ÇANAKKALE ONSEKİZ MART ÜNİVERSİTESİ
BİLGİSAYAR MÜHENDİSLİĞİ

Nesneye Dayalı Analiz ve Tasarım Dersi

Proje Konusu

Kargo Takip Sistemi

Proje Grubu 48

130401058

Dilek Üzülmez

120401062

Hatice Bilge Göktaş

DANIŞMAN

Yrd. Doç. Dr. Ali Murat Tiryaki

ÇANAKKALE, 2016

İÇİNDEKİLER

KARGO TAKİP SİSTEMİ

Vizyon

Kargo Takip Sistemi kullanıcılarından biri olan müşteri, masaüstü araçlarını kullanarak kargolamak istedikleri ürünlerinin zarar görmeden ve en uygun fiyata taşınmasını sağlamak amacıyla tasarlanmıştır.

Sistem üzerinde yetkilendirilmiş bir kullanıcı bulunacaktır. Sistemin ana aktörü "Sistem Yöneticisi" olacaktır. "Sistem Yöneticisi" sistem üzerinde gerekli işlemleri gerçekleştirebilecek, sistemdeki kullanıcıların sistemi kurallara uygun olarak kullanıp kullanmadıklarını denetleyip kuralları ihlal eden kullanıcıları sistemden çıkararak sistemdeki kullanıcıların madur olmalarına engel olabilecek, sistemin bakım ve onarımını yaparak sistemin sorunsuz şekilde çalışmasını sağlayacaktır. Sistemin tüm sorumluluğu "Sistem Yöneticisi"ne aittir.

Taşıyıcılar sisteme üye olmak için öncelikle "Sistem Yöneticisi"nin şart koştuğu bilgileri eksiksiz olarak sisteme girmesi gerekmektedir. Üye olduktan sonra "Sistem Yöneticisi"nin izin verdiği bilgilerini güncelleyebilir ve üyeliklerini puan toplayarak geliştirebilir ve ayrıcalıklara sahip olabilirler. Taşıyıcılar güvenirlik puanlarına göre gruplandırılırlar. Güvenirlik puanı çok olan taşıyıcılar altın üye olurlar. Altın üye olan taşıyıcılar 500 kg üzerindeki ürünler veya miktarı küçük fakat piyasa ederi büyük olan önemli ürünlerde müşterilere öncelikli tercih olarak sunulacaklardır. Geri kalan taşıyıcılar da normal taşıyıcı olarak sistemi kullanırlar. Altın üyelerimizden kargo başına %8lik sistem ücreti alınacaktır. Normal kullanıcılar ise kargo başına %10luk bir sistem ücretine tabi tutulacaktır.

Müşteriler kurumsal veya bireysel olarak sisteme üyelik yaparlar. Kurumsal olan müşteriler kargo ücretlendirmelerinde %10 indirim hakkına sahip olacaklardır. Üye olduktan sonra müşteri ürünü için maksimum verebileceği kargo fiyatını belirterek yeni bir ürün girişinde bulunabilir. Ürünü girdikten sonra ilgili taşıyıcılar ürün için açık eksiltmeye katılarak en uygun fiyat veren 3 taşıyıcıdan uygun gördüğünü seçebilir ya da hiçbirine vermeyebilir. Aynı zamanda daha önce göndermiş olduğu ürünleri de görebilir. Müşteri, taşıyıcının önceden belirttiği kargo alım ücreti üzerinden %20lik depozito ücretini, anlaşma yaptığı taşıyıcının hesabına aktarır. (Kargo gönderiminden vazgeçse bile bu ücret yatırılmak zorundadır.)

Ayriyeten anlaşma yapan müşteriler ile taşıyıcılar birbirleri ile sistem üzerinden dolaylı olarak haberleşebileceklerdir.

Sistem, gönderilecek ürünün türüne ve müşterinin verdiği fiyata göre gereksinimleri karşılayacaktır.

Actor-Goal Model

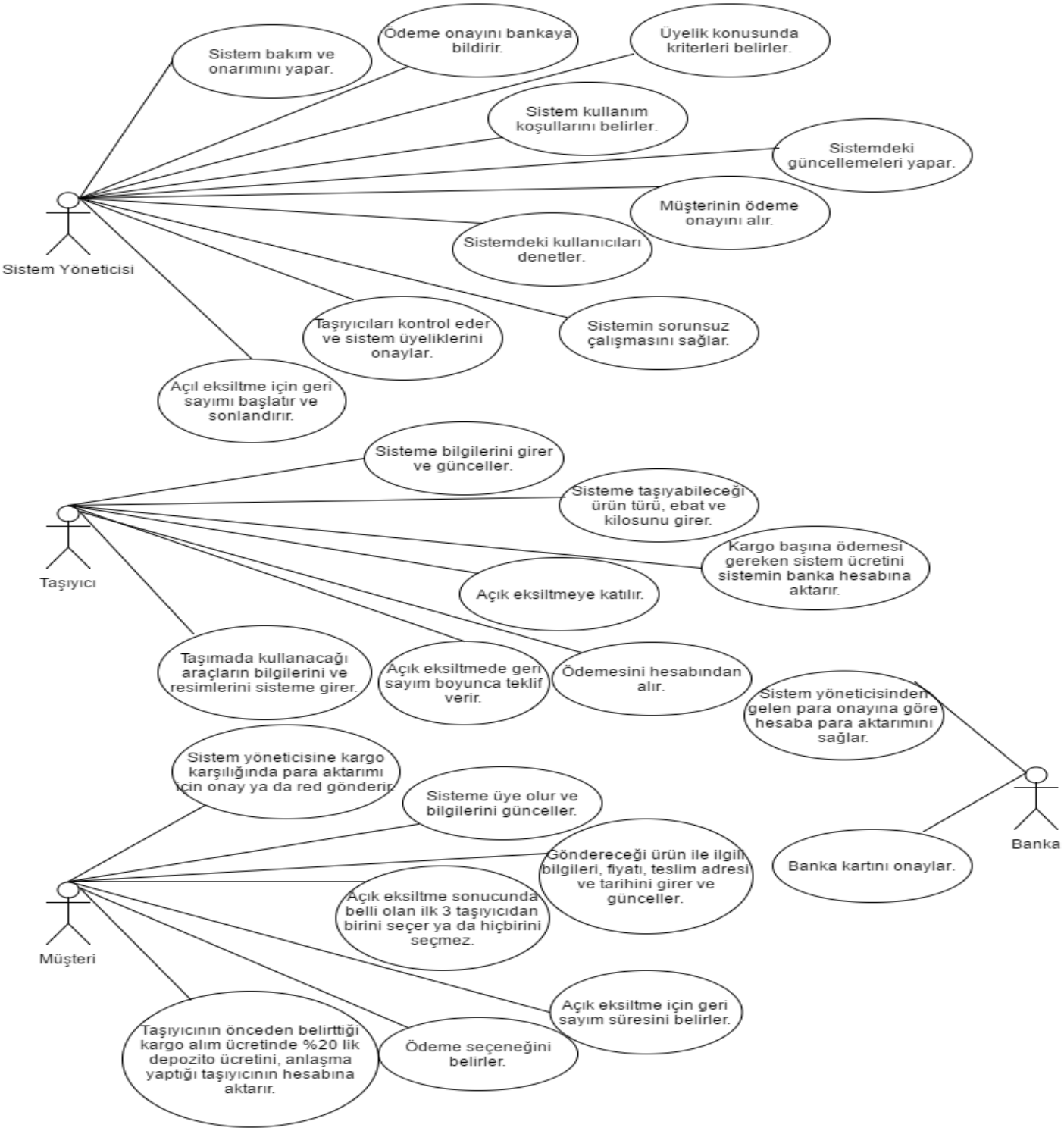
Actor	Goal
Sistem Yöneticisi	• Üyelik konusunda kriterleri belirler. • Sistem kullanım koşullarını belirler. • Sistemin onarım ve bakımını yapar. • Sistemdeki kullanıcıları denetler. • Sistemin sorunsuz çalışmasını sağlar. • Taşıyıcıların sisteme üyeliklerini kontrol eder ve uygunluğuna göre onaylar. • Sistemde güncellemeleri yapar. • Müşterinin ödeme onayını alır. • Ödeme onayını bankaya bildirir. • Geri sayım başlatır ve sonlandırır.
Taşıyıcı	• Sisteme bilgilerini girer ve günceller. • Taşıyabileceği ürün türü, ağırlığı ve ebatlarını girer. • Taşımada kullanacağı araçların bilgilerini ve resimlerini sisteme girer. • Açık eksiltmeye katılır. • Ödemeyi hesabından alır. • Kargo başına ödemesi gereken sistem ücretini, sistemin banka hesabına aktarır. • Açık eksiltmede geri sayım boyunca teklif verirler.

Actor Goal

Actor	Goal
Müşteri	• Müşteri sisteme üye olur ve bilgilerini günceller. • Göndereceği ürün bilgilerini, fiyatı, ulaşmasını istediği tarih gibi bilgileri belirtebilir, güncelleyebilir, silebilir. • Açık eksiltme sonucunda gelen 3 taşıyıcıdan birini seçebilir veya hiç birini seçmez. • Ödeme seçeneğini belirler(havale/eft ve kartla). • Sistem yöneticisine kargo karşılığında para aktarımı için onay yada red gönderebilir. • Taşıyıcının önceden belirttiği kargo alım ücreti üzerinden %20lik depozito ücretini, anlaşma yaptığı taşıyıcının hesabına aktarır. (Kargo gönderiminden vazgeçse bile bu ücret yatırılmak zorundadır.) • Açık eksiltme için geri sayım süresini belirler.

Use Case Model

Actorler	Kargo Takip Sİstemi	Dış Actorler



BUSSINESS RULES (TİCARİ KURALLAR)

ID	RULE	CHANGEBILITY	SOURCE
Rule 1	Ödemek için gerekli kredi kartı bilgileri bulunmalıdır.	Elektronik imza kullanılarak belgeleme yapılacaktır. 5 yıl içinde dijital kod imzasına geçilmelidir.	Banka politikası
Rule 2	Vergi Kuralları. Satışa verginin eklenmesi gerekiyor. Geçerli ayrıntılar için hükümetin tüzüğüne bakılmalıdır.	Vergi kuralları yıllık olarak değişir.	Vergi Kanunları
Rule 3	Kurumsal müşterilere hesaplanan kargo fiyatı üzerinden %10 indirim uygulanacaktır.	Yüksek.	Kullanıcı politikası
Rule 4	Belirtilen sürede kargolar, istenilen adrese iletilmelidir.	Yok.	Tüketici hakları
Rule 5	Kargo işlemi iptal edilse bile taşıyıcıya depozito ödenmelidir.	Yok.	Tüketici hakları
Rule 6	Altın üye olan taşıyıcılarımızdan kargo başına %8lik, normal taşıyıcılardan ise %10luk bir sistem ücreti alınacaktır.	Yüksek.	Şirket kazanç politikası

GLOSSARY

Sistem Yöneticisi : Sistem üzerinde gerekli işlemleri gerçekleştirebilecek, sistemdeki kullanıcıların sistemi kurallara uygun olarak kullanıp kullanmadıklarını denetleyip kuralları ihlal eden kullanıcıları sistemden çıkararak sistemdeki kullanıcıların madur olmalarına engel olabilecek, sistemin bakım ve onarımını yaparak sistemin sorunsuz şekilde çalışmasını sağlayacaktır.

Müşteri : Kargo gönderi talebinde bulunarak sistemimize üye olmuş kullanıcılardır.

Taşıyıcı : Kargo taşımak için açık eksiltmeye katılan ve müşteri tarafından bizzat seçilen kargo getir-götür işlemi yapacak kişidir.

Açık Eksiltme : Müşteri ve taşıyıcı arasında gerçekleşen, kargo için müşterinin verebileceği maksimum fiyat üzerinden eksiltmeye başlanılan ve ilgili kargo taşıyıcılarının bu fiyat üzerinden eksilterek en uygun fiyatı verme işlemidir.

Kargo : Müşterinin bir yerden bir yere taşıyıcı aracılığıyla taşınmasını istediği herhangi bir pakettir.

E-mail : Sisteme dahil olan kullanıcıların unique olarak kabul edilen kullanıcı adlarıdır.

Depozito : Müşteri ve taşıyıcının madur olmaması için sistem tarafından daha sonra ilgili tarafa verilmek üzere alınan bedeldir.

USE CASE 1

Kargonun Sisteme Eklenmesi

Scope : Kargo Ekleme Sistemi

Level : User goal

Primary Actor: Müşteri

Stakeholders and Interests:

Müşteri : Kargonun sorunsuz bir şekilde ve zamanında, en uygun fiyata istediği yere ulaştırılmasını ister. Güvenirlik puanı yüksek olan taşıyıcıyı tercih eder veya etmez.

