
74- Kuyruğa eklendikten sonra da o katta istek olduğuna dair istek_kontrol metotunda o katın değişkeni true yap.
75- Eş zamanlı olarak çıkış threadi de 1000 ms de bir çıkış üret.
76- Asansor_kuyruk sınıfından kuyruk adında bir nesne oluştur.
77- Hangi kattan çıkış yapılacağı 1-4 arasında bir sayı üretilerek a değişkenine eşitle.
78- Üretilen sayı için katta kuyruk olup olmadığı kontrol et.
79- Eğer kattaki kuyrukta kimse yok ise tekrardan 1-4 arasında bir sayı üret.
80- Eğer var ise if koşuluna gir.
81- Burada Kat_Yonetici üzerinden katta bulunan kuyruktaki kişi sayısı c değişkenine eşitle.

82- Eğer c sayısı beşten büyük ise beşe eşitle.
83- Eğer değil ise yolcu sayısı kadar çıkış üretilip b değişkenine ata.
84- Çıkış yapılacak yolcu sıfırıncı kattan çıkış yapacağı için kuyruk.setUlasilacak_kat(0) olarak ata ,yolcu sayısı da atanır.
85- Atanan değerlerde kat_Bilginin hangi kattan çıkış yapıldıysa o kata ekle.
86- Bu adımlar takip edildikten sonra break yaparak do-while döngüsü içinden çık.
87- kat_bilgi ArrayListinde tutulan kat bilgileri kat_yöneticisinden hangi kattan çıkış yapılacaksa o katın kuyruğuna ekle.
88- Kuyruğa eklendikten sonra da o katta istek olduğuna dair istek_kontrol metodunda o katın değişkeni true yap.

Karşılaştığımız Sorunlar

Threadleri ilk kez öğrenip bu projede ilk defa kullandığımız için proje iskeletini kurmakta ve eş zamanlı olarak çalıştırmakta zorlandık.

Threadler çok CPU kullandığı için donanımızda bazı zamanlarda donmalar yaşadık.

Projenin Bize Kattığı Yararlar

MultiThreading kullanımını öğrendik.

Bir projeyi nasıl senkronize bir şekilde çalıştırmamız gerektiğini öğrendik.

Kullanılan Fonksiyonlar

**** Basla.java**

public synchronized void araYuz()

Program ara yüzünün bastırıldığı fonksiyondur.

public synchronized void Guncelle()

Sürekli değişen değerleri JFrame de günceller.

**** Asansor_thread.java**

GETTER VE SETTERLER

public int getHedef_kat()

public ArrayList<Asansor_kuyruk> getInside()

public void setHedef_kat(int hedef_kat)

public String getYon()

public void setYon(String yon)

public void setAsansor_durum(boolean asansor_durum)

public int getBulundugu_kat()

public int getYolcu_sayisi()

public void setYolcu_sayisi()

public int getUlasilacak_kat()

public void setUlasilacak_kat()

public synchronized void zamanArttir()

public synchronized int getZaman()

public synchronized boolean isKontrol_asansor()

public synchronized void setKontrol_asansor(boolean kontrol_asansor)

public int getYolcu_sayisi()

public int getAsansor_id()

public boolean isAsansor_durum()

public int getIndirdigi_yolcu_sayi()

public synchronized void setIndirdigi_yolcu_sayi(int indirdigi_yolcu_sayi)

public int getMax_kapasite()

public int getToplam_aldigi_yolcu()

public synchronized void setToplam_aldigi_yolcu(int toplam_aldigi_yolcu)

public ArrayList<Asansor_hareket> getHareket()

public Kat_Yonetici getKontrol()

public void hareket(Asansor_hareket hareket)

public String getAsansor_mod()

public void setAsansor_mod(String asansor_mod)

public void hareket(List<Asansor_hareket> events)

Metotlar

public synchronized Asansor_hareket yolcu_yukle(ArrayList<Asansor_kuyruk> kuyruk)

Kuyruğu alıp asansörün hangi kata gideceğinin hareketlerini yolcu istekleri doğrultusunda belirler ve hareket ArrayListine ekler. İlk gideceği katı return eder.

public synchronized void yolcu_indir(int kat)

Asansör hedeflediği kata varınca içerideki yolcuları vardığı kata indirir.

public synchronized ArrayList<Asansor_kuyruk> yolcu_yukle2()

Katlardaki kuyruğu asansör kapasitesini geçmeyecek şekilde kontrol ederek asansöre alır ve return eder.

**** Kat_Yonetici.java**

GETTER VE SETTERLAR

public void asansor_al(Asansor_Thread[] asansorler)

public int getCikis_yapan()

public void setCikis_yapan(int cikis_yapan)

public Kat_Bilgi[] getKat_bilgi()

public synchronized void istek_kontrol(int istek_kontrol)

METOTLAR

public synchronized Asansor_hareket istek_Varmi()

Aldığı parametrelerle asansörün bulunduğu katta mı yoksa farklı katta mı olup olmadığını kontrol eder ve asansörün o kata gitmesi için hareketi return eder.

**** Asansor_hareket.java**

public Asansor_hareket(int varacagi_kat,int varis_suresi)

public int getVaracagi_kat()

public int getVaris_suresi()

**** Asansor_kuyruk.java**

public int getYolcu_sayisi()

public void setYolcu_sayisi()

public int getUlasilacak_kat()

public void setUlasilacak_kat(int ulasilacak_kat)

**** Kat_Bilgi.java**

- GETTER VE SETTERLAR

public int getKuyruk_sayisi()

public void setKuyruk_sayisi(int kuyruk_sayisi)

public synchronized void setSanal_katta_bulunan(int sanal_katta_bulunan)

public synchronized void sanal_sil(int temp)

public int getKatta_bulunan_kisi()

public synchronized void setKatta_bulunan_kisi(int katta_bulunan_kisi)

public int getYaklasan_asansor()

public void setYaklasan_asansor(int yaklasan_asansor)

public ArrayList<Asansor_kuyruk> getKuyruk()

public void setKuyruk(ArrayList<Asansor_kuyruk> kuyruk)

public synchronized void yolcu_sil(int yolcu_sayisi)

public boolean isYolcu_istek()

public void setYolcu_istek(boolean yolcu_istek2)

**** Kontrol_Thread.java**

public synchronized void Asansor_kontrol()

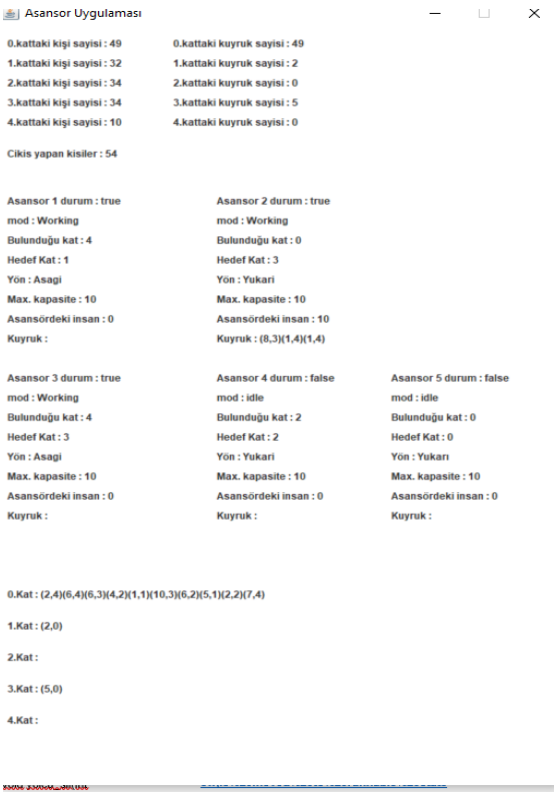
Tüm katlardaki toplam kuyruk sayısını yoğunluğa göre asansörün çalışıp çalışmayacağını belirer.

**** Zaman.java**

public synchronized void zamanArttir()

public synchronized int getZaman()

Ekran Görüntüleri



Kaynakça

Synchronized in Java

https://www.geeksforgeeks.org/synchronized-in-java/

Multithreading in Java

https://www.geeksforgeeks.org/multithreading-in-java/

Lifecycle and States of a Thread in Java

https://www.geeksforgeeks.org/lifecycle-and-states-of-a-thread-in-java/#:~:text=New%20Thread%3A%20When%20a%20new,is%20moved%20to%20runnable%20state