[**OOAD ('09 Grubundan alıntıdır.)**](https://www.facebook.com/notes/ege-%C3%BCniversitesi-bilgisayar-m%C3%BChendisli%C4%9Fi-11-ders-notlar%C4%B1/ooad-09-grubundan-al%C4%B1nt%C4%B1d%C4%B1r/748773751821405)

SSS

OOAD Nedir?

Object-oriented analysis and design, yani nesneye-yönelik analiz ve tasarım. Önceki dönemlerde alınan (Cenk hocanın dersiydi herhalde) nesne dersinin devamı. Bu derste amacımız, gerçek hayattaki bir problemi, nesneye yönelik bir tasarıma uygun olarak modellemeyi öğrenmek. Bunun için de belli yöntemler kullanıyoruz, ders bize bu yöntemleri anlatıyor. Bu ders bir programlama dersi değil, ancak tabi nesne nedir bilmezseniz özellikle çalışmalar ilerledikçe işiniz zorlaşacaktır.

OOAD dersi için ödev/devam zorunluluğu falan ne durumda?

 Eğer bu sorunun cevabını bilmiyorsanız, işiniz zor. Devam zorunluluğu yok, ancak hemen her hafta küçük adımlarla ilerletilen bir ödev var. Notlar sene sonunda verilecek, lâkin haftalık ödevlerin geç teslimi not kırılması anlamına geliyor.

Naçizane tavsiyem, özellikle bu sene dersi takip etmenizden yana. Zira sınıftaki tartışmaları takip ediyorsanız, zaten sınava hazırlanmış oluyorsunuz.

Çıkmış sorular var mı?

En azından bir set soru var. Zaten bu sorular kabaca çıkacak soru potansiyelini belirliyor.

İsmail Arvas’ın gruba attığı mesaja göre:

1. “Referandum Tanımlama” kullanım durumunu kitabınızda tanımlanan “fully dressed” formatına uygun olarak yazınız. Bu kullanım durumuna ilişkin Sistem Sıradüzen Diyagramını (SSD) çiziniz. (30 puan)

2. “Referandum Tanımlama” kullanım durumu için alan modelini tanımlayınız. (30 puan)

3. Kitabınızdaki Monopol oyununu oyuncunun bir kareye gelince bu karede bulunan puanı aldığı ve belirli bir puana erişen ilk oyuncunun oyunu kazandığı şekilde tasarlayınız. Dışarıdan “Play Game” etkileşimi geliyor, bu etkileşimden sonra gerçekleşen nesne etkileşimlerini Etkileşim Diyagramı (Interaction Diagram) kullanarak çiziniz. Yararlandığınız “GRASP” desenlerini nesnelerin üzerinde belirtiniz ve gerektiğinde açıklayınız. (40 puan)

Ders notu/kitabı/pdf var mı? Bilgehan’da dersin kitap çıktısı var diye biliyorum. Craig Larman’ın Applying UML and Pattenrs: An Introduction toObject-oriented Analysis and Design and Iterative Development kitabını işliyoruz. Bende 3rd edition PDF’i var, sonraki sürümleri var mı bilgim dahilimde değil.

Ödev hakkında nereden bilgi alabilirim? Sınavda ödevden soru çıkar mı? Evet, muhtemelen çıkar. Bu soruları cevaplarken asistanların da bizimle paralel hazırladığı projeyi baz alacağız. Moodle’da vizyon dökümanı ve Use Case dökümanı bulunmakta. USC dökümanı bize yollanan e-mail’den sonra güncellenmiş, son haline bakmayı unutmayın. Bu doküman domain modelleri de içeriyor.

I. Problemler

Elimizde üç adet “problem” var: NextGen POS System, Monopoly Game ve MenuPlanner (proje).

POS, bir dükkanda gerçekleştirilen satışları kontrol ve kayıt etmek üstüne bir yazılım. Dolayısıyla ürün stokları, satış sırasında oluşturulan “sepetler”, satış gerçekleştirilmesi ve kayıtlardan sorumlu olacak.

Monopoly game, bir oyun simülasyonu. Kuralların detaylarından ziyade, oyun başlayıp sürene kadar ilerlemelerini sağlayan bir yazılım.

MenuPlanner ise, kullanıcılara dengeli beslenme için önerilerde bulunan bir sistem.

Okuyup geçtiğinizi tahmin ettiğim bu konu, aslında en önemli kısım. “O zaman anlatsana” diyeceksiniz, siz de haklısınız. Ama kullandığımız yöntem, modelleme üstüne ve örneklere baktıkça her şey oturacak zaten. Demek istediğim şu: Konu nedir bilirseniz, yöntemleri de anlarsanız, tüm adımları gerçekleştirebilirsiniz, ezberlemekten çok daha kolay olacaktır, hem de finalde de çok daha rahat hazırlanırsınız. Hem de tabi pragmatizmi bir yana koyarsak, konuyu anlamış, dersin hakkını vermiş olursunuz.

II. Modelleme

Problemi kabaca tanımladıktan sonra, modellemeye geçiyoruz. Modelledikçe daha iyi anlıyoruz, anladıkça çözüyoruz, çözdükçe daha iyi modelliyoruz, yeterince iyi kurmayı başarırsak sonunda şirinleri bile görüyoruz.

Adım 1. Use Case

Modellemenin ilk adımlarında, elimizdeki tek şey, olayın ne olduğunu biliyor olmamız. Bu yüzden olayları düşünüyor, belli durumlarda olacak olaylar dizilerini kurgulayıp yazıyoruz. Bu kurgulara Use Case deniyor.

Use Case terimleri

Aktör: Davranışa sahip öğelere verilen isim. Bir insan olabileceği gibi bir bilgisayar sistemi vs de olabilir. (Örn: Kasiyer, oyuncu, kullanıcı)

Senaryo: Yazılımın kullanımındaki belli bir süreci ifade eden terim. Her use case, içinde en az bir senaryo içerir. (Örn: “başarıyla satış gerçekleştirme”, “satış gerçekleşirken sistemin çökmesi”)

Use Case: Bir ya da daha fazla senaryoyu barındıran, yazılım kullanımındaki bir sürecin genel olarak alınmış hâli. (Örn: “Satış gerçekleştir”, “öğün ekle”)

Use Case yazım türleri

Brief: Genelde tek paragraflık, özet anlatım.

Casual: Yan senaryoları da anlatan birkaç paragraflık, formal olmayan stil.

Fully dressed:”Ana” dökümantasyon tarzı. Formatı şöyledir

Use Case Name: USC ismi

Scope: Tasarlanan sistemin ismi

Level: User-goal ya da subfunction (çoğu zaman ilki olacak bizim örneklerde)

Primary Actor: Sistemi çağıran aktör

Stakeholders and Interests: Bu use case’de “işi görülen” aktörler ve istedikleri sonuçlar

Preconditions: Başlamadan önce gerçekleştirilmiş olması gereken koşullar

Success Guarantee/Postconditions: Bittiğinde gerçekleşmiş olması gereken durumlar

Main success Scenario: Murphy işlere karışmazsa neler olacağını anlatan olaylar dizisi

Extensions/Alternate Scenarios: Murphy’nin burnu işin içine girince olacakları anlatan diziler

Special Requirements: İşlevsel olmayan gereksinimler

Technology and Data Variatons List: Gerekli I/O ve Data formatları

Frequency of Occurence: Ne sıklıkla kullanılacağı

Miscellaneous: İlgili diğer konular

Adım 2. Domain Model

Domain model, elimizdeki konuyu gerçek dünyaya ait öğelerle oluşturup açıkladığımız modeldir. Programlama kısmına geçerken şüphesiz birçok öğe kullanılacaktır, ancak bu noktada düşüncemiz programlama kısmı değil, bahsi geçtiği gibi MADAŞ, yani “modelle-anla-daha iyi modelle-anladıkça anla-şirinleri gör” prensibidir.

Use Case’ler için Domain Modeller hazırlanır, daha sonra bunlar birleşerek büyük resmi (Unified Domain Model) oluşturur.

Adım 3.  System Sequence Diagram (SSD)

SSD, belli bir senaryo sırasında gerçekleşen aktör-sistem arası event akış diyagramıdır. Aktör’den sisteme giden oklar, sistem event mesajlarıdır. Kesik çizgiler bunlara cevap mesajlarıdır. Loop’lar kutuya alınarak gösterilebilir.

Adım 4. Interaction Diagram

Interaction Diagramlar, sistem akışını anlatan diyagramlardır. Genelde alt tür ismiyle anıyoruz bunları.İki türü vardır:

1. Sequence Diagram

Daha çok kullanmamız olası olan, olaylar akışı şeklinde gösterilen, SSD’yi andıran formata sahip diyagram. Notasyonu comm’a göre daha kuvvetli ve katı olduğu için avantaja sahiptir.

2. Communication Diagram

Bu diyagram daha farklı bir format kullanır. Burada olaylar gerçekleştikçe diyagram uzar. Oklarla her tarafa gidebilen bir yapısı var, şakalı komikli bir format kendisi. Yerden falan kazandırır ve tabi belli durumlarda avantajlıdır. Ancak örneklerimizde sequence daha rahat ve daha az kafa karıştırıcıymış gibi geliyor. Yine de kabaca ne olduğunu bilmekte fayda var.

III. GRASP

Wikipedia’dan az önce okuduğuma göre General Responsibility Assignment Software Patterns diye, büyük çabalarla İngilizce bir kelimeye dönmüş bu kısaltmanın Türkçesi “Genel Sorumluluk Atama Yazılım Örgüsü gibi bişey oluyor. GSAYÖ demesi çok rahat olmadığından GRASP diyelim.

Bu patternler, tasarımımızı yaparken göz önünde bulunduracağımız rehberler. Hepsi belli tasarım sorularına cevap gibi de düşünülebilir (kitaptan çaldım). Bizi kötü tasarımdan koruyan yazılım melekleri, bunları sevip korumamız gerekiyor. İyi de anlamak gerekiyor. Rehber olduklarını tekrardan hatırlatmak isterim. Zaman zaman çeliştikleri de oluyor. Kesin gibi düşünmektense, “mümkün şartlarda uyulması gereken kurallar” gibi düşünmek daha doğru olabilir.

Şu noktada bizi ilgilendirenler şunlar:

Creator

Yeni sınıfları kim yaratacak?

Bu soru bazen şaşırtmacalı olabiliyor. Genelde eğer bir sınıf, diğerini kapsıyorsa, “kaydediyorsa”, içeriyorsa ya da kullanıyorsa, diğer sınıfın yaratıcısı olması iyi bir fikir olabilir. Örneğin POS’ta bir satışa ait her bir eleman, saleslineitem olarak geçiyor. Yani Sale=Sepet, SalesLineItem=2 adet Kalem gibi bir şey. Sale, salesLineItem’ı içerdiği için, salesLineItem’ları Sale sınıfı yaratsın dememiz mantıklı görünüyor.

Creator genelde low coupling’i de getirecektir. Sonuçta zaten ilgili iki sınıfı bağlayınca, muhtemelen gereksiz bir bağ kurmaktan kurtulmuş oluyoruz..

Information Expert/Expert

Sorumlulukları neye gore, kime vereceğiz?

Information Expert diyor ki, bir işleme ait bilgiler kimdeyse, o yapsın. Tabi tersten de düşünülebilir, yani eğer bir işi yapacak sınıf daha belli gibiyse ve diğer prensiplere ters düşmeyecekse, bilgileri o sınıfın tutmasını isteyebiliriz. Genel olarak diyoruz ki “Herkes bildiği işi yapsın”.

Üstteki örnekten gidersek, Sale, Expert’e göre bir satışın toplam ücretini hesaplama sorumluluğunu almalıdır. Çünkü salesLineItem’ları o tutar (SLI diyeceğim bundan sonra). Ancak her bir ürünün alt-toplamını SLI’lar tutmalıdır, çünkü ürün bilgilerini ve ürün sayısını tutan SLI’dır.

Info Expert genelde High Cohesion ve Low Coupling’i destekler, zira hem odaklanmanın artmış olması, hem de yeni bir bağ eklenmemiş olması olasıdır.

Controller

Sistem operasyonlarını UI katmanından sonra alan ve kordine eden (kontrol eden) öğe hangisidir?

Bu iş için genelde bir ya da daha fazla sistem atıyoruz. Mesela “Sale” işlemlerinin tümünde controller görevi yapacak bir SaleController olacağı gibi, “SystemController” diye tüm sistemi kapsayan bir controller da olabilir. Diğer patternleri göz önünde bulundurup, sistemin genişliğine göre karar vermek önemli

Low Coupling

Sistemin değişikliklerden az etkilenmesini, düşük bağımlılığa ve yüksek yeniden kullanılırlık değerine sahip olmasını nasıl sağlarız?

Coupling (aklına dizi gelenler?) kelime anlamı olarak eşleşme demek. