Restoran Yönetim Sistemi Projesi Raporu

Fatma Nur Kurt
210202003
Kocaeli Universitesi Bilgisayar Mühendisliği İÖ
kurtfatmanur8@gmail.com

Esin Özdemir
210202086
Kocaeli Universitesi Bilgisayar Mühendisliği İÖ
esinozdemir2002@gmail.com

***Index Terms*—Java, Java Swing, GUI,PostgreSql, SQL, Sorgu, SFW, Class, Method, ArrayList, Layout .**

I. ÖZET

Bu rapor belgesi 2023-2024 Güz dönemi Yazılım Laboratuvarı - I 3. projeyi açıklamaya yönelik bilgiler içermektedir. Bu proje kapsamında öğrencilerin işletim sistemlerinin temel kavramlarını (thread yönetimi ve senkronizasyon mekanizmalarını) anlamaları ve bu kavramları bir simülasyon üzerinde uygulamaları hedeflenmektedir. Projede bir restoran yönetim sistemi gerçekleştirmeniz beklenmektedir. Bu proje bir restoranın günlük işleyişini simüle eden bir multithread uygulamasıdır. Buradaki amaç, gerçek zamanlı bir ortamda thread senkronizasyonu ve kaynak yönetimi konseptlerini uygulamaktır. Restoranda, müşteri grupları, garsonlar, aşçılar ve bir kasa elemanı bulunmaktadır. Her bir rol, farklı görevleri ve sorumlulukları olan ayrı bir thread olarak işlenecektir. Uygulama arayüzden çalıştırılacaktır.

Programlama Dili: Herhangi bir programlama dilini kullanarak Masaüstü simülasyon uygulaması yapılması hedeflenmiştir.

Sürecin aşağıdaki şekilde ilerlemesi istenmektedir. • Müşterilerin sıraya girmesi • Garsonların sipariş alması • Aşçıların siparişleri hazırlaması • Müşterilerin siparişleri ödemesi

Her müşteri, restorana giriş yaparken bir thread olarak oluşturulur. Her garson, müşterilere hizmet vermek için ayrı bir thread'de çalışır. Her aşçı, yemek hazırlama sürecini yönetmek için kendi thread'inde çalışır. Kasa işlemleri, ödeme alma ve hesap kapatma (masanın uygun duruma gelmesi) işlemleri için ayrı bir thread'de gerçekleştirilir.

Müşteri thread'leri arasında masa seçimi ve oturma sırası için senkronizasyon gerekir. Garson thread'leri, siparişleri alırken ve işlerken diğer garsonlarla koordineli çalışmalıdır. Aynı masaya birden fazla garsonun bakması önlenmelidir. Aşçı thread'leri, sınırlı sayıda ocakta yemek hazırlarken birbirleriyle uyum içinde olmalıdır. Aynı siparişi birden fazla aşçının hazırlaması engellenmelidir. Kasa thread'i, her seferinde yalnızca bir siparişin ödemesini işleyebilir.

Her masa dolu olduğunda yeni gelen müşteri bir bekleme listesine alınır. Boş masa olduğunda müşteri sırasına göre uygun müşteri masaya yerleştirilir. Proje kapsamında müşteriler iki farklı kategoride değerlendirilebilir: Normal müşteriler ve öncelikli müşteriler (65 yaş ve üzeri). Öncelikli müşteriler restorana geldiklerinde, normal müşterilerin önüne geçmelidir. Bu durum masalara oturma sırasında öncelikli müşteri avantajı sağlamalıdır. Boş masa varken garsonlar bekleme durumundadır. Masa doldukça garsonlar sırayla müşterilerden sipariş alır. Aynı anda gelen olursa rastgele sipariş alımı yapabilir. Her garson aynı anda sadece bir müşterinin siparişini alabilir. Aşçılar sipariş gelene kadar bekler. Sipariş geldikten sonra yemek yapmaya başlarlar. Yemek hazır oldukça sıradaki siparişleri hazırlamaya başlarlar. Sipariş alındığında, her bir aşçı aynı anda en fazla 2 yemek hazırlayabilir. Sipariş hazır olduktan sonra müşteriler yemeğe başlar. Müşteriler yemek yerken, kasa işlemleri için beklemelidir. Yemek biten masanın ödemesi kasa tarafından alınır. Kasa her seferinde sadece 1 siparişin ödemesini gerçekleştirebilir. Müşteri Bekleme Süresi: Yeni müşteri restorana geldiğinde tüm masalar doluysa 20 saniye süren bir bekleme süresi simüle edilir. 20 saniye sonra müşteri restorandan ayrılmaktadır. Sipariş Alma Süresi: Garsonların müşterilerden sipariş alması 2 saniye sürer. Yemek Hazırlama Süresi: Aşçılar sipariş aldıktan sonra yemek hazırlamaya başlarlar ve bu süreç 3 saniye sürer. Yemek Yeme Süresi: Müşterilerin yemek yeme süresi 3 saniye olarak ayarlanır. Ödeme İşlemi Süresi: Müşteriler yemeklerini bitirdikten sonra, ödeme işleminin gerçekleşmesi 1 saniye sürer. Tüm adımlar anlık olarak bir metin dosyasına yazdırılması istenmektedir.

Projede temel olarak problem-1 ve problem-2 olmak üzere 2 adet problem verilmiştir.

Problem 1: Restoranda belirli sayıda masa, garson, aşçı ve kasa bulunduğunda, bu kaynakların etkileşimini ve işleyişini nasıl simüle edilir? Uygulama çalıştırıldığında müşterilerin gelme sırası belirlenerek projenin özellikleri ve süreler dikkate alınarak simülasyon gerçekleştirilir. İşlem sırası geldiğinde kullanıcı onayıyla işlemler yürütülür. Simülasyon başlangıcında müşteri senaryosu belirlenir. Bu aşamada, zaman sınırlamalarına bağlı kalmaksızın müşterilerin öncelik düzeylerinin belirlenmesi hedeflenmektedir. Restoranda başlangıçta 6 masa, 3 garson, 2 aşçı ve 1 kasa bulunmaktadır. Aynı anda en fazla 10 müşteri gelebilir.

Problem 2: Dinamik bir müşteri akışında; hangi sayıda masa, garson ve aşçı çalıştırılırsa en fazla kazanç sağlanır? Sabit Akış Modeli:

Toplam süre, müşteri gelme aralığı uygulamada belirlenmelidir. Örnek: Her 5 saniyede 4 müşteri gelmektedir ve 1'i öncelikli olmaktadır. 3 dakikalık sürede gelecek müşteriye göre restoranın maksimum kapasitesi hesaplanır (masa, garson, aşçı). Belirlenen simülasyon adımı içerisinde maksimum

verimle kaynakların çalışması beklenmektedir. Maliyetler: Masa Maliyeti: Her masa için maliyet 1 birim olarak kabul edilir. Garson Maliyeti: Her garson için maliyet 1 birim olarak kabul edilir. Aşçı Maliyeti: Her aşçı için maliyet 1 birim olarak kabul edilir. Müşteri Kazancı: Her müşteri başına elde edilen kazanç 1 birim olarak hesaplanır. Bir problem için bulunan çözüm senaryolarını ve en iyi çözümü göstermeniz beklenmektedir.

II. YÖNTEM

Başlangıçta Restoran Yönetim Sistemi adlı bir JFrame oluşturduk. bu framin içine bir flowLayout panel oluşturarak bu panele iki tane button ekledik. Butonlar ”problem 1” ve ”problem 2” olmak üzere tıklanıldığında ilgili işlemleri gerçekleştirdik. FileWriter ve BufferedWriter kullanarak oluşturduğumuz metinDosyası.txt ye tüm adımları anlık olarak yazdırdık. İlk olarak problem 1 i ele aldığımızda problem 1 butonuna tıklanıldığında frami kapatıyoruz ve yeni frameler açıyoruz. Garson, kasa ve aşçı olmak üzere 3 adet frame oluşturuyoruz. bu framelere BorderLayout paneller ekliyoruz. 3 Frame de ekrana sığacak şekilde boyutlarını, konumlarını ve görünürlüklerini ayarlıyoruz. getStepCount() diye bir metot oluşturduk. Bu metotta kullanıcıdan adım sayısını kontrol ederek alıyoruz ve adım sayısını döndürerek stepCount adlı değişkenimize atıyoruz. stepCount eğer 0 dan büyükse işlemlerimize devam ediyoruz. Her adımdaki normal ve öncelikli müşterilerin sayısını tutmak üzere iki adet integer dizi oluşturuyoruz. Aynı zamanda oluşturduğumuz Waiter sınıfımızdan 3 adet nesne oluşturuyoruz ve onları waiters adlı waiter türündeki arraylistimize ekliyoruz. Chef sınıfımızdan 2 adet nesne türeterek chefs adlı Chef türündeki arraylistimize ekliyoruz. Böylelikle 3 garson ve 2 aşçı mızı oluşturduk. Bir for döngüsünde adım sayımız kadar döngü oluşturuyoruz. Bu döngüde getCustomerCount metodumuzu kullanarak müşteri türüne ve adım sayısına göre müşterilerileri oluşturduğumuz integer dizilerimize atıyoruz. Sonrasında her adımda gelen müşteri sayısının 10 dan fazla olup olmadığının kontrolünü gerçekleştiriyoruz. Sonrasında createCustomer metodu ile müşterileri Customer sınıfından türeterek oluşturuyoruz ve customers adlı Customer türündeki arraylistimize ekliyoruz. Sonuç olarak tüm müşterileri bu arraylistte tutuyoruz. Arraylistimiz eğer boş değil ise tüm senaryoyu döngü içerisinde yazdırıyoruz ve kullanıcıya bilgilendirme mesajı kullanarak iletiyoruz. startCustomerThreads metodumuza customers arraylistimizi parametre olarak gönderiyoruz ve burada arraylistteki tüm müşterilerin Threadlerini oluşturup başlatıyoruz. CustomersThread adlı arraylistimize Threadlerin kontrolünü kolay sağlayabilmek için oluşturulan tüm threadleri ekliyoruz. Masa sayımızı 6 olarak belirleyip maxTables adlı değişkenimize atıyoruz ve 6 adet Table sınıfından nesne türetiyoruz. Bu masalara ilk gelen 6 müşteriyi yerleştiriyoruz. Oluşturulan masaları da Table türündeki tables arraylistimizde tutuyoruz. simulateCustomersAtTables() metodumuzu çağırıyoruz. Bu metotta garsonun arayüzündeki tüm simülasyon işlemlerini gerçekleştiriyoruz. Burada ilk olarak boş masa sayısını ve müşteri sayısını karşılaştırıp en küçük olana göre bir döngü oluşturuyoruz. 3 adet oluşturduğumuz panellerimize eklemeler yapıyoruz. Garson panelimize Sipariş işlemlerinin bulunacağı bir textArea ekliyoruz. Her adımda bu textarea ya eklemeler yapıyoruz. Bir bekleme listesi oluşturuyoruz ve masalara oturanları bu bekleme listesinden siliyoruz geriye sadece bekleyen müşteriler kalıyor. Her masaya müşteri oturduğunda remainingTables adlı değişkenimizi azaltıyoruz. İlk 6 masamız dolduktan sonra bekleme listesindeki öncelikli müşterileri öne alarak ilk olarak masalar boşladıkça öncelikli müşterilerin oturmasını sağlıyoruz. Masaları simüle edebilmek için ilk olarak masa sayısı kadar dikdörtgen paneller oluşturuyoruz ve her panele border ve arkaplan rengi ekleyerek Fig. 1. gösterildiği gibi simüle ediyoruz. Waiters arraylistimizdeki her bir garson için

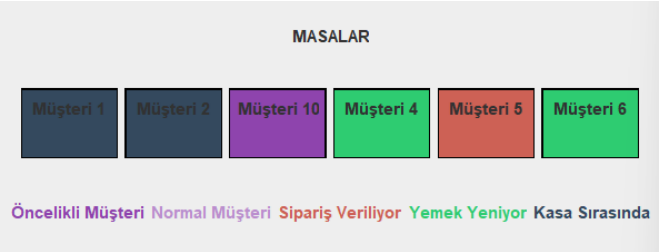


Fig. 1. Simülasyon Renkleri Ve Anlamları

Thread oluşturarak bu Threadleri start komutuyla başlatıyoruz. Threadleri kontrol altına alabilmek için WaiterThread adlı Thread türündeki arraylistimize threadlerimizi ekliyoruz. Aynı işlemleri aşçılar için de yapıyoruz ve Chef Threadlerimi start ile başlatıyoruz. Cashier sınıfımızdan bir nesne türeterek bu nesne ile bir Thread oluşturuyoruz ve bu Threadi start ile başlatıyoruz. Müşteri, Garson, Aşçı, Kasa sınıflarımızda Runnable sınıflarını implements ettik.

Masa Sınıfı:
Masa sınıfında masa numarası, masaya oturan müşteri, sipariş durumu ve her masaya özel bir kilit (lock) özelliklerini tutuyoruz. Sınıfın yapıcısında ReentrantLock() ile her masaya ait farklı kilitler üretiyoruz. Getter setter metodlarıyla işlemleri gerçekleştiriyoruz.

Müşteri Sınıfı:
Müşteri sınıfımızda müşteri numarası, öncelik durumu, oturduğu masa, yemek yeme durumu, adım durumu, restorana giriş zamanı ve kendine özel bir müşteri kilidi (Lock) özelliklerini tutuyoruz. Bu özelliklerden getter ve setter metodları lazım olanların getter ve setter metodlarını oluşturuyoruz. Run metodunu override ederek Threadlerimizin çalıştığı durumları kontrol ediyoruz. Bu metodun içinde synchronized kullanarak her müşterinin keni lock una göre senkronize bir şekilde çalışmasını sağlıyoruz. müşterinin restorana giriş zamanını bir değişkene atıyoruz ve müşteriyi bekleme listesine ekliyoruz. Bir while döngüsünde her 1 saniyede bir müşterinin bekleme listesinde olup olmadığını kontrol ediyoruz. Eğer 20 saniye geçmesine rağmen hala

bekleme listesinde ise o bekleme listesinden müşteriyi silip müşterinin threadini interrupt ederek durmasını sağlıyoruz. Ve her yaptığımız işlemi append kullanarak arayüzdeki ilgili textarea ya yazdırıyoruz. Eğer müşteri yemeğini yediyse müşteriyi ödeme listesine ekliyoruz. Son olarak müşteri sayılarını güncelliyoruz.

Fig. 2. Garson Arayüzü

Garson Sınıfı:
Garson sınıfımızda garson numarası, masa arraylisti özelliklerini tutuyoruz. Run metodunu override ederek Threadlerimizin çalıştığı durumları kontrol ediyoruz. Tüm masaları for döngüsünde dolaşarak tryTakeOrder metodumuza masaları parametre olarak veriyoruz ve çağırıyoruz. Burada table sınıfındaki o masaya ait klit mekanizmasını getter ile getiriyoruz ve trylock metodu ile garsonlar arası masa almada senkronizasyon sağlıyoruz. Eğer masa boş ve bekleme listesinde müşteri var ise bekleme listesindeki ilk müşteriyi masaya oturtuyoruz ve arayüzdeki gerekli işlemleri ayarlıyoruz. Eğer masa dolu ve sipariş alınmamış ise masadan sipariş alıyoruz. Bu sürecin simüle edilebilmesi için 2 sn olmak üzere sleep metodu ile bir bekletme gerçekleştiriyoruz. Garsonun aldığı siparişi 1 saniye içerisinde de aşçıya iletmesini yani sipariş listesine müşterinin numarasını eklemesini sağlıyoruz. Aynı zamanda aşçıların listeye bir ekleme yapıldığında haberdar olmasını notifyAll metodu ile sağlıyoruz. Masanın Order özelliğini de true olarak ayarlıyoruz. Son olarak garsonun masada işi biterse table a ait kilit mekanizmasını unlock metodu kullanarak serbest bırakıyoruz.

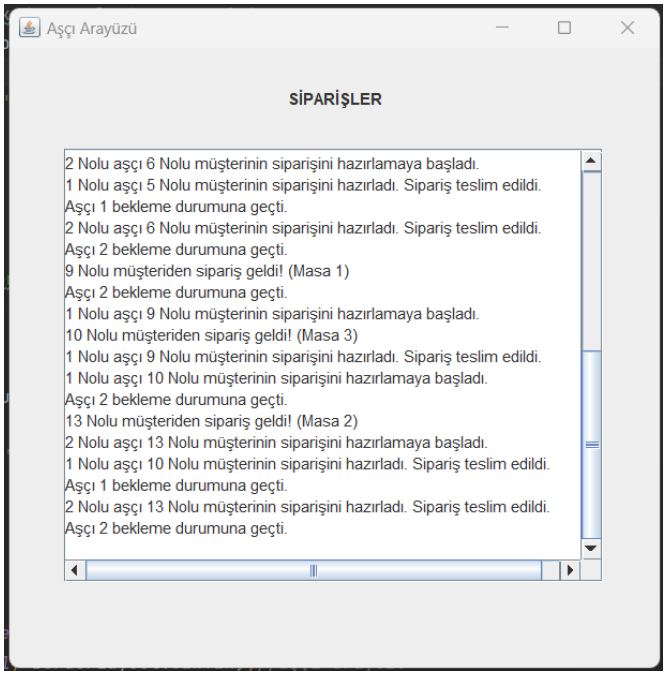


Fig. 3. Aşçı Arayüzü

Aşçı Sınıfı:
Aşçı sınıfımızda aşçı numarası ve her aşçıya özel bir kilit (lock) özelliklerini tutuyoruz. Run metodunu override ederek Threadlerimizin çalıştığı durumları kontrol ediyoruz. getOrder metodu ile aynı siparişi iki aşçının hazırlamasını önlemek için sipariş listesi üzerinde bir senkronizasyon sağladıktan sonra listedeki ilk siparişi döndürüyoruz ve prepareOrder motoduna parametre olarak gönderiyoruz. prepareOrder metodunda her aşçıya özel kilidi kullanarak lock metodu ile kitleme yapıyoruz. Sonrasında hazırlamaya başladığımız siparişi 3 saniye simüle etmek için sleep metoduyla bekletiyoruz. Sipariş hazırlama bittiğinde unlock metodu ile kilidi kaldırıyoruz. Eğer getOrder metodunda sipariş bulunmazsa waitForOrder metodu ile aşçıyı sipariş listesi üzerinde wait metodu ile bekleme durumuna alıyoruz. Böylelikle her sipariş geldiğinde aşçılar bekleme durumdan uyanıyor ve siparişleri hazırlıyorlar.

Kasiyer Sınıfı:
Kasiyer sınıfımızda tek bir kasamız olacağı için kasa numarası tutmadık. Run metodunu override ederek Threadin çalıştığı durumları kontrol ediyoruz. Mainde oluşturduğumuz ödeme listesi (kasa sırası) üzerinden kasamızı senkronize ediyoruz. Bir for döngüsünde ödeme listesindeki müşterilere ulaşıyoruz ve listedeki müşterilerin sırayla ödemelerini, 3 saniye simüle etme süresi kullanarak yani sleep metodu ile uyutarak gerçekleştiriyoruz. Bunları yaparken genel ödeme sayısını görmek içinde ödeme yapan müşteri sayısını ödeme yapıldıkça arttırıyoruz ve arayüzde gösteriyoruz. Müşteriyi masadan kaldırıyoruz. Müşteriyle ilgili olan diğer sınıflardaki

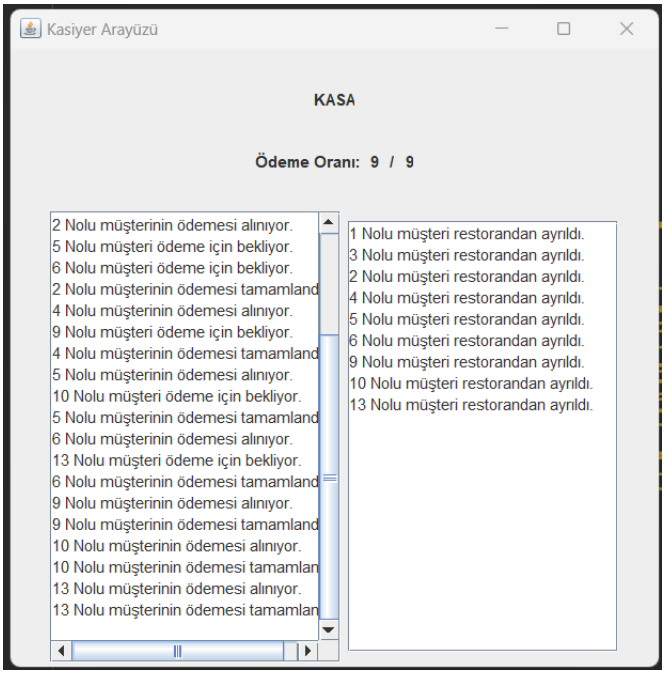


Fig. 4. Kasiyer Arayüzü

özellikleri de silerek müşteri threadini interrupt yaparak durduruyoruz. Boş masa sayısını artırarak güncelliyoruz. Son olarak simülasyonun bitip bitmediğini anlamak için masalarda müşterinin olup olmadığını kontrol ediyoruz. Tüm masalar boş ise simülasyonu bitiriyoruz ve tüm threadleri interrupt ile durduruyoruz.

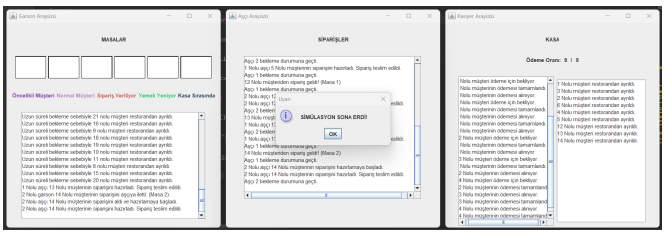


Fig. 5. Restoran Yönetim Sistemi Simülasyon Sonu

Problem 2: Bu problemde belirli süre içinde müşteri akışı hesaplanarak restoranın kaynakları belirlenerek maksimum müşteri sayısına hizmet vermek amacıyla kazanç hesaplanması istenmektedir. Öncelikle bir buton yardımıyla toplam süre, müşteri gelme aralığı (sn), 65 yaş altındaki müşteri sayısı (önceliksiz) ve 65 yaş üstündeki müşteri sayısı (öncelikli) JOptionPane.showInputDialog metodu ile kullanıcıdan bir metin girişi yapması istenir. Öncelikle toplam süre içinde kaç müşterinin geleceği hesaplanır.Normal müşteri ve öncelikli müşteriler toplanarak kaç müşteri geleceği bulunur.Toplam süre ile çarpılıp periyoda bölünür ve 1 eklenir (0 saniyede de müşteri geldiği için) bu sayede kaç toplam müşteri sayısı bulunur. İlk olarak kazancı toplam müşteri sayısına eşitliyoruz bunu sonradan kullanmak için. Garson, aşçı, müşteri, kasa, masa ve garson için ayrı ayrı sınıflarımızı oluşturduk. Bu sınıflardan maksimum kazancı hesaplayabilmek için nesneler oluşturulup ilgili ArrayList listesinin içine eklenir. Bu, oluşturulan nesneleri takip etmek için bir liste kullanılmasını sağlar. İç içe for kullanarak masa sayısı,garson sayısı ve aşçı sayısı kadar nesneler oluşturulup arrayliste eklenir ancak her döngüde masa,aşçı ve garsonun .clear() metodu ile liste içindeki tüm öğeleri temizlemek (silerek boşaltmak) için kullanılmıştır. Böylece her döngüde ilgili sayı kadar nesne oluşturulması sağlanmış olur.

Program çalıştırıldığı andan itibaren direkt müşteriler geleceği için time değişkeni ile kontrol sağlanarak döngüler ile öncelikli müşteri kadar nesne üretilip arayliste aktarırılır. Aynı şekilde girilen sayı kadar normal müşteriler de kendi listesine eklenir. Ek olarak toplam müşteriler listesine hem öncelikli hem de önceliksiz müşteriler eklenir. Bu sayede 0. saniyede gelecek müşteriler bir ilgili liste içinde tutulmuş olur. Masa,garson ve aşçı sayısı toplanır ve kazançtan çıkarılır. Bu bizim aslında yeni kazancımız olmuş olur. Döngü sayesinde toplam gelecek müşteri sayısını kontrol ederek eğer bekleme süresi 20 sn geçerse ya da müşteri masadan kalkıp hesabı öderse bu sayı 1 azaltılır ve totalCustomerCount = 0 olursa müşteri kalmamıştır döngü sona erer. Bu döngü içinde öncelikle masa sayısı kadar masaların boş mu dolu mu olduğu masa sınıfında isFull özelliği ile kontrol edilir. Eğer masa boşsa önceliklil müşteri sıfır değilse ve sıfırdan büyükse masa isFull ile meşgul konumuna gelir oturan seatedCustomers listesine ilgili öncelikli müşteri eklenir.Öncelikli müşterinin masa numarasına eşitlenir ki ödeme işleminden sonra ilgili masa müsait durumuna geçmelidir. Son olarak .remove(0) ile öncelikli müşteriler listesinden bir müşteriyi çıkarmak için işlem yapılır. Müşteri oturduktan sonra artık öncelikli müşteri olmadığı için listeden çıkarılması gereklidir.Öncelikli müşteriler oturduktan sonra aynı işlemi normal müşteriler için de yapılır. Bu sayede 0. saniye için en başta eklediğimiz listeleri kullanarak başta öncelikli müşterinin sonra da normal müşterilerin oturması sağlanmış olur.

Sıradaki aşama ise oturan müşterilerden sipariş almamız gerekiyor. Hem öncelikli hem de önceliksiz müşterilerin eklendiği seatedCustomers listesini kullanarak müşterilerden sipariş alınmalıdır. Müşteri sınıfında orderProcess özelliği başlangıçta sıfır olarak tanımlıdır. Eğer oturan müşteriler sipariş vermemişse ilgili adımda sipariş hazırlanma süreci başlatılır. Öncelikle garson sayısı kadar ve garson sınıfında isEmpty özelliği başlangıçta bir olarak tanımlıdır.Yani garson müsaitse oturan müşteri orderProcess özelliğini bir yapar ve müşterinin sipariş verdiğini ifade eder. Garson bu arada meşguldür ,sipariş alır ve garsonun hangi müşterinin siparişini aldığını belirtebilir. Yani bu aşamada bir müşterinin sipariş alındığını, garsonun meşgul olmadığını ve garsonun hangi müşterinin siparişini aldığını kaydeden bir işlemdir.

Sıradaki aşama oturan müşterilerin yemeğinin hazırlanma aşamasıdır. Oturan müşteri kadar döngü oluşturulur. Eğer oturan müşteri sipariş vermişse ve aşçı müsait ise aşçı sayısı kadar döngü oluşturulur. Bu döngüde oturan müşterilerin yemeği hazır değilse aşçının hangi müşterinin siparişini hazırlamaya başladığını, aşçının meşgul olduğunu, müşterinin

siparişi üzerine aşçının yemeği hazırlamaya başladığını yapar. Bu sayede aşçı müşterilerin siparişini hazırlar.
Sıradaki aşama oturan müşteri yemeğini yedi ve bitirmediyse ve ödeme süresi başlamamışsa döngü oluşturulur. Bu döngüde seatedCustomers listesindeki ilgili indeksine sahip müşterinin oturduğu masanın numarasını alarak, bu masanın tables listesindeki karşılık gelen masa öğesinin isFull özelliğini 0 olarak ayarlar. Bu, müşterinin oturduğu masanın artık boş olduğunu belirtir.fullTable listesinden, müşterinin oturduğu masayı çıkarır. Bu, dolu masaları takip eden bir başka listedir. seatedCustomers listesindeki ilgili indeksine sahip müşterinin odemeSuresi özelliğini bir artırır. Bu, müşterinin ödeme süresini temsil eder.Toplam müşteri sayısını bir azaltır. Bu, müşterinin ayrılması nedeniyle toplam müşteri sayısının güncellenmesidir. seatedCustomers listesindeki ilgili indeksine sahip müşterinin isFinished özelliğini true olarak ayarlar. Bu, müşterinin artık servis sürecini tamamladığını belirtir.break ile aynı anda birden fazla müşterinin ödeme yapmasını engellemek için kullanılır. Bir müşteri ödeme yaptığında döngüden çıkılır ve diğer müşterilere ödeme yapma şansı verilmez. Bu sayede bir müşterinin restorandan ayrılma durumu ve ayrıldıktan sonra ilgili durumların güncellenmesini sağlanır.
Müşterinin bekleme süresi 20 saniye ,garsonların müşterilerden sipariş alması 2 saniye, Aşçıların müşterilere yemek hazırlaması 3 saniye, Müşterilerin yemek yeme süresi 3 saniye ve müşteriler yemeklerini bitirdikten sonra ödeme işleminin gerçekleşmesi 1 saniye sürer.
Yemekleri hazır olan oturmuş müşterilerin yeme süreçlerini kontrol edilmesi gerekir. Eğer müşterinin yeme süresi 3'e ulaşırsa, müşterinin yeme işleminin tamamlandığını belirten isEat özelliği 1 olarak ayarlanır. seatedCustomers listesindeki her bir müşteri için bir döngü başlatır.Eğer seatedCustomers listesindeki ilgili indeksine sahip müşterinin isFoodReady özelliği 1 ise (yemeği hazır demektir). Müşterinin eatingTime özelliğini bir artırır. Bu, müşterinin yemeye başlama süresini temsil eder. Eğer müşterinin eatingTime özelliği 3'e eşitse müşterinin yeme işleminin tamamlandığını belirten isEat özelliği 1 olarak ayarlanır.
Aşçıların yemek hazırlama süreçlerini kontrol eder ve aşçıların hazırladığı yemekleri müşterilere hizmete sunar. Eğer bir şefin yemek hazırlama süresi 3'e ulaşırsa, aşçının durumu sıfırlanır ve hazırlanan yemek müşteriye hizmete sunulur. chefs listesindeki her bir aşçı için bir döngü başlatır. Eğer aşçının isEmpty1 özelliği false ise (yani aşçı meşgulse) Aşçının yemekHazirlamaSuresi1 özelliğini bir artırır. Eğer aşçının yemek hazırlama süresi 3'e eşitse aşçının yemek hazırlama süresini sıfırlar, isEmpty1 özelliğini true olarak ayarlar, yani şef artık boş durumdadır,aşçının hazırladığı yemeğin müşteriye servise hazır olduğunu belirten isFoodReady özelliğini 1 olarak ayarlar. Burada, customerNumber1 ile şefin hangi müşteri için yemek hazırladığını belirtir. Aşçının hangi müşteri için yemek hazırladığını gösteren customerNumber1 özelliğini sıfırlar veya boş bir değerle ayarlar. Bu, şefin bir müşteri için yemek hazırlama işlemini tamamladığını belirtir.
Garsonların belirli bir süre içinde sipariş almalarını kontrol eder ve sipariş aldıklarında ilgili durumları günceller. waiters listesindeki her bir garson için bir döngü başlatır.Eğer garsonun isEmpty özelliği 0 ise (yani garson meşgulse) Garsonun garsonSiparisAlmaZaman özelliğini bir artırır. Bu, garsonun sipariş almaya başlama süresini temsil eder. Eğer garsonun sipariş alma süresi 2'ye eşitse garsonun isEmpty özelliğini 1 olarak ayarlar, yani garson artık boş durumdadır. Garsonun sipariş aldığı müşterinin isOrdered özelliğini 1 olarak ayarlar. Bu, müşterinin sipariş verdiğini belirtir. Garsonun sipariş alma süresini sıfırlar. Garsonun hangi müşteriden sipariş aldığını belirten customerNumber özelliğini sıfırlar veya boş bir değerle ayarlar. Böylece her döngü için düzgün bir şekilde karşılaştırma yapabilir.
Belirli bir periyotta zaman geçmişse, toplam müşteri sayısı pozitifse ve toplam geçen süre belirli bir zaman değerine eşit veya büyükse öncelikli müşteriler ve normal müşteriler oluşturulur, bu müşterilerin listeleri birleştirilerek totalCustomers listesi elde edilir.
Normal müşterilerin bekleme sürelerini takip ederek, belirli bir bekleme süresini aştıklarında gerekli işlemleri gerçekleştiren bir kontrol mekanizmasını içerir. normalCustomers listesindeki her bir normal müşteri için bir döngü başlatır. Normal müşterinin bekleme süresini bir artırır. Bu, her geçen saniye için müşterinin bekleme süresini takip etmek için kullanılır. Eğer normal müşterinin bekleme süresi 20 saniyeyi aşıyorsa yield değişkenini azaltır. Bu normalde müşteri sayısından, masa garson ve aşçı sayılarının toplamlarından çıkarılmasına eşitlenmişti. Şimdi ise restoranı müşterinin bekleme süresi 20 saniyeyi aşıyorsa terk edenleri çıkarılıp kazancın bulunması sağlanır. Her kombinasyon için elde edilen verimlilik değerlerini karşılaştırarak en yüksek verimli kombinasyonu belirler. sayac değişkeni bir artırır, sayac değişkeni ise restoranı terk eden müşteri sayısına eşittir. Müşteri 20 saniye bekledikten sonra ayrıldığı için toplam müşteri sayısı azaltılır. totalCustomers ve normalCustomers listesinden normal müşteriyi çıkarır. Bu, normal müşteriyi 20 saniye bekledikten sonra ayrılması nedeniyle normal müşterinin çıkartılmasını sağlar. Aynı işlemler öncelikli müşteri için de uygulanır.
Mevcut yield değeri, şu ana kadar elde edilen en büyük verimden büyükse, bu değerleri güncelleyerek en yüksek verimi ve bu durumda kaç masa, garson, aşçı ve müşterinin ayrıldığını tutar.
Son olarak belirli bir durumun sona erdiği veya yeni bir dönemin başladığı bir noktada, ilgili listeleri temizleyerek yeni bir duruma hazırlık yapar.

Problem2Customer() Müşteri Sınıfı:
Parametresiz bir kurucu metottur. Sınıfın nesnesi oluşturulduğunda çağrılır. onceliksiz: Müşterinin önceliğini temsil eder. waitingTime: Müşterinin beklemiş olduğu süreyi temsil eder. isOrdered: Müşterinin sipariş verip vermediğini kontrol eden bir tamsayı değeri . 1 ise sipariş verilmiş, 0 ise verilmemiş. orderProcess: Siparişin hazırlanma sürecini temsil eder. asciYemegiHazirlamayaBasladiMi: Aşçının yemeği hazırlamaya başlayıp başlamadığını kontrol eder.

isFoodReady: Yemeğin hazır olup olmadığını kontrol eder. isEat: Müşterinin yemeğini yiyip yemediğini kontrol eden bir tamsayı değeridir. isFinished: Müşterinin işlemlerini tamamlayıp tamamlamadığını kontrol eden bir boolean değeri. eatingTime: Müşterinin yemeğini yeme süresini temsil eder. tableNumber: Müşterinin oturduğu masa numarasını temsil eder. odemeSuresi: Müşterinin ödeme süresini temsil eder. beklemeSuresiDolduMu: Bekleme süresinin dolup dolmadığını kontrol eden bir boolean değeridir.

Problem2Waiter Garson Sınıfı:
Bu sınıfın kurucu metodu parametresizdir. Sınıfın nesnesi oluşturulduğunda çağrılır. isEmpty: Garsonun meşgul olup olmadığını belirten bir tamsayı değeridir. Değer 1 ise boş, 0 ise meşgul demektir. siparisAldiMi: Garsonun sipariş alıp almadığını kontrol eder. 1 ise sipariş alındı, 0 ise alınmadı demektir. garsonSiparisAlmaZaman: Garsonun son sipariş alma zamanını temsil eder. customerNumber: Garsonun hizmet verdiği müşterinin numarasını temsil eden bir tamsayı değeri . -1 ise hizmet verilmediği anlamına gelir.

Problem2Chef Aşçı Sınıfı:
Bu sınıfın kurucu metodu parametresizdir. Sınıfın nesnesi oluşturulduğunda çağrılır. isEmpty1: Aşçının meşgul olup olmadığını belirten bir boolean değeridir. true ise boş, false ise meşgul demektir. yemekHazirlamaSuresi1: Aşçının yemek hazırlama süresini temsil eder. customerNumber1: Aşçının hizmet verdiği müşterinin numarasını temsil eder. -1 ise hizmet verilmediği anlamına gelir.

Problem2Table Masa Sınıfı:
Bu sınıfın kurucu metodu parametresizdir. Sınıfın nesnesi oluşturulduğunda çağrılır. isFull: Masa üzerinde müşteri olup olmadığını belirten bir tamsayı değeridir. 0 ise masa boş, 1 ise masa dolu demektir.

Problem2Cashier Kasa Sınıfı:
Bu sınıfın kurucu metodu parametresizdir. Sınıfın nesnesi oluşturulduğunda çağrılır. odemeSuresi: Ödeme süresini temsil eden bir tamsayı değeridir.

YALANCI KOD

•Başla(Problem2)
•Kullanıcıdan Toplam Süre, Müşteri Gelme Aralığı, Normal Müşteri Sayısı, ve Öncelikli Müşteri Sayısı al
•Toplam Müşteri Sayısını Hesapla: (Normal Müşteri Sayısı + Öncelikli Müşteri Sayısı) * ((Toplam Süre / Müşteri Gelme Aralığı) + 1)
•Başlangıçta Verim Değişkenini ve Zaman, Sayac Değerlerini sıfırla
•Masalar, Garsonlar ve Aşçılar listelerini temizle
•En Yüksek Verimi, Masa Sayısını, Garson Sayısını, Aşçı Sayısını ve Ayrılan Müşteri Sayısını saklamak için değişkenleri tanımla
•Toplam Müşteri Sayısını geçici bir değişkende sakla
•Verim = Toplam Müşteri Sayısı
•Döngü: Masa Sayısı arttıkça ,Garson Sayısı arttıkça,Aşçı Sayısı arttıkça
•Müşteri, Garson, Aşçı, Masa listelerini temizle
•Sayacı sıfırla
•Şefleri oluştur ve Şef listesine ekle (her şef 2 yemek yapabilir)
•Garsonları oluştur ve Garson listesine ekle
•Masaları oluştur ve Masa listesine ekle
•Zaman 1 den küçük ise:
Öncelikli Müşteri ve Normal Müşteri listelerini oluştur
Toplam Müşteri listesini güncelle
Verim için toplam müşteri sayısından masa, garson ve aşçı Sayısını çıkar
Döngü: Toplam müşteri sayısı sıfırlanana kadar
Döngü: Masalar üzerinde
Eğer Masa boşsa:
Eğer Öncelikli Müşteri varsa:
•Müşteriyi oturt ve Öncelikli Müşteri listesinden çıkar
Değilse Eğer Normal Müşteri varsa:
•Müşteriyi oturt ve Normal Müşteri listesinden çıkar
Değilse:
Döngüyü Sonlandır
•Garsonların sipariş almasını kontrol et
•Aşçıların yemek hazırlamasını kontrol et
•Müşterilerin yemek yemesini kontrol et
•Müşterilerin ödeme yapmasını kontrol et
•Zamanı bir arttır
•Toplam Müşteri Sayısı 0 olana kadar devam et
•Döngü: Öncelikli Müşteri Sayısı kadar
Müşteri Bekleme Süresini Artır
•Eğer Bekleme Süresi 20'yi Geçerse:
verim Düşür
Öncelikli Müşteriyi Ayrılan Müşterilere Ekle
Toplam Müşteri Sayısını Azalt
Toplam Müşteriler Listesinden
Öncelikli Müşteriyi Çıkar
•Döngü: Normal Müşteri Sayısı kadar
Müşteri Bekleme Süresini Artır
•Eğer Bekleme Süresi 20'yi Geçerse:
Verim Düşür
Ayrılan Müşteri Sayacını Artır
Normal Müşteriyi Ayrılan Müşterilere Ekle
Toplam Müşteri Sayısını Azalt
Toplam Müşteriler Listesinden Normal Müşteriyi Çıkar
•Verim = Toplam Müşteri Sayısı-masa sayısı - garson sayısı - aşçı sayısı - ayrılan müşteri sayısı
•Eğer Verim, En Yüksek Verimden büyükse: Verim sayısını güncelle
•Öncelikli ve Normal Müşteri listelerini temizle
•Diğer listeleri temizle
•Aşçı Sayısı Garson Sayısı ve Masa Sayısı döngüsü sonu
•Sonuçları ekrana yazdır
•Bitir

SONUC

Bu proje, Restoran Yönetim Sistemi adını taşıyan bir Java uygulamasıdır ve bir restoranın günlük işleyişini başarıyla simüle etmektedir. Müşteriler, garsonlar, aşçılar, kasiyerler ve masalar gibi temel unsurlar, Java'nın güçlü Thread mekanizması kullanılarak eş zamanlı bir şekilde modellenmiştir. Her bir müşteri, garson ve aşçının ayrı bir Thread içinde çalışması, restoranın dinamik ve paralel bir yapıda sorunsuz bir şekilde işlemesini sağlar. Bu yaklaşım, gerçek bir restoran senaryosunu taklit etmek için müşterilerin sipariş vermesi, garsonların hizmet sunması ve aşçıların yemek hazırlaması gibi işlemlerin aynı anda ve düzenli bir şekilde gerçekleşmesini mümkün kılar. Ayrıca, uygulama metinDosyasi.txt adlı dosyaya anlık olarak senaryo adımlarını kaydederek, kullanıcının senaryonun gelişimini takip etmesine olanak tanır. Bu proje, Java dilinde Thread kullanımı, GUI tasarımı ve dosya işlemleri konularında pratiğe yönelik bir örneği içermesi bakımından öğretici bir nitelik taşır.

Akış diyagramı rapor klasörüne .jpg olarak, UML diyagramı projeye .png olarak yüklenmiştir.

KATKILAR

Projenin Problem 1 kısmını Fatma Nur Kurt yapmıştır. Problem 2 ve rapor yazım kısmını Esin Özdemir ve Fatma Nur Kurt ortak yapmıştır.

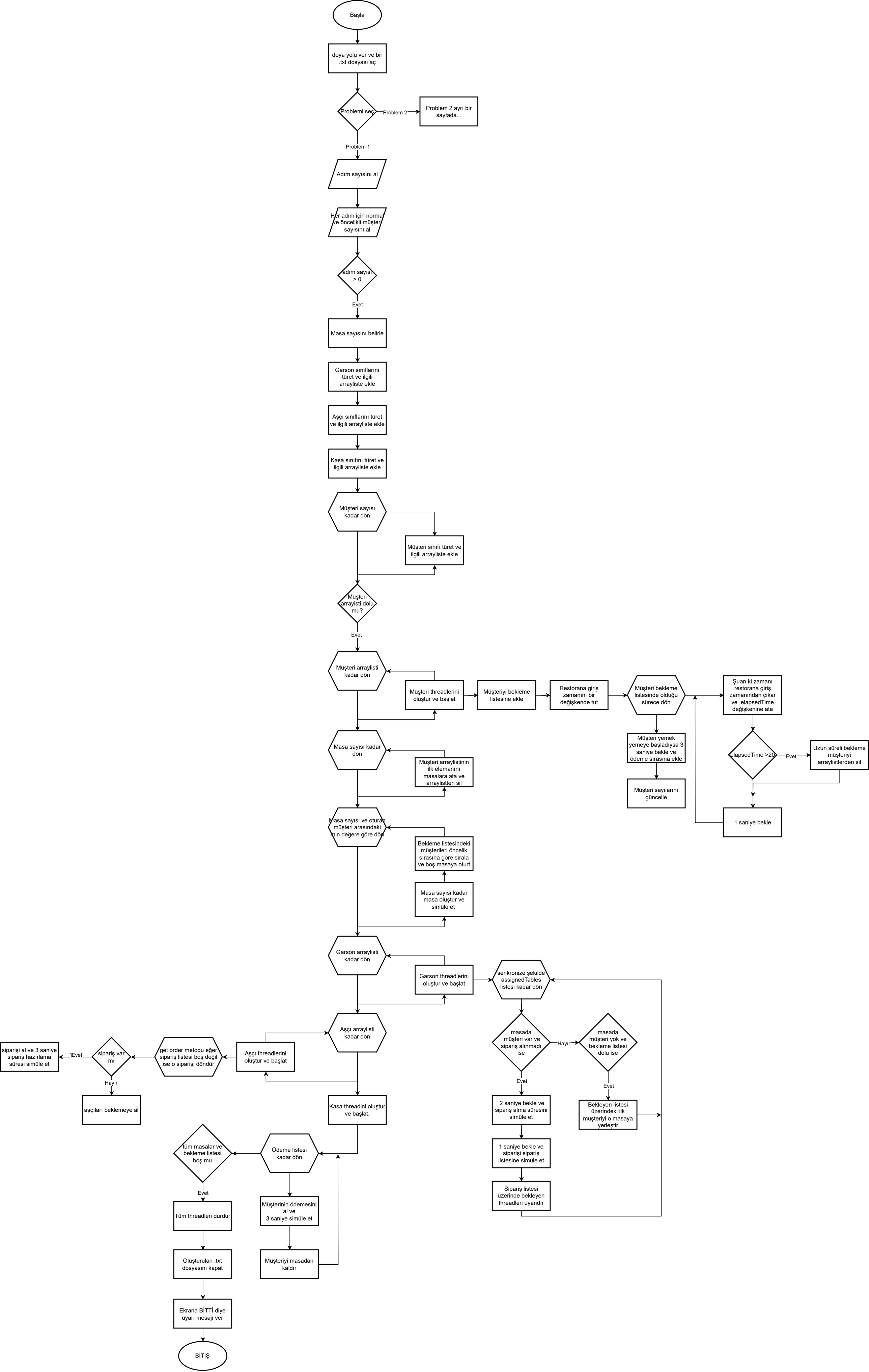
GELIŞTIRME ORTAMI

Projeyi Java dilinde, intellij IDEA 'da geliştirdik.
Kullandığımız kütüphaneler:

- javax.swing.*;
- javax.swing.border.EmptyBorder;
- java.awt.*;
- java.awt.event.ActionEvent;
- java.awt.event.ActionListener;
- java.io.BufferedWriter;
- java.io.FileWriter;
- java.io.IOException;
- java.util.ArrayList;
- java.util.concurrent.TimeUnit;
- java.util.concurrent.locks.Lock;
- java.util.concurrent.locks.ReentrantLock;

KAYNAKÇA

- https://www.youtube.com/
- https://github.com/
- https://htmlcolorcodes.com/
- https://chat.openai.com/
- https://stackoverflow.com/


Başla
doya yolu ver ve bir .txt dosyası aç
Problemi seç
Problem 2
Problem 2 ayrı bir sayfada...
Problem 1
Adım sayısını al
Her adım için normal ve öncelikli müşteri sayısını al
adım sayısı > 0
Evet
Masa sayısını belirle
Garson sınıflarını türet ve ilgili arrayliste ekle
Aşçı sınıflarını türet ve ilgili arrayliste ekle
Kasa sınıfını türet ve ilgili arrayliste ekle
Müşteri sayısı kadar dön
Müşteri sınıfı türet ve ilgili arrayliste ekle
Müşteri arrayisti dolu mu?
Evet
Müşteri arraylisti kadar dön
Müşteri threadlerini oluştur ve başlat
Müşteriyi bekleme listesine ekle
Restorana giriş zamanını bir değişkende tut
Müşteri bekleme listesinde olduğu sürece dön
Müşteri yemek yemeye başladıysa 3 saniye bekle ve ödeme sırasına ekle
Müşteri sayılarını güncelle
Şuan ki zamanı restorana giriş zamanından çıkar ve elapsedTime değişkenine ata
elapsedTime >20
Evet
Uzun süreli bekleme müşteriyi arraylistlerden sil
1 saniye bekle
Masa sayısı kadar dön
Müşteri arraylistinin ilk elemanını masalara ata ve arraylistten sil
Masa sayısı ve oturan müşteri arasındaki min değere göre dön
Bekleme listesindeki müşterileri öncelik sırasına göre sırala ve boş masaya oturt
Masa sayısı kadar masa oluştur ve simüle et
Garson arraylisti kadar dön
Garson threadlerini oluştur ve başlat
senkronize şekilde assignedTables listesi kadar dön
masada müşteri var ve sipariş alınmadı ise
Hayır
masada müşteri yok ve bekleme listesi dolu ise
Evet
2 saniye bekle ve sipariş alma süresini simüle et
1 saniye bekle ve siparişi sipariş listesine simüle et
Sipariş listesi üzerinde bekleyen threadleri uyandır
Bekleyen listesi üzerindeki ilk müşteriyi o masaya yerleştir
Aşçı arraylisti kadar dön
Aşçı threadlerini oluştur ve başlat
get order metodu eğer sipariş listesi boş değil ise o siparişi döndür
sipariş var mı
Evet
siparişi al ve 3 saniye sipariş hazırlama süresi simüle et
Hayır
aşçıları beklemeye al
Kasa threadini oluştur ve başlat.
Ödeme listesi kadar dön
Müşterinin ödemesini al ve 3 saniye simüle et
Müşteriyi masadan kaldır
tüm masalar ve bekleme listesi boş mu
Evet
Tüm threadleri durdur
Oluşturulan .txt dosyasını kapat
Ekrana BİTTİ diye uyarı mesajı ver
BİTİŞ


2. Problem
Sonuç Butonu
Toplam süre
Periyot
Öncelikli Müşteri Sayısı
Önceliksiz Müşteri Sayısı
Masa Garson ve Aşçı için nesne oluşturup arrayliste aktarmak
Öncelikli müşteri ve öncelikssiz müşteriler için nesne oluşturup arrayliste aktarmak
Önce öncelikli daha sonra normal müşterileri boş masa bulup oturmasını sağlamak
Oturan müşterilerin siparişlerini almasını sağlamak
Siparişleri alınan müşterilerin aşçının yemeklerini hazırlaması
Yemeklerini yiyen müşterilerin ödeme işlemine geçmesi
Eğer 20 saniyeyi geçerse bekleme süresi restorandan ayrıl
En büyük kazancı bul (Müşteri sayısı- masa - garson - aşçı -ayrılan müşteri)
Bitir