ASANSÖRLERDEKİ TALEP YOĞUNLUĞUNUN MULTİTHREAD İLE KONTROLÜ

Aziz Yelbay – 170201046, Yusuf Bayraktar – 170201070

aziz1594@hotmail.com , y.sancaktar98@gmail.com

Bilgisayar Mühendisliği Bölümü
Kocaeli Üniversitesi

Özet

Yazılım Laboratuvarı 1 dersi ikinci projesinde İşletim Sistemleri dersinde edinilen bilgilerin uygulamaya geçirilmesi amaçlanmıştır.

Bu proje bir AVM' deki asansörlere gelen istekleri multithread kullanımıyla kuyrukta bekleyen müşteri sayısına göre diğer asansörlerin aktif ya da pasif hale getirilerek yoğunluğun azalmasını sağlamaktadır.

1. Giriş

Amaç 5 katlı bir AVM' deki asansörlere gelen isteklerdeki yoğunluğu, multithread kullanarak diğer asansörlerle birlikte azaltmaktır. AVM' ye her 500 milisaniyede bir 1 ve 10 arasında rastgele müşteri giriş yapmaktadır ve 1 ile 4 arasında rastgele katlara gitmek isteyeceklerdir. Bu gelen müşteriler direkt olarak giriş kattaki kuyruğa girerler. Her 1000 milisaniyede bir ise 1 ve 4 arasında rastgele bir kattan 1 ve 5 arasında rastgele müşteri AVM' den çıkış yapmak istemektedir ve bu müşteriler de bulunduğu kattaki kuyruğa girmektedirler. AVM' de toplam 5 asansör bulunmaktadır. Bu asansörlerden birisi her zaman aktif durumdadır ve kesinlikle pasif duruma getirilemez, diğer 4 asansör ise istek yoğunluğuna göre aktif ya da pasif hale gelmektedir. Tüm asansörlerin maksimum kapasitesi eşittir ve her birininki 10' dur. Asansörlerdeki kat arası geçiş süresi 200 milisaniyedir.

2. Temel Bilgiler

Program JAVA nesneye yönelik programlama dilinde geliştirilmiş olup, tümleşik geliştirme ortamı olarak NetBeans IDE 8.2 kullanılmıştır.

3. Tasarım

Asansörlerdeki Talep Yoğunluğunun Multithread ile Kontrolü Projesinde tasarım geliştirme aşamaları altta belirtilen başlıklar altında açıklanmıştır.

3.1. Sınıfların Özellikleri

LoginThread.Java :
Bu sınıf Runnable Interface' inden implement edilmiştir. Runnable interface 'inde bulunan run metodu override edilmiştir. Sınıfımız bu metodun içinde işlemlerini gerçekleştirmektedir. Her 500 milisaniyede bir 1 ve 10 arasında rastgele müşterinin giriş yapmasını ve bu müşterilerin 1 ve 4 arasında rastgele bir kata gitmek için istek oluşturmasını sağlamaktadır. Bu sınıf, Project.Java main sınıfının içinde start(); metoduyla çalıştırılmaktadır.

ExitThread.Java :
Bu sınıf Runnable Interface' inden implement edilmiştir. Runnable interface' inde bulunan run metodu override edilmiştir. Sınıfımız bu metodun içinde işlemlerini gerçekleştirmektedir. Her 1000 milisaniyede bir 1 ve 4 arasında rastgele bir kattan 1 ve 5 arasında rastgele müşterinin çıkış yapmak için yani 0. kat olan giriş kata gitmek için istek oluşturmasını sağlamaktadır. İstekler sadece çıkış için yapılmaktadır. Yani herhangi bir kattan 1,2,3 ve 4. katlara gitmek için istek oluşturulmamaktadır. Bu sınıf, Project.Java main sınıfının içinde start(); metoduyla çalıştırılmaktadır.

Asansor1.Java :
Bu sınıf Runnable Interface' inden implement edilmiştir. Runnable interface' inde bulunan run metodu override edilmiştir. Sınıfımız bu metodun içinde işlemlerini gerçekleştirmektedir. Sınıfımız katlardaki kuyrukları kontrol eder. Maksimum kapasite olan 10' u aşmayacak şekilde kuyruktaki

müşterilerin talep ettikleri katlara ulaşabilmesini sağlar. Bu asansör sürekli aktif haldedir. Hiçbir zaman pasif hale getirilemez.

Asansor2.Java :
Bu sınıf Runnable Interface' inden implement edilmiştir. Runnable interface' inde bulunan run metodu override edilmiştir. Sınıfımız bu metodun içinde işlemlerini gerçekleştirmektedir. Sınıfımız katlardaki kuyrukları kontrol eder. Maksimum kapasite olan 10' u aşmayacak şekilde kuyruktaki müşterilerin talep ettikleri katlara ulaşabilmesini sağlar. Başlangıçta pasif haldedir. KontrolThread katlardaki kuyrukları kontrol edip kuyruklarda bekleyen toplam müşteri sayısı 20' yi geçtiğinde ilk olarak bu asansörü aktif hale getirir. Eğer Asansör3 pasif durumda ve bu asansör aktif durumdayken katlardaki kuyruklarda bekleyen toplam müşteri sayısı 10' un altına düşerse bu asansör KontrolThread tarafından pasif hale getirilir.

Asansor3.Java :
Bu sınıf Runnable Interface' inden implement edilmiştir. Runnable interface' inde bulunan run metodu override edilmiştir. Sınıfımız bu metodun içinde işlemlerini gerçekleştirmektedir. Sınıfımız katlardaki kuyrukları kontrol eder. Maksimum kapasite olan 10' u aşmayacak şekilde kuyruktaki müşterilerin talep ettikleri katlara ulaşabilmesini sağlar. Başlangıçta pasif haldedir. KontrolThread katlardaki kuyrukları kontrol edip kuyruklarda bekleyen toplam müşteri sayısı 20' yi geçtiğinde Asansör1 ve Asansör2 aktifse bu asansörü aktif hale getirir. Eğer Asansör4 pasif durumda ve bu asansör aktif durumdayken katlardaki kuyruklarda bekleyen toplam müşteri sayısı 10' un altına düşerse bu asansör KontrolThread tarafından pasif hale getirilir.

Asansor4.Java :
Bu sınıf Runnable Interface' inden implement edilmiştir. Runnable interface' inde bulunan run metodu override edilmiştir. Sınıfımız bu metodun içinde işlemlerini gerçekleştirmektedir. Sınıfımız katlardaki kuyrukları kontrol eder. Maksimum kapasite olan 10' u aşmayacak şekilde kuyruktaki müşterilerin talep ettikleri katlara ulaşabilmesini sağlar. Başlangıçta pasif haldedir. KontrolThread katlardaki kuyrukları kontrol edip kuyruklarda bekleyen toplam müşteri sayısı 20' yi geçtiğinde Asansör1, Asansör2 ve Asansör3 aktifse bu asansörü aktif hale getirir. Eğer Asansör5 pasif durumda ve bu asansör aktif durumdayken katlardaki kuyruklarda bekleyen toplam müşteri sayısı 10' un altına düşerse bu asansör KontrolThread tarafından pasif hale getirilir.

Asansor5.Java :
Bu sınıf Runnable Interface' inden implement edilmiştir. Runnable interface' inde bulunan run metodu override edilmiştir. Sınıfımız bu metodun içinde işlemlerini gerçekleştirmektedir. Sınıfımız katlardaki kuyrukları kontrol eder. Maksimum kapasite olan 10' u aşmayacak şekilde kuyruktaki müşterilerin talep ettikleri katlara ulaşabilmesini sağlar. Başlangıçta pasif haldedir. KontrolThread katlardaki kuyrukları kontrol edip kuyruklarda bekleyen toplam müşteri sayısı 20' yi geçtiğinde Asansör1, Asansör2, Asansör3 ve Asansör4 aktifse bu asansörü aktif hale getirir. Eğer bu asansör aktifken katlardaki kuyruklarda bekleyen toplam müşteri sayısı 10' un altına düşerse bu asansör KontrolThread tarafından pasif hale getirilir.

KontrolThread.Java :
Bu sınıf Runnable Interface' inden implement edilmiştir. Runnable interface' inde bulunan run metodu override edilmiştir. Sınıfımız bu metodun içinde işlemlerini gerçekleştirmektedir. Sınıfımız katlardaki kuyrukları kontrol eder. Tüm katlarda kuyrukta bekleyen toplam müşteri sayısı asansör kapasitesi olan 10'un iki katını aştığı yani 20' yi geçtiği durumda yeni asansörü aktif hale getirir. Tüm katlarda kuyrukta bekleyen toplam müşteri sayısı asansör kapasitesi olan 10 'un altına düştüğünde asansörlerden birini pasif hale getirir. Bu işlemleri common.Java sınıfındaki değişkenlere erişip onlar üzerinde değişiklik yaparak gerçekleştirir. Ama asansörlerden birisi her zaman çalışır vaziyette olacağı için bu işlem tek asansörün çalıştığı durumda geçerli değildir. Yani Asansör1 in çalışma durumuna asla müdahale edemez.

Project.Java :
Bu sınıf projenin işlemlerinin çalıştırıldığı main sınıfıdır. LoginThread.Java, ExitThread.Java, Asansor.Java, KontrolThread.Java sınıfları bu sınıfta start(); metoduyla çalıştırılmaktadır.

Musteri.Java :
Bu sınıfın içindeki constructor da random müşteri sayısı ve random kat sayısı bulunmaktadır. LoginThread.Java ve ExitThread.Java sınıfları bu

sınıfı kullanarak rastgele müşterileri ve rastgele gidilecek katları üretmektedir.

Katlar.Java :
Bu sınıf katlardaki müşteri sayılarını tutmaktadır. LoginThread.Java ve Asansor.Java sınıfları bu sınıfı kullanarak işlemlerini gerçekleştirmektedir.

common.Java :
Bu sınıf her bir asansörün aktiflik durumunu true ya da false değer olarak tutan değişkenlere sahiptir. KontrolThread sınıfı bu sınıftaki değişkenlere erişerek asansörlerin aktif ya da pasif durumda olmasını sağlar.

NewJFrame.Java :
Programın işleyişinin kullanıcıya basit ve anlaşılır bir şekilde gösterilmesini sağlayan framedir.

3.2. Algoritma

Program çalıştırıldığında ilk olarak LoginThread ve ExitThread çalışmaktadır. Daha sonra sürekli aktif durumda olacak olan Asansör1 çalışmaktadır. Daha sonra KontrolThread çalışıp tüm katlardaki kuyrukları kontrol ederek kuyrukta bekleyen toplam müşteri sayısı 20'yi her geçtiğinde öncelik sırası Asansör2, Asansör3, Asansör4 ve Asansör5 olmak üzere bu öncelik sırasına göre yeni bir asansörü daha aktif hale getiriyor. Kuyruktakilerin toplam sayısı 10 un altına her düştüğünde de en son aktif hale getirilen asansörü pasif hale getiriyor. Ama her zaman çalışan Asansör1' i asla pasif hale getiremez. Yukarıda da belirtildiği gibi asansörlerden birisi kuyruktaki müşteri sayısına bakılmaksızın her zaman aktif durumda kalıyor ve müşterileri taşımaya devam ediyor.

Programımız sürekli çalışan bir programdır. Yani kullanıcı programı sonlandırmadığı sürece sürekli çalışıp işlem yapmaya devam etmektedir.

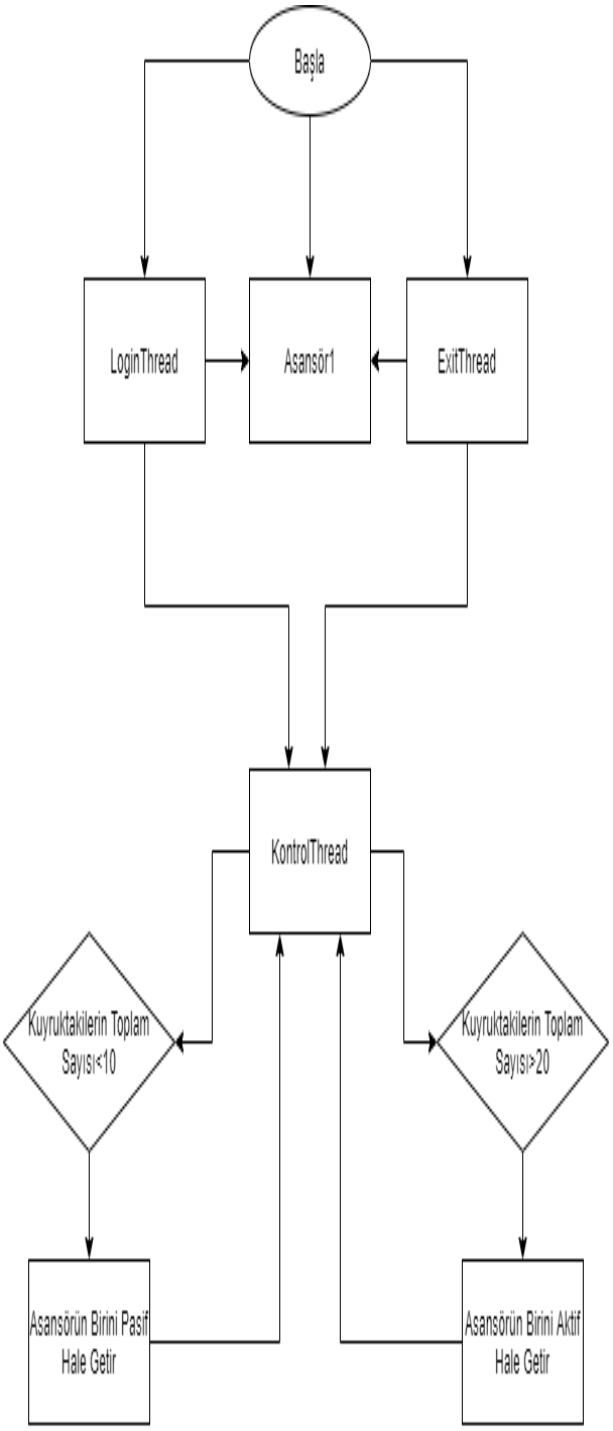
3.3. Ara Yüz

Asansörlerdeki Talep Yoğunluğunun Multithread ile Kontrolü Projesinde programın işleyişinin daha iyi takip edilmesi için ara yüz olarak swing kütüphanesini kullanarak bir JFrame oluşturduk. Bu ara yüz programın işleyişini kullanıcıya basit ve anlaşılır bir şekilde göstermektedir.

4. Sonuçlar

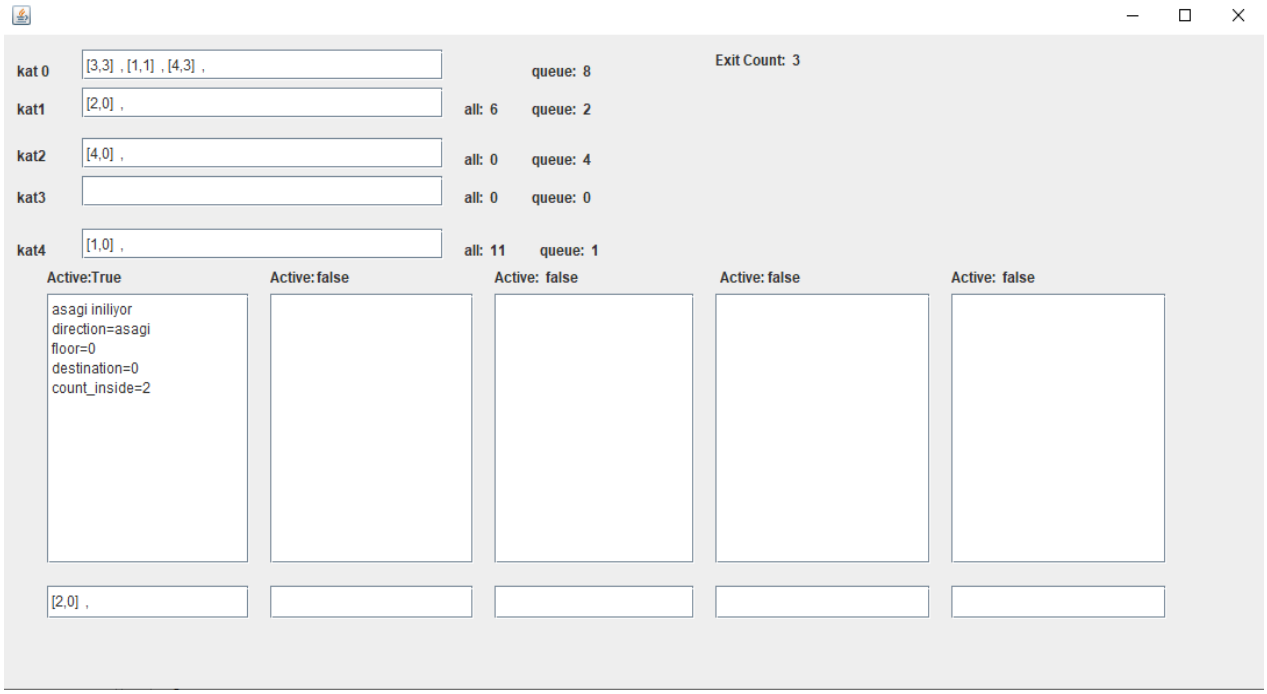
Proje kodlandıktan sonra yaptığımız çalışmalarda . Projemiz son hali ile başarılı bir şekilde çalışmakta ve bizden beklenen isterleri sağlamaktadır.

5. Akış Diyagramı

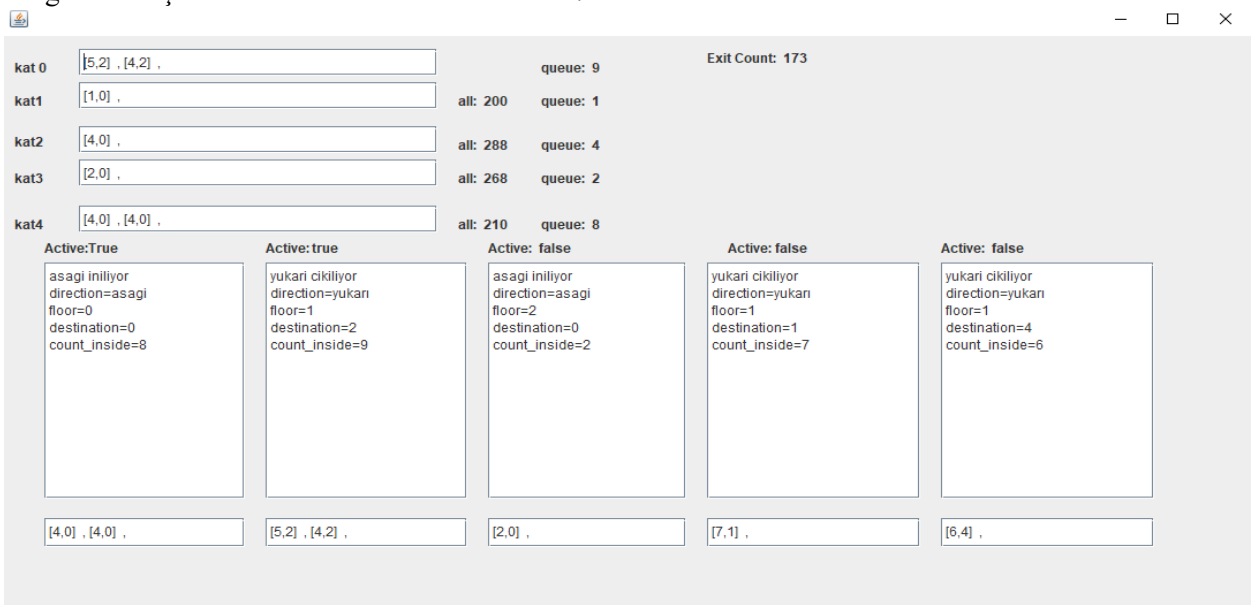


6. Ekran Çıktıları

Program ilk çalıştırıldığında asansörlerin durumu:



Programın başka bir anında asansörlerin durumu:



7. Kaynakça

1)
https://gelecegiyazanlar.turkcell.com.tr/konu/android/egitim/android-101/threadler
2)
https://www.udemy.com/share/101WziAkUfdV9bTXo=/
3)
https://www.yusufsezer.com.tr/java-thread/