Precondition : Müşteri sisteme kayıtlı ve giriş yapmış olmalı

Postcondition: Müşteri sisteme yeni bir kargo eklemiş olmalı

Main Success Scenario

1. Müşteri kargo ekleme isteğinde bulunur.
2. Sistem ürün listesini görüntüler.
3. Müşteri ürün seçer, kargoya ekler.
4. Sistem eklenen ürün bilgisini ve kargo bilgi formunu görüntüler.
5. Müşteri kargo bilgilerini ve açık eksiltme bilgilerini girer.
6. Sistem kargo bilgilerini ve açık eksiltme bilgilerini görüntüler.
7. Müşteri kargo ekleme işlemini sonlandırma isteğinde bulunur.
8. Sistem kargoyu kaydeder ve açık eksiltmeyi başlatır.

Extentions

*a.Sistem herhangi bir adımda çöker ise;

1.Müşteri sistemi yeniden başlatır ve sisteme yeniden giriş yapar.

3.a-Müşteri bilgileri eksik girer ise;

1-)Sistem, müşteriye “bilgilerinizi eksiksiz giriniz” uyarı mesajını verir.

2-)Sistem müşteriyi eksik bilgiler içeren sayfaya geri döndürür.

USE CASE 2

Taşıyıcının Teklif Vermesi

Scope : Açık Eksiltme Sistemi

Level : User goal

Primary Actor : Taşıyıcı

Stakeholders and Interests:

Taşıyıcı : Başarılı bir kargo ulaştırması yaptıktan sonra en yüksek güvenilirlik puanını almayı hedefler. Taşıyabileceği kargo ürünü en yüksek kazanç ile müşteriye en uygun fiyatı vererek taşımayı ister.

Precondition : Taşıyıcı sisteme kayıtlı ve giriş yapmış olmalı

Postcondition: Taşıyıcı açık eksiltmeye katılmış olmalı

Main Success Scenario

1.Taşıyıcı teklif verme isteğinde bulunur.

2.Sistem taşıyıcıya uygun olan açık eksiltmeleri listeler.

3. Taşıyıcı ilgilendiği açık eksiltmelerden birini seçer.

4. Sistem taşıyıcının seçtiği açık eksiltmeleye ait teklifleri görüntüler.

5. Taşıyıcı teklif fiyatını girer.

6. Sistem açık eksiltmeye ait teklif bilgilerini görüntüler.

7.Taşıyıcı teklif sonlandırma isteğinde bulunur.

8.Sistem teklif kaydedildi bilgisi görüntüler.

Extentions

*a.Sistem herhangi bir adımda çöker ise,

1-)Taşıyıcı sistemi yeniden başlatır ve sisteme yeniden giriş yapar.

USE CASE 3

Müşterinin Açık Eksiltmeyi Değerlendirmesi

Scope : Taşıyıcı Seçme Sistemi

Level : User goal

Primary Actor : Müşteri

Stakeholders and Interests :

Müşteri : Kargonun sorunsuz bir şekilde ve zamanında, en uygun fiyata istediği yere ulaştırılmasını ister. Güvenirlik puanı yüksek olan taşıyıcıyı tercih eder veya etmez.

Precondition : Müşteri sisteme kayıtlı olmalı ve en az bir açık eksiltmesi bulunmalı

Postcondition: Müşteri kargosu için bir taşıyıcı seçmiş olmalı

Main Success Scenario

1.Müşteri, taşıyıcı seçme isteğinde bulunur.

2. Sistem, müşteriye ait açık eksiltmeleri sunar.

3. Müşteri, değerlendirme yapacağı açık eksiltmeyi seçer.

4.Sistem, en uygun fiyat veren 3 teklifi ve taşıyıcı bilgilerini listeler.

5.Müşteri, tekliflerden birini seçer.

6.Sistem, ilgili taşıyıcıyı bilgilendirir ve seçildi bilgisi görüntüler.

7.Müşteri, taşıyıcı secme işlemini sonladırma isteğinde bulunur.

8.Sistem, taşınacaklar listesinden bu kargo kaydını kaldırır ve kargo listesini günceller.

Extentions

*a.Sistem herhangi bir adımda çöker ise;

1-)Müşteri sistemi yeniden başlatır ve sisteme yeniden giriş yapar.

5.a- Müşteri, en uygun fiyat veren 3 taşıyıcıdan birini seçmez ise;

1-)Sistem, müşteriyi kargo bilgilendirme sayfasına yönlendirir.

2-)Müşteri hizmet tercihlerini, verebileceği maksimum kargo fiyatını günceller.

3-)Açık Eksiltme Sistemi Use Case'i bir daha uygulanır.

SUPPLEMENTARY SPECIFICATION(EK GEREKSİNİMLER)

Functional

- Sistemin hafta da bir backup'ı alınır.
- Her kullanıcının kendine ait kullanıcı adı olmalıdır.(unique) Ve bunlar birbirinden farklı olmalıdır.
- Sisteme sadece kayıt olan kullanıcılar girebilecektir.(Kullanıcı adı ve parola ile)
- Tüm kargo işlemleri müşteri ve taşıyıcı onayına bağlıdır.
- Bütün sistemsel hatalar, kayıt altına alınır.
- Sistem yöneticisinin kriterleri dışındaki kullanıcılar sistemden men edilecektir.
- Sistemin çökmesi durumunda, acil durum yapılandırılması olmalıdır.

Usability

- Sistemimizi daha önce herhangi bir sosyal medya sayfasına üye olup kullanmış, bir bilgisayar kullanıcısı rahatlıkla kullanılabilecektir.
- Kullanıcı adı ve parola alınması yeterlidir.
- Dil desteği eklenmelidir.(Özellikle yabancı kullanıcılar için)

Reliability

- Sisteme kayıtlı müşteri bilgileri dışarıya kapalıdır.
- Sistemde herhangi müşteri hakkında bir bilgi hatası durumunda, işlem geriye alınabilecektir ve bu işlem baştan gerçekleştirilecektir.
- Para ödemeleri en güvenli sistemler ile sağlanacaktır.

Performance

- Müşterinin login olduktan sonra oturumu kullanmak için herhangi bir süre kısıtlaması yoktur.
- Sistem hızlı ve adımsal olarak çalışacaktır.(Her adım birbirinin devamı şeklinde olacaktır.)
- Çok kullanıcı girişine karşı dayanıklı olmalıdır.

Supportability

- Her türlü müşteriye adapte olabilecek bir sistem oluşturulmalıdır.
- Kullanıcı sayısının en az olduğu zamanlarda sistem bakımı yapılmalıdır. Bakım sırasında sistemsel oluşabilecek hatalara karşı login kullanıcılar logout duruma getirilmelidir.

Interface

- İnternet bağlantısı olan her makineden erişilebilmelidir.

Reports

- Günlük, haftalık, aylık ve yıllık olarak raporlar sistemde kayıt altında tutulmalıdır.

Software and Hardware Restrictions

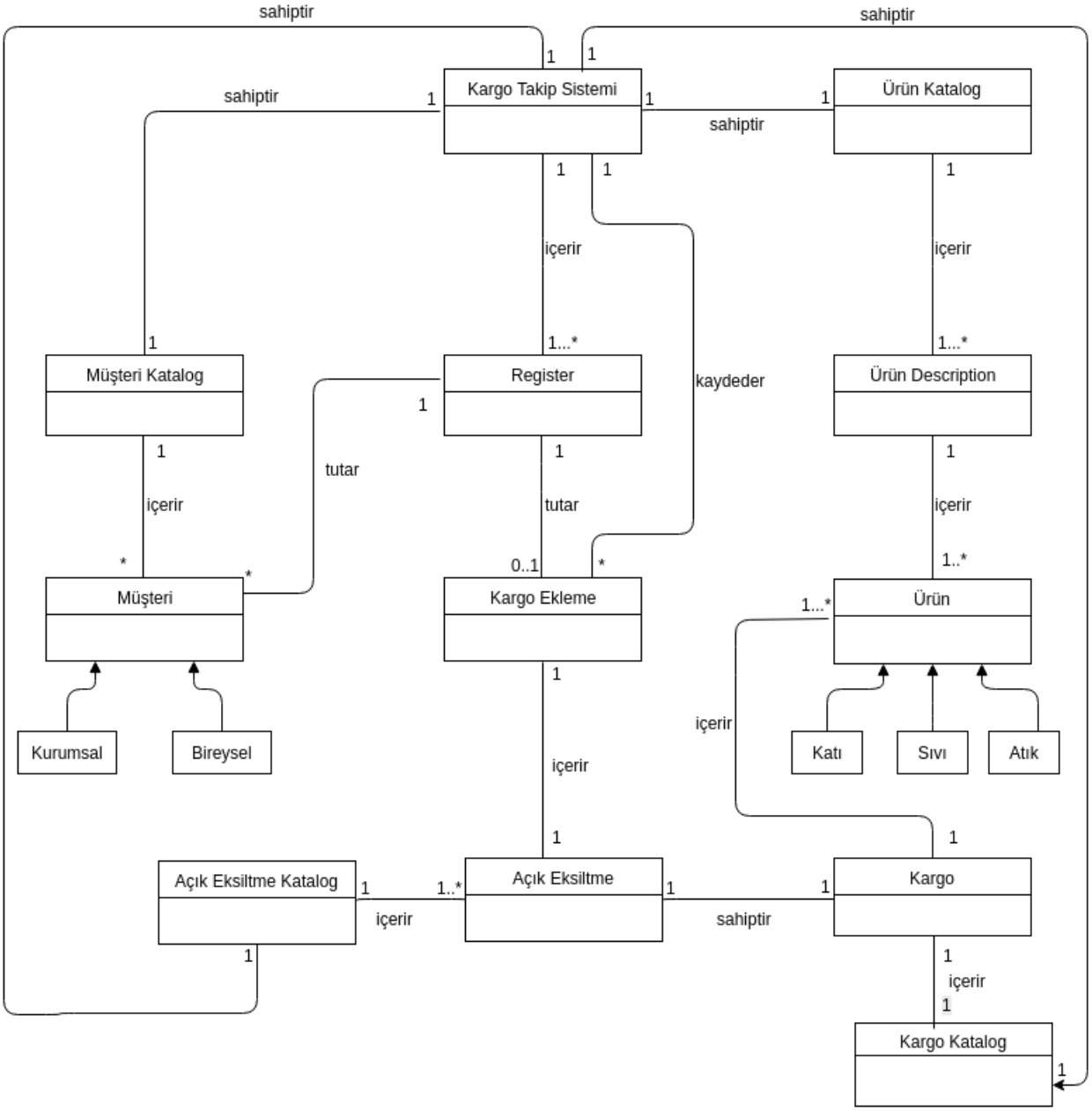
- Sistem, Windows işletim sistemlerinde çalışabilmelidir.

Legal

- Sistem için gerekli belgeler ve sertifikalar sistem yöneticisinde bulunmalıdır.

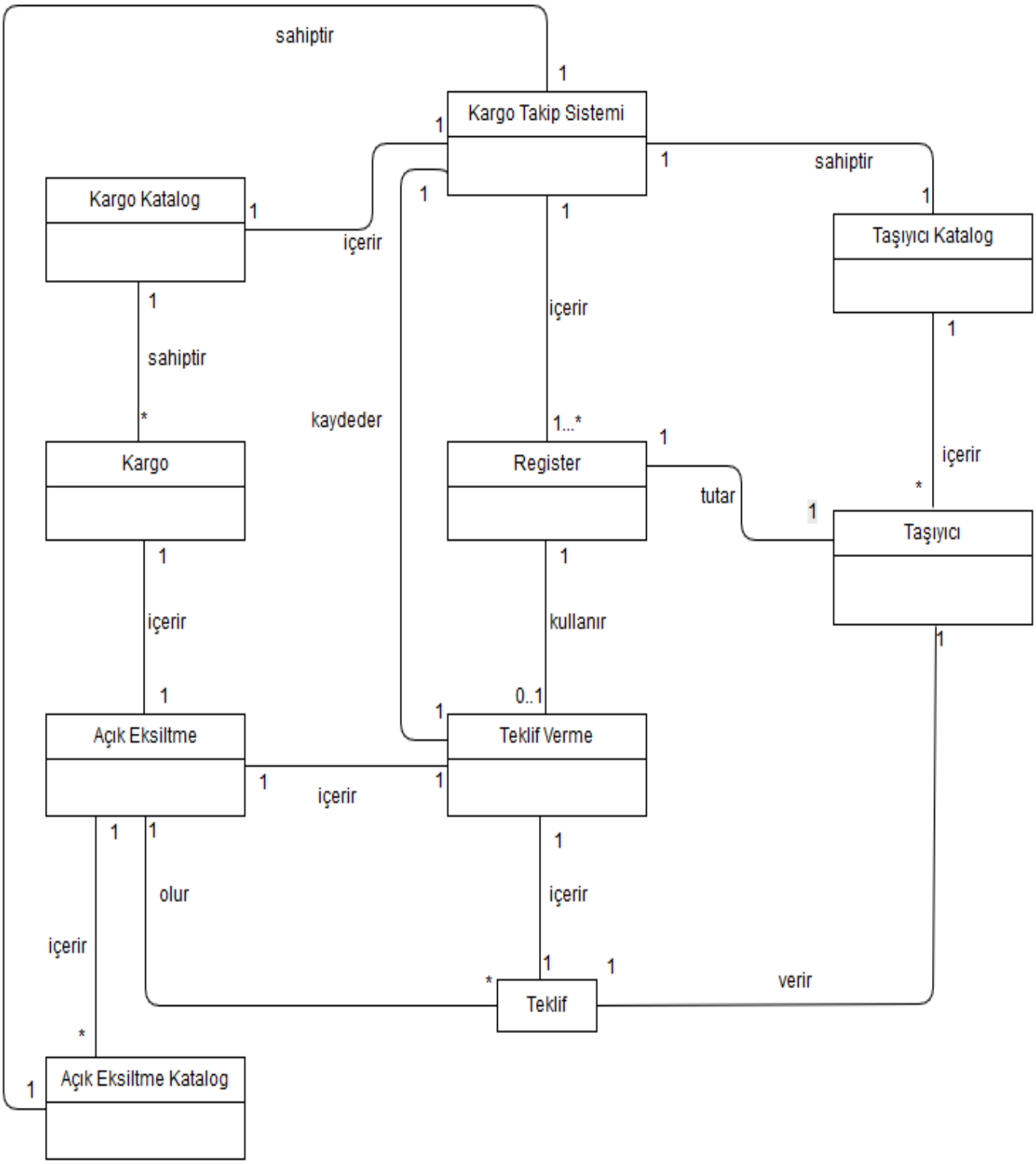
USE CASE 1 DOMAIN MODEL

Kargonun Sisteme Eklenmesi



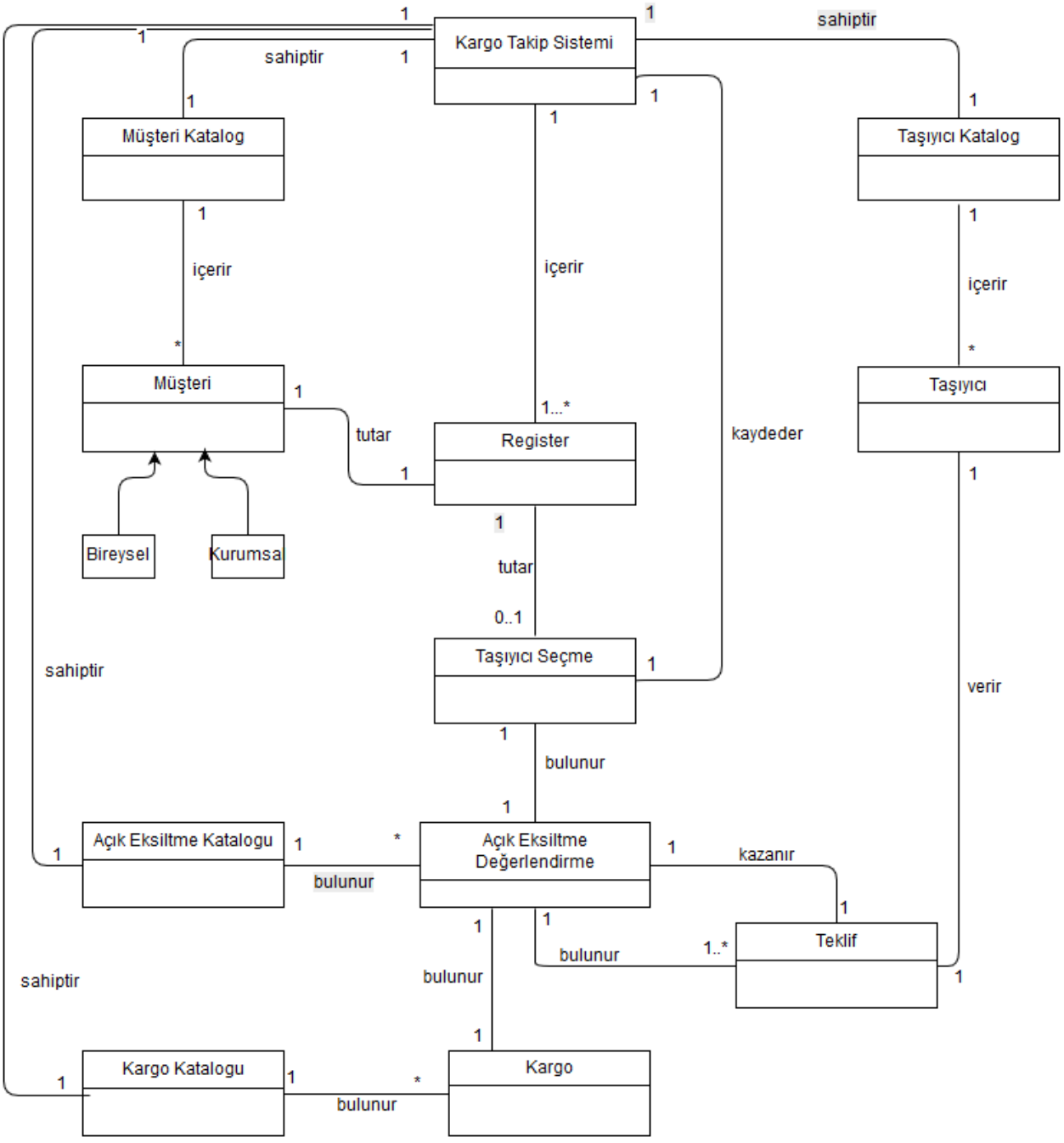
USE CASE 2 DOMAIN MODEL

Taşıyıcının Teklif Vermesi



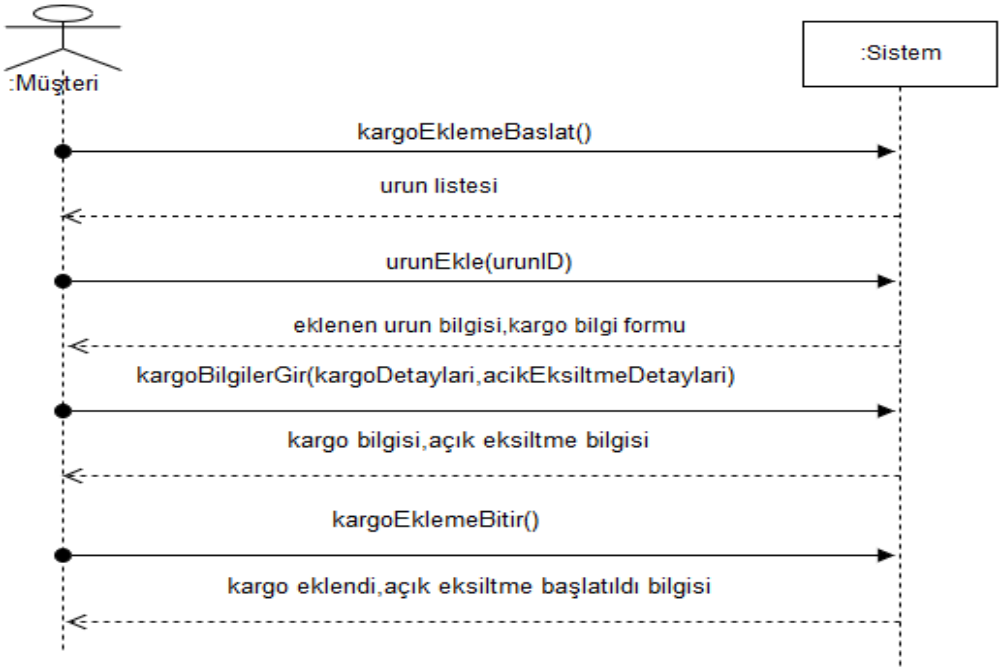
USE CASE 3 DOMAIN MODEL

Müşterinin Açık Eksiltmeyi Değerlendirmesi

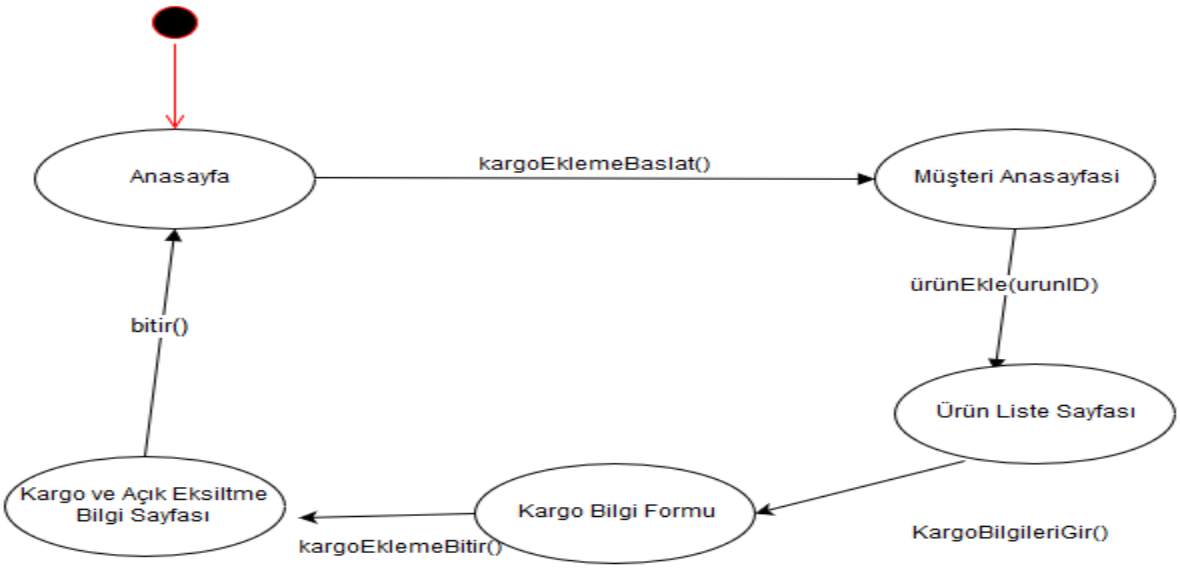


Use Case 1 SSD Model

Kargonun Sisteme Eklenmesi



Use Case 1 State Machine Diagram



USE CASE 1 OPERATION CONTRACT

Kargonun Sisteme Eklenmesi

OC1 **:** kargoEklemeBaslat()

Operation **:** kargoEklemeBaslat()

Cross References : UseCases:Kargo Ekleme

Precondition **: ----**

Postcondition **:**

- Kargo ekleme sınıfının ke isminde bir nesnesi oluşturulmuş olmalı
- ke nesnesi Register nesnesi ile ilişkilendirilmiş olmalı
- Açık Eksiltme sınıfının ae isminde bir nesnesi oluşturulmuş olmalı
- ae nesnesi ke nesnesi ile ilişkilendirilmiş olmalı
- Kargo sınıfının k nesnesi oluşturulmuş olmalı
- k nesnesi ae nesnesi ile ilişkilendirilmiş olmalı

OC2 **:** urunEkle()

Operation **:** urunEkle(urunID)

Cross References: UseCases:Kargo Ekleme

Precondition **:** Devam eden Kargo Ekleme nesnesi oluşturulmuş olmalı

Postcondition **:**

- Devam eden Kargo sınıfının nesnesi parametre olarak gelen urunID'nin belirttiği ürün nesnesi ile ilişkilendirilmiş olmalı.

OC3 **:** kargoBilgileriGir()

Operation **:** kargoBilgileriGir(kargoDetaylari,acikEksiltmeDetaylari)

Cross References: UseCases:Kargo Ekleme

Precondition **:** Devam eden Kargo Ekleme nesnesi oluşturulmuş olmalı

Postcondition **:**

- Devam eden Kargo nesnesinin kargoDetaylari özelliği, parametre olarak gelen kargoDetaylari ile set edilmiş olmalı
- Devam eden Kargo nesnesinin acikEksiltmeDetaylari özelliği, parametre olarak gelen acikEksiltmeDetaylari ile set edilmiş olmalı

OC4 **:** kargoEklemeBitir()

Operation **:** kargoEklemeBitir()

Cross References: UseCases:Kargo Ekleme

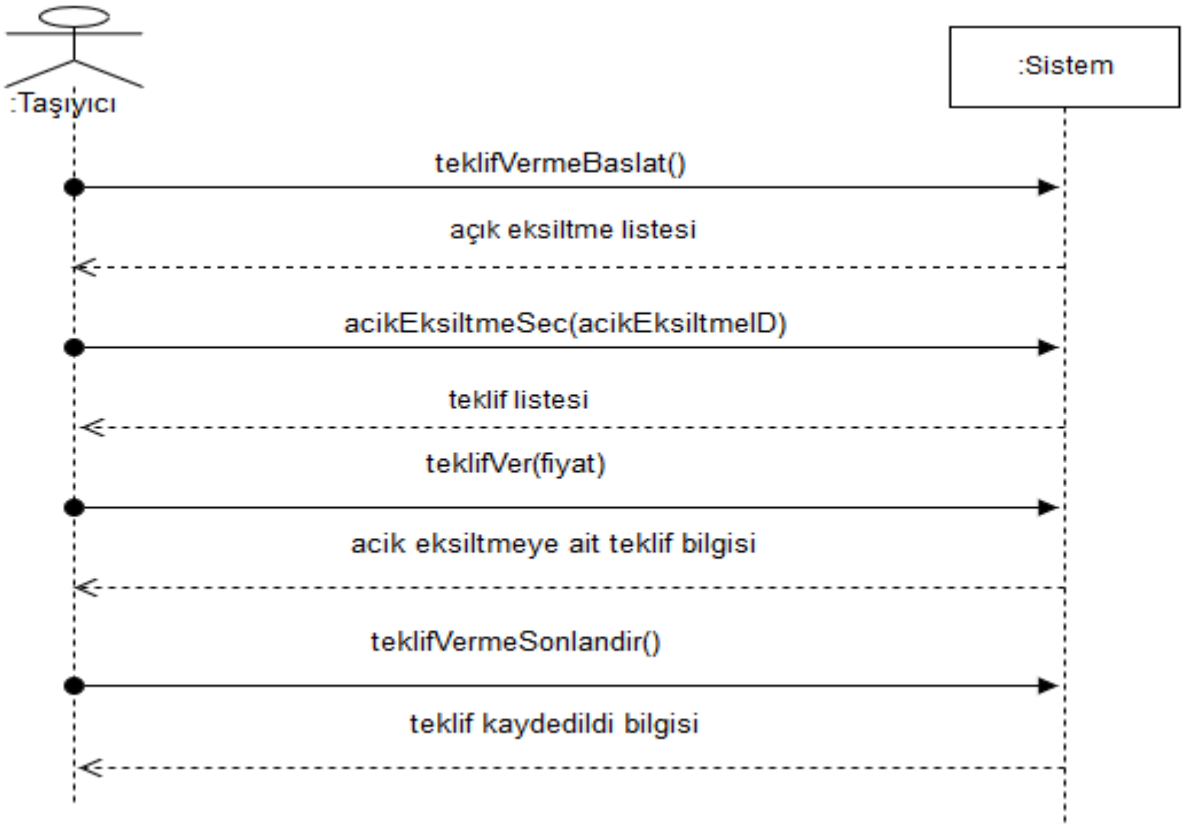
Precondition **:** Devam eden Kargo Ekleme nesnesi oluşturulmuş olmalı

Postcondition **:**

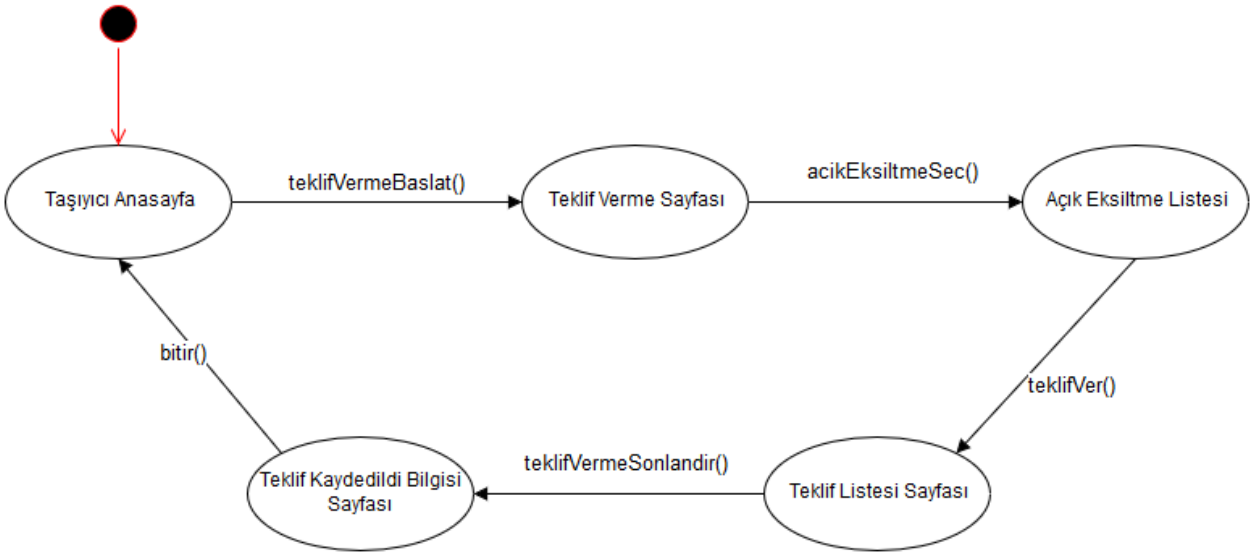
- Devam eden Kargo Ekleme nesnesinin complete özelliği true ile set edilmiş olmalı
- Devam eden Kargo Ekleme nesnesi Kargo Takip Sistemi ile ilişkilendirilmiş olmalı

Use Case 2 SSD Model

Taşıyıcının Teklif Vermesi



Use Case 2 State Machine Diagram



USE CASE 2 OPERATION CONTRACT

OC1 **:** teklifVermeBaslat()

Operation **:** teklifVermeBaslat()

Cross References : Use Cases:Taşıyıcının Teklif Vermesi

Precondition **: ----**

Postcondition **:**

- Teklif verme sınıfının tv isminde bir nesnesi oluşturulmuş olmalı
- tv nesnesi Register nesnesi ile ilişkilendirilmiş olmalı

OC2 **:** acikEksiltmeSec()

Operation **:** acikEksiltmeSec(acikEksiltmeID)

Cross References **:** Use Cases:Taşıyıcının Teklif Vermesi

Precondition **:** Devam eden TeklifVerme nesnesi oluşturulmuş olmalı

Postcondition **:**

- Devam eden TeklifVerme nesnesi ile parametre olarak gelen acikEksiltmeID'nin belirttiği AçıkEksiltme nesnesi ile ilişkilendirilmiş olmalı

OC3 **:** teklifVer()

Operation **:** teklifVer(fiyat)

Cross References **:** Use Cases:Taşıyıcının Teklif Vermesi

Precondition **:** Devam eden TeklifVerme nesnesi oluşturulmuş olmalı

Postcondition **:**

- Teklif sınıfının t isminde bir nesnesi oluşturulmuş olmalı
- t nesnesi devam eden TeklifVerme nesnesi ile ilişkilendirilmiş olmalı
- t nesnesi AçıkEksiltme nesnesi ile ilişkilendirilmiş olmalı
- t nesnesi Register nesnesi ile ilişkili olan Taşıyıcı nesnesi ile ilişkilendirilmiş olmalı
- t nesnesinin fiyat özelliği parametre olarak gelen fiyat ile set edilmiş olmalı.

OC3 **:** teklifVermeSonlandir()

Operation **:** teklifVermeSonlandir()

Cross References **:** Use Cases:Taşıyıcının Teklif Vermesi

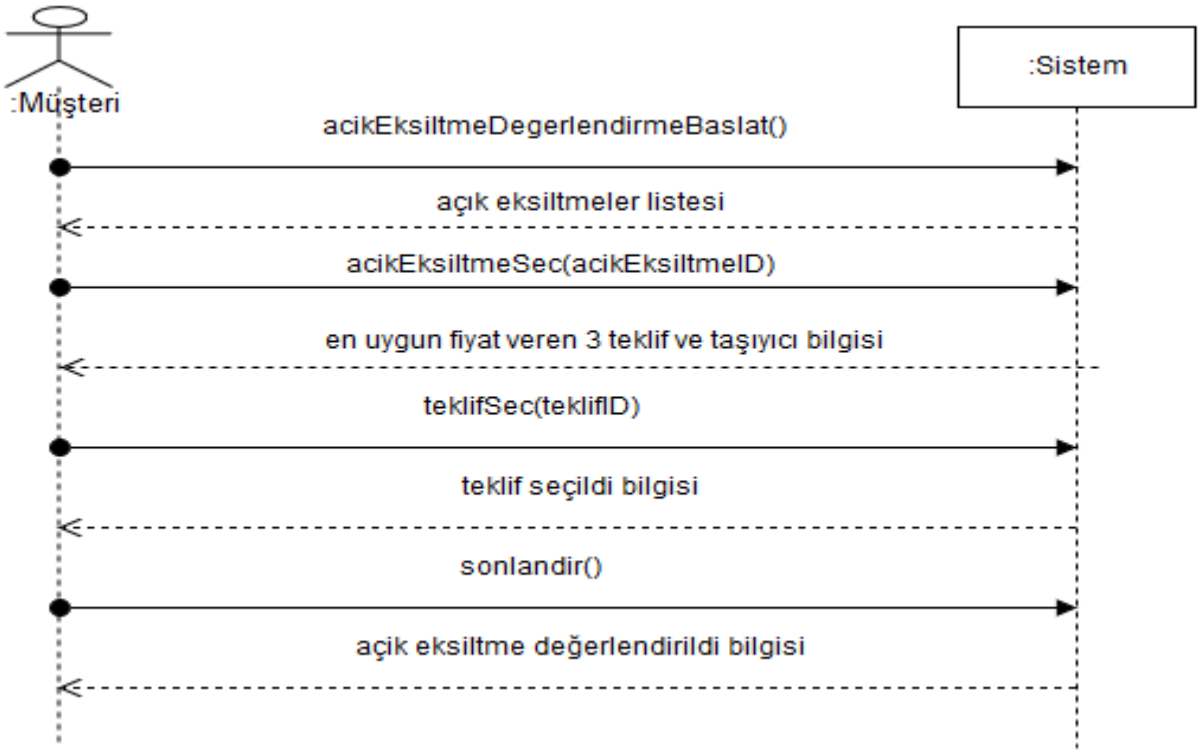
Precondition **:** Devam eden TeklifVerme nesnesi oluşturulmuş olmalı

Postcondition **:**

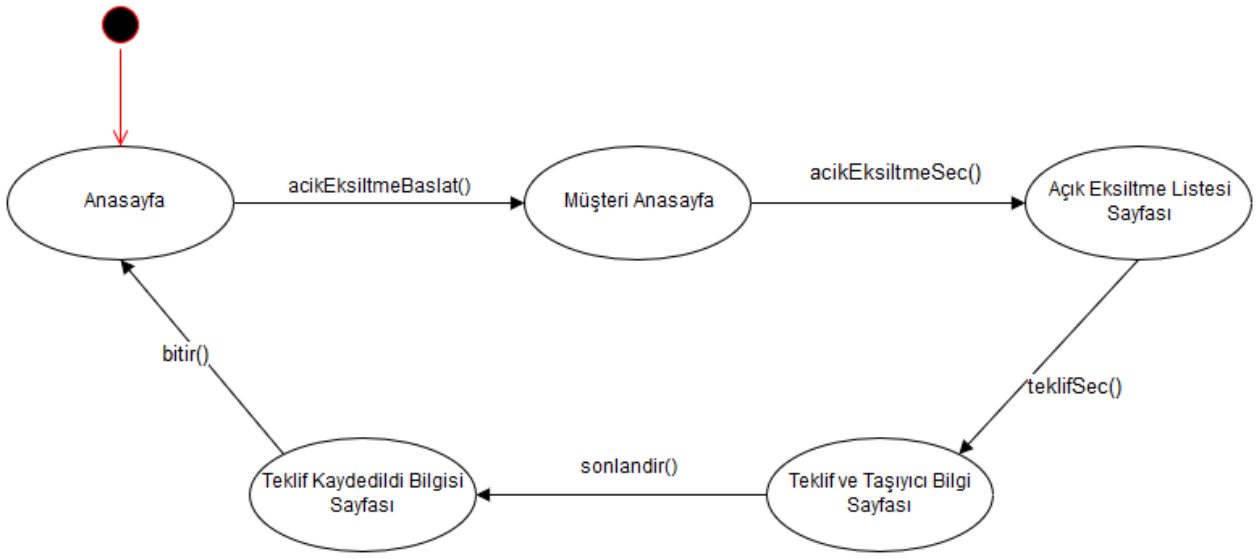
- Devam eden TeklifVerme nesnesi KargoTakipSistemi nesnesi ile ilişkilendirilmiş olmalı.

Use Case 3 SSD Model

Müşterinin Açık Eksiltmeyi Değerlendirmesi



Use Case 3 State Machine Diagram



USE CASE 3 OPERATION CONTRACT

Müşterinin Açık Eksiltmeyi Değerlendirmesi

OC1 **:** acikEksiltmeDegerlendirBaslat()

Operation **:** acikEksiltmeDegerlendirBaslat()

Cross References : UseCases:Müşterinin Açık Eksiltmeyi Değerlendirmesi

Precondition **: ----**

Postcondition **:**

- Açık Eksiltme Değerlendirme sınıfının aed isminde bir nesnesi oluşturulmuş olmalı
- aed nesnesi Register sınıfının nesnesi ile ilişkilendirilmiş olmalı

OC2 **:** acikEksiltmeSec()

Operation **:** acikEksiltmeSec(acikEksiltmeID)

Cross References: UseCases:Müşterinin Açık Eksiltmeyi Değerlendirmesi

Precondition **:** Açık Eksiltme Değerlendirme sınıfının devam eden bir nesnesi oluşturulmuş olmalıdır

Postcondition **:**

- Devam eden Açık Eksiltme Değerlendirme nesnesi parametre olarak gelen acikEksiltmeID nin belirttiği Açık Eksiltme nesnesi ile ilişkilendirilmiş olmalı

OC3 **:** teklifSec()

Operation **:** teklifSec(teklifID)

Cross References: UseCases:Müşterinin Açık Eksiltmeyi Değerlendirmesi

Precondition **:** Açık Eksiltme Değerlendirme sınıfının devam eden bir nesnesi oluşturulmuş olmalıdır

Postcondition **:**

- Devam eden Açık Eksiltme nesnesi parametre olarak gelen teklifID nin belirttiği Teklif nesnesi ile ilişkilendirilmiş olmalı
- Devam eden Açık Eksiltme nesnesinin degerlendirildi özelliği true ile set edilmiş olmalı.

OC4 **:** sonlandir()

Operation **:** sonlandir()

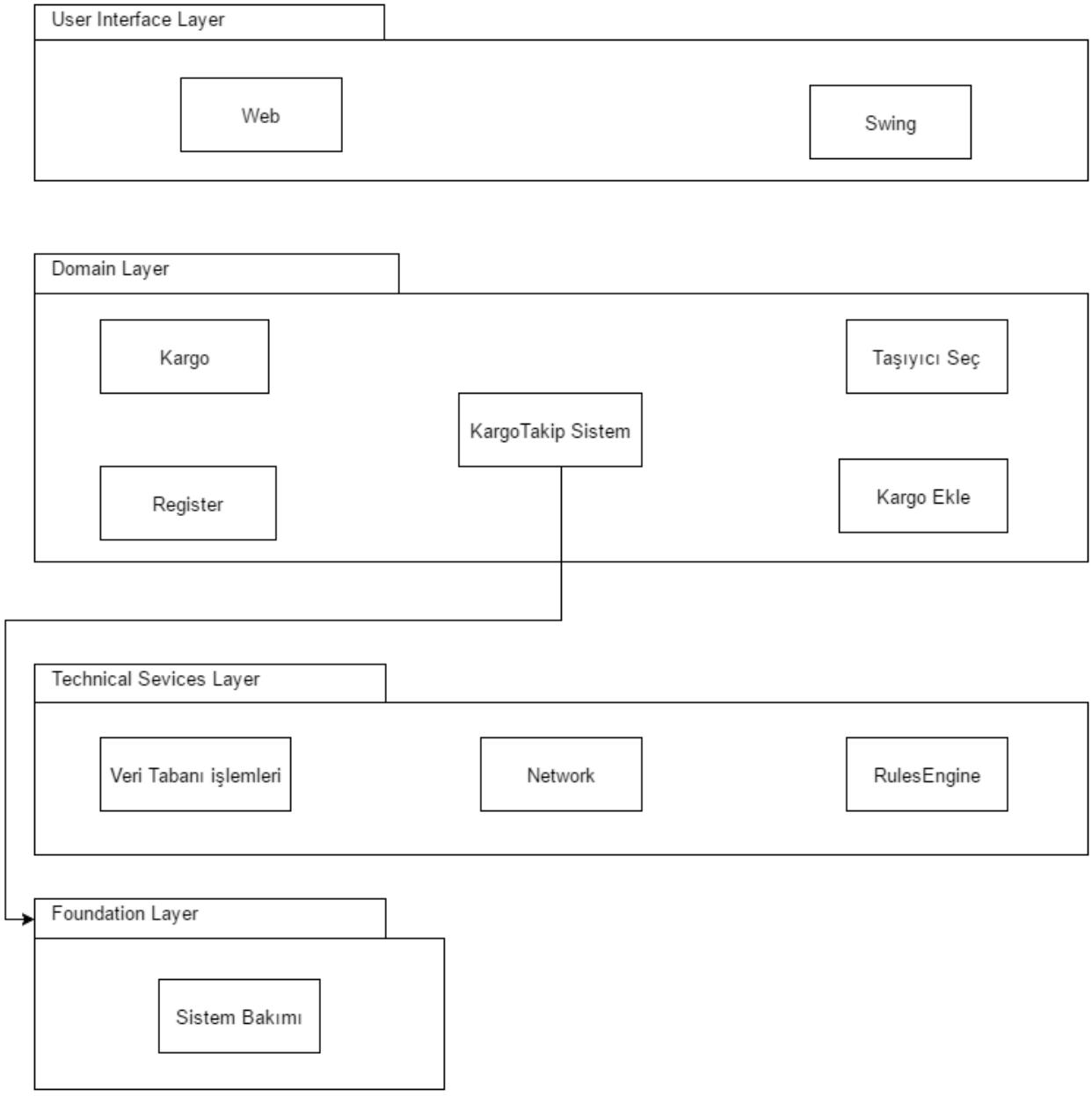
Cross References **:** UseCases:Müşterinin Açık Eksiltmeyi Değerlendirmesi

Precondition **:** Açık Eksiltme Değerlendirme sınıfının devam eden bir taşıyıcı seçme nesnesi olmuş olmalıdır

Postcondition **:**

- Devam eden Açık Eksiltme Değerlendirme nesnesi ile Kargo Takip Sistemi nesnesi ilişkilendirilmiş olmalı

Logical Architecture

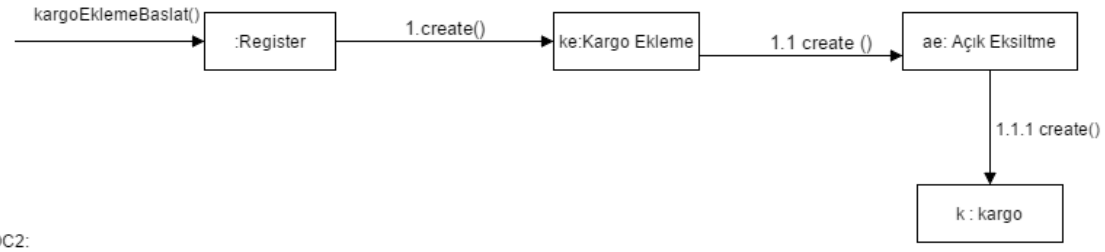


User Interface katmanından başlatılan sistem olayları Domain katmanındaki gerekli sınıflara gönderilir. Bu çağrılan sınıflar gelen verileri işler. Domain katmanı gelen bilgileri gerekli olan alt katmanlara aktarır veya gerekli destekleri sağlar.

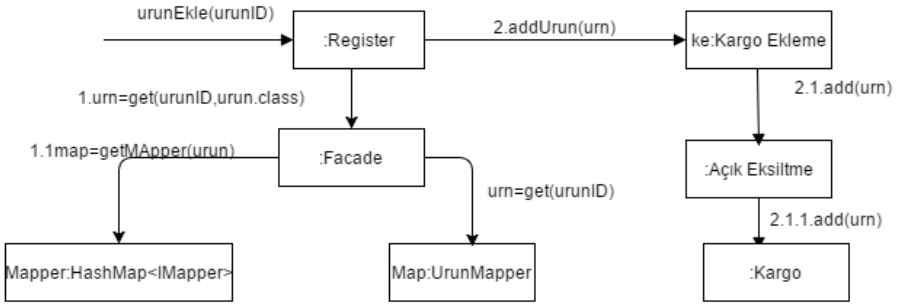
USE CASE 1 INTERACTION DIAGRAM

Kargonun Sisteme Eklenmesi

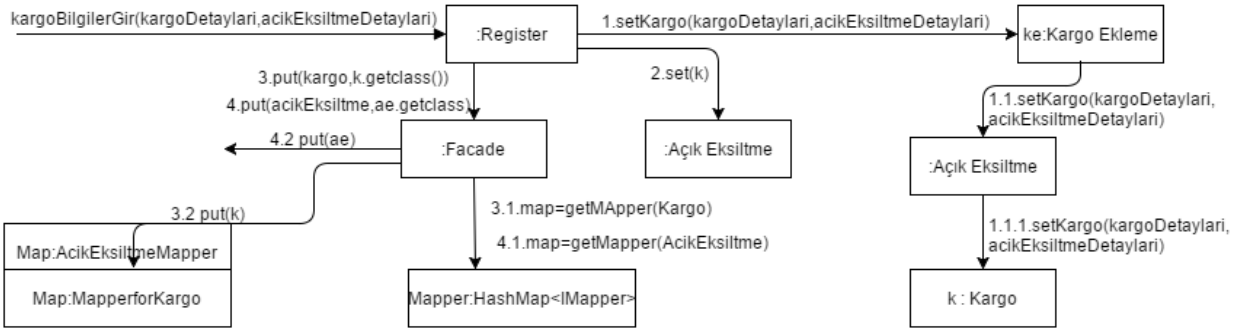
OC1:



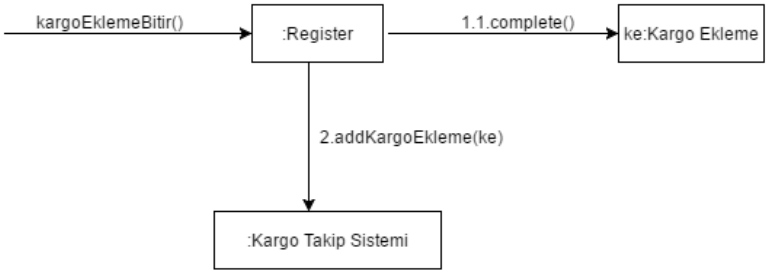
OC2:



OC3:



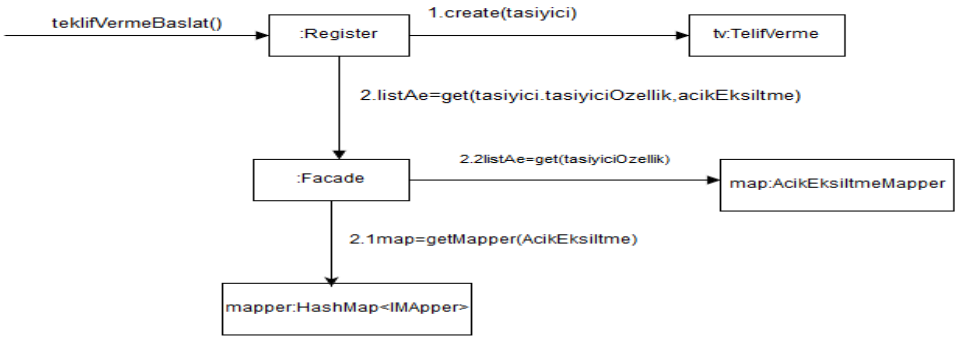
OC4:



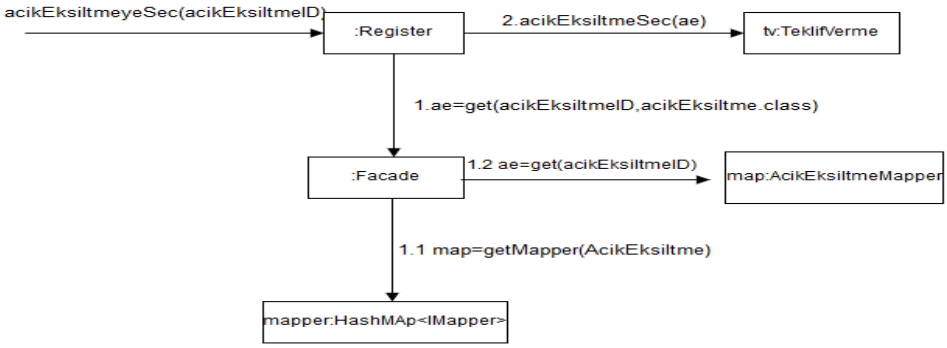
USE CASE 2 INTERACTION DIAGRAM

Taşıyıcının Açık Eksiltmeye Katılması

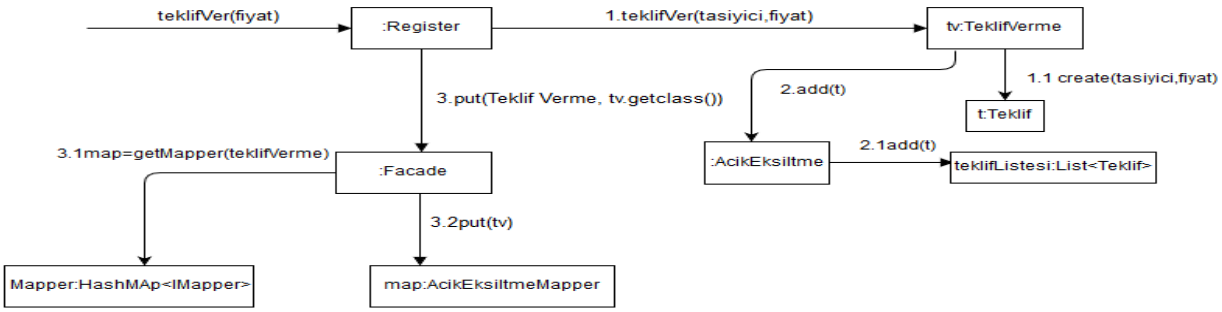
OC1:



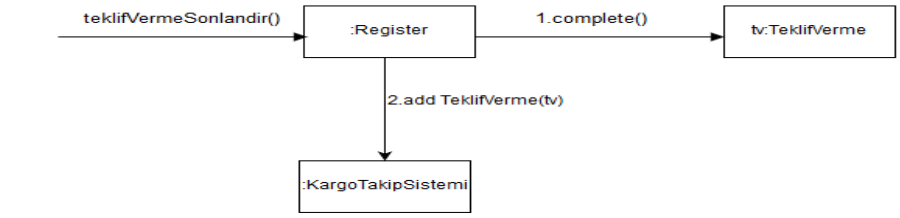
OC2:



OC3:



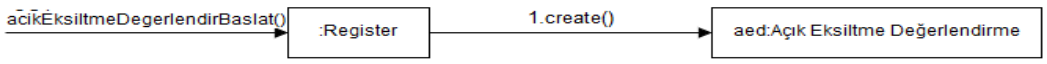
OC4:



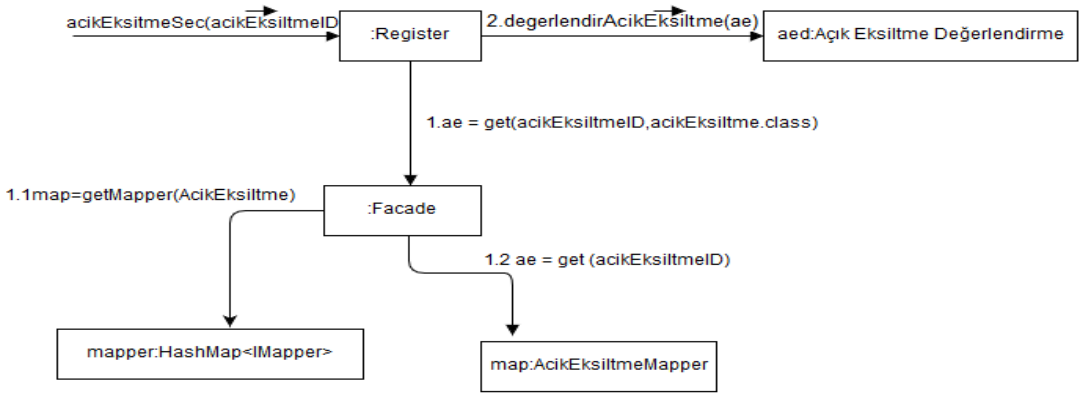
USE CASE 3 INTERACTION DIAGRAM

Müşterinin Açık Eksiltmeyi Değerlendirmesi

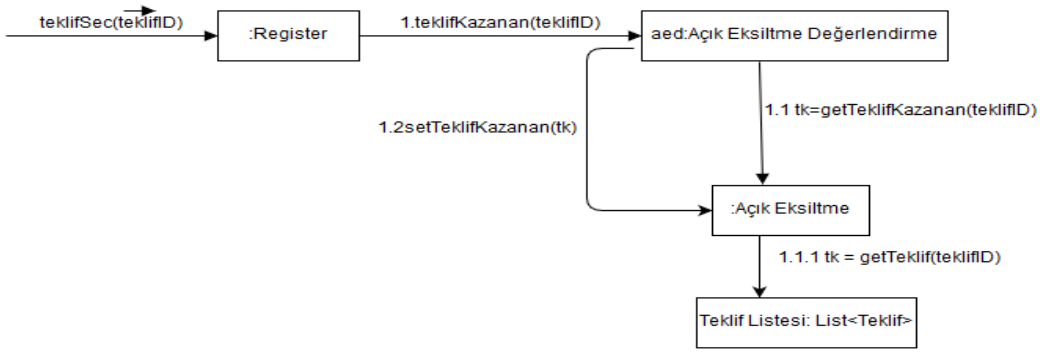
OC1:



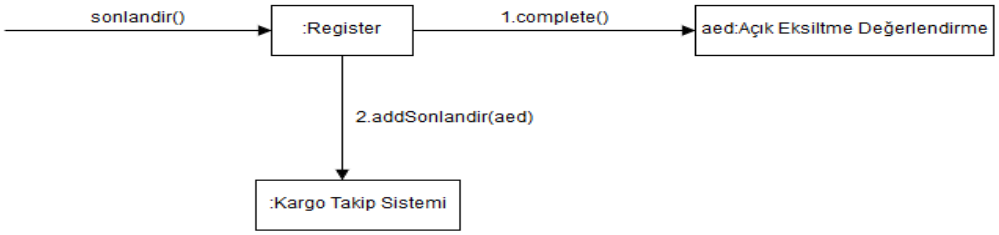
OC2:



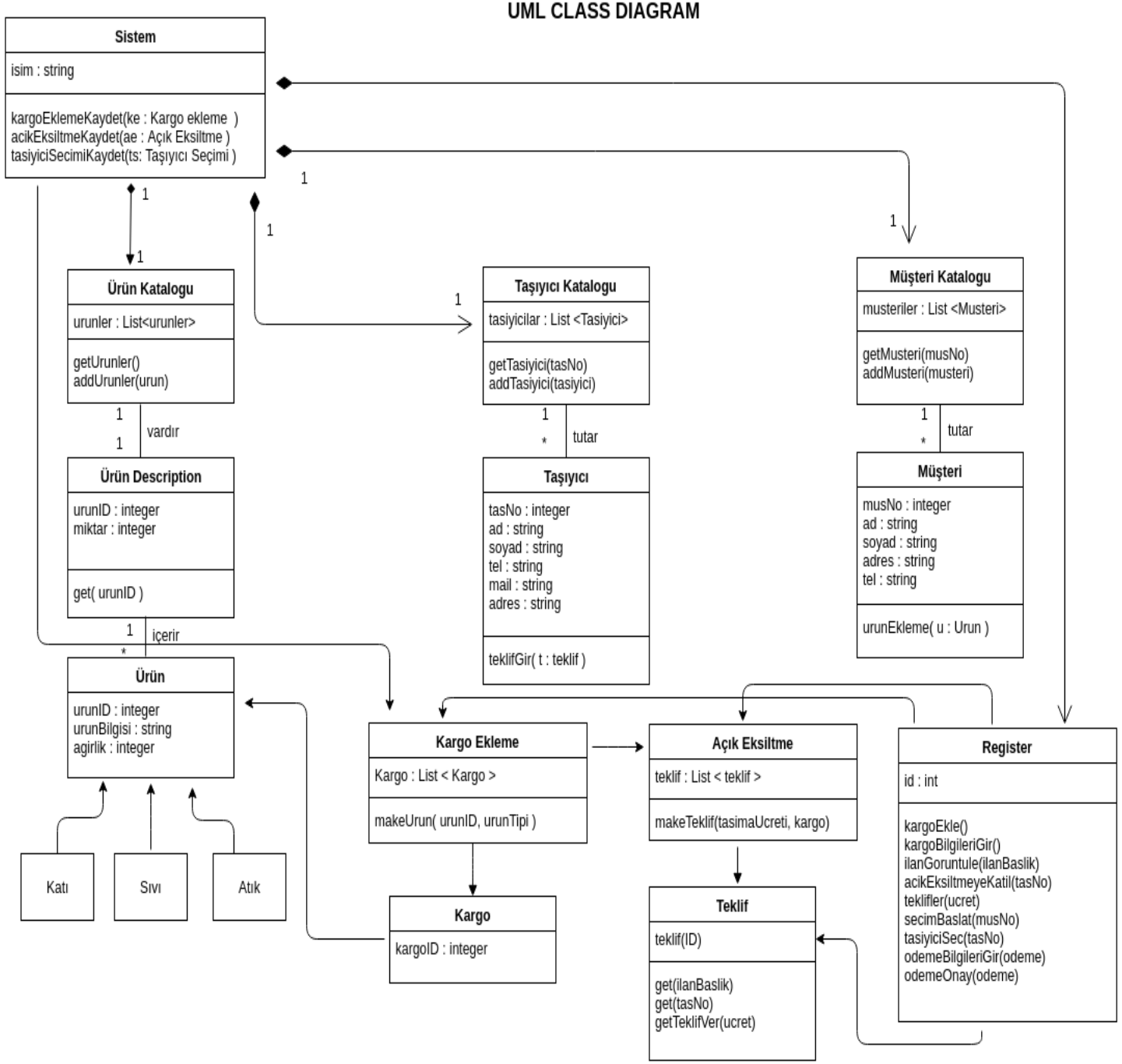
OC3:



OC4:



UML CLASS DIAGRAM



Kodlar

Testler

```
using System;
using Microsoft.VisualStudio.TestTools.UnitTesting;
using TestNDAT;
using TestNDAT.Kişiler;
using TestNDAT.ÜrünBilgileri;

namespace UnitTestProject1
{
    [TestClass]
    public class UnitTest1
    {
        Register actualregister;
        Müşteri actualMüşteri;
        Taşıyıcı actualTaşıyıcı;
        public UnitTest1()
        {
            actualregister = new Register();
            actualMüşteri = new Müşteri("ali", "asdasd");
            actualregister.kullanıcı = actualMüşteri;
            actualregister.KargoEkle();
            Boyut actualBoyut = new Boyut(3,4,5);
            Ağırlık actualAğırlık = new Ağırlık(10);
            Tür.Türler actualTür = Tür.Türler.EvcilHayvan;
            actualregister.ÜrünOluştur(actualBoyut, actualAğırlık, actualTür);
            actualregister.ÜrünEkle(0, 7);
            actualregister.BaşlangıçFiyatıBelirle(new Fiyat(20.0f));
            actualregister.Onay();
            actualTaşıyıcı = new Taşıyıcı(Kişi.KullanıcıTürleri.Taşıyıcı, "veli",
"sadsf");
            actualregister.TaşıyıcıEkle(actualTaşıyıcı);
            actualregister.TaşıyıcıGirişi(1);
            actualregister.AçıkEksiltmeyeKatıl();
            actualregister.İşSeç(0);
            actualregister.FiyatTeklifEt(new Fiyat(15));
            actualregister.MüşteriGirişi(0);
            actualregister.TaşıyıcıSeçmeyeKatıl();
            actualregister.KargoİşiSeç(0);
            actualregister.TeklifSeç(1);
            actualregister.TaşıyıcıOnayla();
        }


        #region Use Case UC1: Müşterinin Kargo Eklemesi Testleri

        [TestMethod]
        public void MüşteriKullanıcıGirişiTest()
        {
            actualregister.MüşteriEkle(actualMüşteri);
            actualregister.MüşteriGirişi(12);
            Assert.AreEqual(actualregister.kullanıcı, actualMüşteri);
        }

        [TestMethod]
        public void KargoEkleIDTest()
        {
```

```
            Kargo expectedKargo = new Kargo(actualMüşteri);
            expectedKargo.ID = 1;
            Assert.AreEqual(expectedKargo.ID, actualregister.kargo.ID);
        }

        [TestMethod]
        public void KargoEkleMüşteriTest()
        {
            Kargo expectedKargo = new Kargo(actualMüşteri);
            expectedKargo.ID = 0;
            Assert.AreEqual(expectedKargo.müşteri, actualregister.kargo.müşteri);
        }

        [TestMethod]
        public void ÜrünOluşturTest()
        {
            Boyut expectedBoyut = new Boyut(3,4,5);
            Ağırlık expectedAğırlık = new Ağırlık(10);
            Tür.Türler expectedTür = Tür.Türler.EvcilHayvan;
            ÜrünDescription expectedDescription = new ÜrünDescription(expectedBoyut,
expectedAğırlık, expectedTür);
            Assert.AreEqual(expectedDescription.ürünID,
                ((ÜrünDescription)actualregister.facade.Get(0,
typeof(ÜrünDescription))).ürünID + 1,
                expectedDescription.ürünID + " != " +
((ÜrünDescription)actualregister.facade.Get(0, typeof(ÜrünDescription))).ürünID
                );
        }

        [TestMethod]
        public void ÜrünEkleTest()
        {
            Boyut expectedBoyut = new Boyut(3, 4, 5);
            Ağırlık expectedAğırlık = new Ağırlık(10);
            Tür.Türler expectedTür = Tür.Türler.EvcilHayvan;
            ÜrünDescription expectedDescription = new ÜrünDescription(expectedBoyut,
expectedAğırlık, expectedTür);
            ÜrünLineItem expectedÜrünLineItem = new ÜrünLineItem(expectedDescription, 7);
            Assert.AreEqual(expectedÜrünLineItem.ürünDescription.ürünID,
                actualregister.kargo.ürünLineItem.ürünDescription.ürünID + 1);
        }

        [TestMethod]
        public void BaşlangıçFiyatıBelirleFiyatTest()
        {
            Fiyat expectedFiyat = new Fiyat(20.0f);
            Fiyat actualFiyat = actualregister.kargo.minimumFiyat;
            Assert.AreEqual(expectedFiyat.değer, actualFiyat.değer);
        }

        [TestMethod]
        public void BaşlangıçFiyatıBelirleIDveKişiTest()
        {
            Fiyat expectedFiyat = new Fiyat(20.0f);
            Teklif expectedTeklif = new Teklif(0, actualMüşteri, expectedFiyat);
            Teklif actualTeklif = (Teklif)actualregister.facade.Get(0, typeof(Teklif));
            Assert.AreEqual(expectedTeklif.ID, actualTeklif.ID + 1);
            Assert.AreEqual(expectedTeklif.kişi, actualTeklif.kişi);
```

```
}

[TestMethod]
public void KargoİşiOnayTest()
{
    bool expectedIsComplete = true;
    bool actualIsComplete = actualregister.kargo.isComplete;
    Assert.AreEqual(expectedIsComplete, actualIsComplete);
}

#endregion
#region Use Case UC2: Taşıyıcının Teklif Vermesi Testleri

[TestMethod]
public void TaşıyıcıKullanıcıGirişiTest()
{
    Assert.AreEqual(actualregister.kullanıcı, actualTaşıyıcı);
}

[TestMethod]
public void AçıkEksiltmeyeKatılTest()
{
    Assert.AreEqual(actualTaşıyıcı, actualregister.açıkEksiltme.taşıyıcı);
}

[TestMethod]
public void İşSeçTest()
{
    Assert.AreEqual(actualregister.kargo, actualregister.açıkEksiltme.kargoİşi);
}

[TestMethod]
public void FiyatTeklifiTest()
{
    bool actualTeklifYapıldıMı = actualregister.açıkEksiltme.teklifYapıldıMı;
    Assert.AreEqual(true, actualTeklifYapıldıMı);
}

[TestMethod]
public void AçıkEksiltmeOnayTest()
{
    actualregister.AçıkEksiltmeOnayla();
    bool actualIscomplete = actualregister.açıkEksiltme.isCompleted;
    Assert.AreEqual(true, actualIscomplete);
}
#endregion
#region Use Case UC3: Müşterinin Açık Eksiltme Değerlendirmesi Tests

[TestMethod]
public void MTSMüşteriKullanıcıGirişiTest()
{
    Assert.AreEqual(actualMüşteri.Ad, actualregister.kullanıcı.Ad);
}

[TestMethod]
public void TaşıyıcıSeçmeyeKatılTest()
{
    Assert.AreEqual((Müşteri)actualregister.kullanıcı,
```

```
                actualregister.mts.işlemYapanMüşteri);
        }

        [TestMethod]
        public void KargoİşiSeçTest()
        {
            Assert.AreEqual(actualregister.kargo.ID,
                actualregister.mts.kargoİşi.ID);
        }

        [TestMethod]
        public void TaşıyıcıTeklifiSeçTest()
        {
            Taşıyıcı actualTaşıyıcı = actualregister.mts.seçilecekTaşıyıcı;
            Assert.AreEqual(this.actualTaşıyıcı, actualTaşıyıcı);
        }

        [TestMethod]
        public void TaşıyıcıyıOnaylaTest()
        {

            bool actualIsCompleted = actualregister.mts.isCompleted;
            Assert.AreEqual(true, actualIsCompleted);
        }

        #endregion
    }
}
```

Müşteri

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace TestNDAT.Kişiler
{
    public class Müşteri: Kişi
    {
        public Müşteri(String Ad, String Soyad) : base( Ad, Soyad)
        {
            Türü = KullanıcıTürleri.Müşteri;
        }
        public override string ToString()
        {
            return base.ToString();
        }
    }
}
```

Taşıyıcı

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace TestNDAT.Kişiler
{
    public class Taşıyıcı : Kişi
    {
        public Taşıyıcı(String Ad, String Soyad) : base(Ad, Soyad)
        {
            this.Türü = KullanıcıTürleri.Taşıyıcı;
        }

        public Taşıyıcı(KullanıcıTürleri türü, string ad, string soyad) : base(türü, ad,
soyad)
        {
        }

        public override string ToString()
        {
            return base.ToString();
        }
    }
}
```

Kişi

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace TestNDAT.Kişiler
{
    public abstract class Kişi
    {

        public enum KullanıcıTürleri { Müşteri, Taşıyıcı };
        public int OID { get; set; }
        public String Ad { get; set; }
        public String Soyad { get; set; }
        public KullanıcıTürleri Türü { get; set; }
        public Kişi(KullanıcıTürleri türü, String ad, String soyad)
        {
            this.Türü = türü;
            this.OID = IDGenerator.GetKişiID();
            this.Ad = ad;
            this.Soyad = soyad;
        }

        public Kişi(string Ad1, string Soyad1)
        {
            this.OID = IDGenerator.GetKişiID();
```

```
            this.Ad = Ad1;
            this.Soyad = Soyad1;
        }

        public override string ToString()
        {
            return OID + " | " + Türü.ToString() + " | " + this.Ad + " | " + this.Soyad;
        }

    }
}
```

IMapper

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using TestNDAT.ÜrünBilgileri;

namespace TestNDAT.Facade.Mappers
{
    public interface IMapper
    {
        Object Get(int OID);
        void Put(Object obj);

        List<String> GetAll();
    }
}
```

AçıkEksiltmeMapper

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace TestNDAT.Facade.Mappers
{
    public class AçıkEksiltmeMapper : IMapper
    {
        private AçıkEksiltme açıkEksiltme;
        public object Get(int OID)
        {
            return this.açıkEksiltme;
        }

        public void Put(object obj)
        {
            açıkEksiltme = (AçıkEksiltme)obj;
        }



        public List<String> GetAll()
        {
            throw new NotImplementedException();
        }
```

```
    }
}
```

KargoMapper

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace TestNDAT.Facade.Mappers
{
    public class KargoMapper : IMapper
    {

        public static Dictionary<int, Kargo> kargoİşleri;
        public static List<String> kargoİşleriOutput;
        public KargoMapper()
        {
            kargoİşleri = new Dictionary<int, Kargo>();
            kargoİşleriOutput = new List<string>();
        }
        public object Get(int OID)
        {
            try
            {
                return kargoİşleri[OID];
            }
            catch (Exception)
            {
                return null;
            }

        }

        public void Put(object obj)
        {
            Kargo k = obj as Kargo;
            kargoİşleri.Add(k.ID, k);
            kargoİşleriOutput.Add(k.ToString());
        }

        public List<String> GetAll()
        {
            return kargoİşleriOutput;
        }
    }
}
```

KişiMapper

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace TestNDAT.Facade.Mappers
{
```

```
    public class KişiMapper : IMapper
    {
        TaşıyıcıMapper tm ;
        MüşteriMapper mm ;
        public KişiMapper()
        {
            tm = new TaşıyıcıMapper();
            mm = new MüşteriMapper();
        }

        public object Get(int OID)
        {
            throw new NotImplementedException();
        }

        public void Put(object obj)
        {
            throw new NotImplementedException();
        }

        public List<string> GetAll()
        {
            return  mm.GetAll()
                .Concat(tm.GetAll())
                .ToList();
        }
    }
}
```

MüşteriMapper

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using TestNDAT.Kişiler;
namespace TestNDAT.Facade.Mappers
{
    public class MüşteriMapper : IMapper
    {
        private static List<String> müşterilerOutput;
        private static Dictionary<int, Müşteri> müşteriler;
        public MüşteriMapper()
        {
            if (müşterilerOutput == null)
            {
                müşterilerOutput = new List<string>();
            }
            if (müşteriler == null)
            {
                müşteriler = new Dictionary<int, Müşteri>();
            }
        }
        public object Get(int OID)
        {
            try
            {
                return müşteriler[OID];
```

```
            }
            catch (Exception)
            {

                return null;
            }

        }

        public void Put(object obj)
        {
            Müşteri m = obj as Müşteri;
            müşterilerOutput.Add(m.ToString());
            müşteriler.Add(m.OID, m);
        }

        public List<String> GetAll()
        {
            return müşterilerOutput;
        }
    }
}
```

MüşterininAçıkEksiltmeyiDeğerlendirmesi

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace TestNDAT.Facade.Mappers
{
    public class MüşterininAçıkEksiltmeyiDeğerlendirmesiMapper : IMapper
    {
        MüşterininTaşıyıcıyıSeçmesi mts;
        public object Get(int OID)
        {
            return this.mts;
        }

        public void Put(object obj)
        {
            mts = obj as MüşterininTaşıyıcıyıSeçmesi;
        }




        public List<String> GetAll()
        {
            throw new NotImplementedException();
        }
    }
}
```

TaşıyıcıMapper

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
```

```
using System.Threading.Tasks;
using TestNDAT.Kişiler;

namespace TestNDAT.Facade.Mappers
{
    public class TaşıyıcıMapper : IMapper
    {
        public static List<String> taşıyıcılarOutput;
        private static Dictionary<int, Taşıyıcı> taşıyıcılar;
        public TaşıyıcıMapper()
        {
            if (taşıyıcılarOutput == null)
            {
                taşıyıcılarOutput = new List<string>();
            }
            if (taşıyıcılar == null)
            {
                taşıyıcılar = new Dictionary<int, Taşıyıcı>();
            }

        }
        public object Get(int OID)
        {
            try
            {
                return taşıyıcılar[OID];
            }
            catch (Exception)
            {
                return null;
            }
        }

        public void Put(object obj)
        {
            Taşıyıcı t = obj as Taşıyıcı;
            taşıyıcılarOutput.Add(t.ToString());
            taşıyıcılar.Add(t.OID, t);
        }
        public List<String> GetAll()
        {
            return taşıyıcılarOutput;
        }
    }
}
```

TeklifMapper

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using TestNDAT.Kişiler;
using TestNDAT.ÜrünBilgileri;

namespace TestNDAT.Facade.Mappers
{
```

```
    public class TeklifMapper: IMapper
    {
        Teklif t;
        public object Get(int OID)
        {
            return t;
        }

        public void Put(object obj)
        {
            t = (Teklif)obj;
        }



        public List<String> GetAll()
        {
            throw new NotImplementedException();
        }
    }
}
```

ÜrünDescriptionMapper

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using TestNDAT.ÜrünBilgileri;

namespace TestNDAT.Facade.Mappers
{
    public class ÜrünDescriptionMapper:IMapper
    {
        ÜrünDescription üd;

        public object Get(int OID)
        {
            return üd;
        }

        public void Put(object obj)
        {
            this.üd = (ÜrünDescription)obj;
        }



        public List<String> GetAll()
        {
            throw new NotImplementedException();
        }
    }
}
```

ÜrünMapper

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using TestNDAT.ÜrünBilgileri;

namespace TestNDAT.Facade.Mappers
{
    class ÜrünMapper : IMapper
    {
        Ürün ü;

        public object Get(int OID)
        {
            return this.ü;
        }

        public void Put(object obj)
        {
            this.ü = (Ürün)obj;
        }

        public List<String> GetAll()
        {
            throw new NotImplementedException();
        }
    }
}
```

DBFacade

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using TestNDAT.Facade.Mappers;
using TestNDAT.Kişiler;
using TestNDAT.ÜrünBilgileri;

namespace TestNDAT.Facade
{
    public class DBFacade
    {
        private Dictionary<Type, IMapper> mappers;
        private static DBFacade facade;
        private DBFacade()
        {
            mappers = new Dictionary<Type, IMapper>
            {
                { typeof(Müşteri), new MüşteriMapper() },
                { typeof(Teklif), new TeklifMapper() },
                { typeof(Ürün), new ÜrünMapper() },
                { typeof(ÜrünDescription), new ÜrünDescriptionMapper() },
                { typeof(Kargo), new KargoMapper() },
                { typeof(Taşıyıcı), new TaşıyıcıMapper() },
                { typeof(AçıkEksiltme), new AçıkEksiltmeMapper() },
```

```
                { typeof(MüşterininTaşıyıcıyıSeçmesi), new
MüşterininAçıkEksiltmeyiDeğerlendirmesiMapper() },
                { typeof(Kişi), new KişiMapper() },
            };
        }
        public static DBFacade GetInstance()
        {
            if (facade == null)
            {
                return new DBFacade();
            }
            else
            {
                return facade;
            }
        }

        public Object Get(int OID, Type persistenceClass)
        {
            IMapper mapper = mappers[persistenceClass];
            return mapper.Get(OID);
        }


        public void Put(Object obj, Type persistenceClass)
        {
            mappers[persistenceClass].Put(obj);
        }

        public List<String> GetAll(Type persistenceClass)
        {
            return mappers[persistenceClass].GetAll();
        }
    }
}
```

AçıkEksiltme

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using TestNDAT.Kişiler;
using TestNDAT.ÜrünBilgileri;

namespace TestNDAT
{
    public class AçıkEksiltme
    {
        public Taşıyıcı taşıyıcı;
        public Kargo kargoİşi { get; set; }
        public bool teklifYapıldıMı;
        public bool isCompleted;

        public AçıkEksiltme(Taşıyıcı taşıyıcı)
        {
            this.taşıyıcı = taşıyıcı;
        }
```

```
        public void İşSeç(Kargo kargoİşi)
        {
            this.kargoİşi = kargoİşi;
        }

        public Teklif FiyatTeklifEt(Fiyat fiyat)
        {
            if (fiyat.değer < kargoİşi.minimumFiyat.değer)
            {
                Teklif teklif = new Teklif(this.kargoİşi.ID, this.taşıyıcı, fiyat);
                teklifYapıldıMı = true;
                return teklif;
            }
            else
            {
                teklifYapıldıMı = false;
                return null;
            }
        }

        public void Onayla()
        {
            if (this.teklifYapıldıMı && this.kargoİşi != null && this.taşıyıcı != null)
            {
                isCompleted = true;
            }
            else
            {
                isCompleted = false;
            }
        }
    }
}
```

Kargo

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using TestNDAT.ÜrünBilgileri;
using TestNDAT.Kişiler;

namespace TestNDAT
{
    public class Kargo
    {
        public Müşteri müşteri;
        public int ID;
        public ÜrünLineItem ürünLineItem;
        public Fiyat minimumFiyat;
        public bool isComplete;

        public Kargo(Müşteri müşteri)
        {
            this.müşteri = müşteri;
        }

        public ÜrünDescription ÜrünOluştur(Boyut boyut, Ağırlık ağırlık, Tür.Türler tür)
```

```
        {
            ÜrünDescription üd = new ÜrünDescription(boyut, ağırlık, tür);
            return üd;
        }

        internal void ÜrünEkle(ÜrünDescription eklenecekÜrünDesc, int quantity)
        {
            this.ürünLineItem = new ÜrünLineItem(eklenecekÜrünDesc, quantity);
        }

        public Teklif BaşlangıçFiyatıBelirle(Fiyat fiyat)
        {
            this.minimumFiyat = fiyat;
            Teklif t = new Teklif(ID, müşteri, fiyat);
            return t;
        }

        internal bool Onay()
        {
            if (ID != null && this.minimumFiyat != null && this.müşteri != null &&
                this.ürünLineItem != null)
            {
                this.isComplete = true;
                return isComplete;
            }
            else
            {
                this.isComplete = false;
                return isComplete;
            }
        }

        public override string ToString()
        {
            return "ID: " + ID + "\n" +
                "LineItem: " + ürünLineItem.ToString() + "\n" +
                "Fiyat: " + minimumFiyat.değer.ToString() + "\n" +
                "Müşteri: " + müşteri.ToString();
        }
    }
}
```

MüşterininTaşıyıcıSeçmesi

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using TestNDAT.Kişiler;
using TestNDAT.ÜrünBilgileri;

namespace TestNDAT
{
    public class MüşterininTaşıyıcıyıSeçmesi
    {
        public Müşteri işlemYapanMüşteri;
        public Taşıyıcı seçilecekTaşıyıcı;
        public Kargo kargoİşi;
        public bool isCompleted;
```

```
        public MüşterininTaşıyıcıyıSeçmesi(Müşteri işlemYapanMüşteri)
        {
            this.işlemYapanMüşteri = işlemYapanMüşteri;
        }

        public void TaşıyıcıSeç(Taşıyıcı seçilecekTaşıyıcı)
        {
            this.seçilecekTaşıyıcı = seçilecekTaşıyıcı;
        }



        public void TaşıyıcıyıOnayla()
        {
            if (işlemYapanMüşteri != null && seçilecekTaşıyıcı != null)
            {
                isCompleted = true;
            }
            else
            {
                isCompleted = false;
            }
        }



        public void KargoİşiSeç(Kargo kargoİşi)
        {
            this.kargoİşi = kargoİşi;
        }
    }
}
```

Register

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using TestNDAT.Facade;
using TestNDAT.Kişiler;
using TestNDAT.ÜrünBilgileri;

namespace TestNDAT
{
    public class Register
    {
        public DBFacade facade;
        public Kişi kullanıcı;
        public Kargo kargo;
        public AçıkEksiltme açıkEksiltme;
        public MüşterininTaşıyıcıyıSeçmesi mts;
        public Register()
        {
            facade = DBFacade.GetInstance();
        }
```

```
#region Kullanıcı Girişi
public List<String> TümKullanıcılarıGetir()
{
    return facade.GetAll(typeof(Kişi));
}

public List<String> KullanıcıEkle(string ad, string soyad, string kullanıcıTürü)
{
    try
    {
        switch (kullanıcıTürü)
        {
            case "0":
                MüşteriEkle(new Müşteri(ad, soyad));
                return TümKullanıcılarıGetir();
            case "1":
                TaşıyıcıEkle(new Taşıyıcı(ad, soyad));
                return TümKullanıcılarıGetir();
            default:
                return null;
        }
    }
    catch (Exception)
    {

        return null;
    }


}

public bool KullanıcıGirişi(int input)
{
    if (!MüşteriGirişi(input))
    {
        if (!TaşıyıcıGirişi(input))
        {
            return false;
        }
    }
    return true;
}

public List<String> KullanıcılarıGetir()
{
    var kişiler = facade.GetAll(typeof(Kişi));
    if (kişiler == null) kişiler = new List<string>()
    {
        "Sisteme kayıtlı Herhangi bir kişi bulunmamaktadır"
    };
    return kişiler;
}

public void MüşteriEkle(Müşteri m)
{
    facade.Put(m, typeof(Müşteri));
}
```

```
        public bool MüşteriGirişi(int OID)
        {
            kullanıcı = (Müşteri)facade.Get(OID, typeof(Müşteri));
            if (kullanıcı == null) return false;
            return true;
        }
        public void TaşıyıcıEkle(Taşıyıcı t)
        {
            facade.Put(t, typeof(Taşıyıcı));
        }

        public bool TaşıyıcıGirişi(int OID)
        {
            kullanıcı = (Taşıyıcı)facade.Get(OID, typeof(Taşıyıcı));
            if (kullanıcı == null) return false;
            return true;
        }
        #endregion

        #region Use Case UC1 : Müşterinin Kargo Eklemesi

        public List<String> KargoEkle()
        {
            kargo = new Kargo((Müşteri)kullanıcı);
            kargo.ID = IDGenerator.GetKargoİşiID();
            return facade.GetAll(typeof(Kargo));
        }

        public void ÜrünOluştur(Boyut boyut, Ağırlık ağırlık, Tür.Türler tür)
        {
            ÜrünDescription üd = kargo.ÜrünOluştur(boyut, ağırlık, tür);
            facade.Put(üd, typeof(ÜrünDescription));
        }

        public void ÜrünEkle(int ürünID, int quantity)
        {
            ÜrünDescription eklenecekÜrünDesc = (ÜrünDescription)facade.Get(ürünID,
typeof(ÜrünDescription));
            kargo.ÜrünEkle(eklenecekÜrünDesc, quantity);
        }

        public void BaşlangıçFiyatıBelirle(Fiyat fiyat)
        {
            Teklif t = kargo.BaşlangıçFiyatıBelirle(fiyat);
            facade.Put(t, typeof(Teklif));
        }

        public string Onay()
        {
            bool isComplete = kargo.Onay();
            if (isComplete)
            {
                facade.Put(kargo, typeof(Kargo));
                return "Onaylandı!";
            }
            else
            {
                return "Onaylanamadı! Bilgilerden biri eksik!";
```

```
            }
        }

        #endregion

        #region Use Case UC2 : Taşıyıcının Teklif Vermesi
        public List<String> AçıkEksiltmeyeKatıl()
        {
            açıkEksiltme = new AçıkEksiltme((Taşıyıcı)kullanıcı);
            return facade.GetAll(typeof(Kargo));
        }

        public void İşSeç(int kargoİşiID)
        {
            Kargo kargoİşi = (Kargo)facade.Get(kargoİşiID, typeof(Kargo));
            açıkEksiltme.İşSeç(kargoİşi);
        }

        public string FiyatTeklifEt(Fiyat fiyat)
        {
            Teklif fiyatTeklifi = açıkEksiltme.FiyatTeklifEt(fiyat);
            if (fiyatTeklifi != null)
            {
                facade.Put(fiyatTeklifi, typeof(Teklif));
                return fiyat + " TL'lik teklif yapıldı.";
            }
            else
            {
                return "Teklif yapılamadı!\n" + fiyat + " TL'lik teklif minimum değerden
fazla!";
            }
        }

        public string AçıkEksiltmeOnayla()
        {
            açıkEksiltme.Onayla();
            if (açıkEksiltme.isCompleted)
            {
                facade.Put(açıkEksiltme, typeof(AçıkEksiltme));
                return "Onaylandı.";
            }
            else
            {
                return "Eksik bir girdi var!";
            }
        }
        #endregion

        #region Use Case UC3 : Müşterinin Açık Eksiltme Değerlendirmesi


        public void TaşıyıcıSeçmeyeKatıl()
        {
            mts = new MüşterininTaşıyıcıyıSeçmesi((Müşteri)kullanıcı);
        }

        public void KargoİşiSeç(int kargoİşiID)
        {
```

```
            this.kargo = facade.Get(kargoİşiID, typeof(Kargo)) as Kargo;
            mts.KargoİşiSeç(kargo);
        }

        public void TeklifSeç(int teklifID)
        {
            Teklif teklif = (Teklif)facade.Get(teklifID, typeof(Teklif));
            Taşıyıcı taşıyıcı = teklif.kişi as Taşıyıcı;
            mts.TaşıyıcıSeç(taşıyıcı);
        }

        public string TaşıyıcıOnayla()
        {
            mts.TaşıyıcıyıOnayla();
            if (mts.isCompleted)
            {
                facade.Put(mts, typeof(MüşterininTaşıyıcıyıSeçmesi));
                return mts.seçilecekTaşıyıcı.Ad + " onaylandı.";
            }
            else
            {
                return "Eksik bilgi var";
            }
        }

        #endregion



        public List<string> KargoİşleriniGetir()
        {
            return facade.GetAll(typeof(Kargo));
        }
    }
}
```

Teklif

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using TestNDAT.Kişiler;
using TestNDAT.ÜrünBilgileri;

namespace TestNDAT
{
    public class Teklif
    {
        public int ID;
        public int kargoişiID;
        public Kişi kişi;
        public Fiyat fiyat;

        public Teklif(int kargoİşiID, Kişi müşteri, Fiyat fiyat)
        {
            this.ID = IDGenerator.GetTeklifID();
            this.kargoişiID = kargoİşiID;
```

```
            this.kişi = müşteri;
            this.fiyat = fiyat;
        }
    }
}
```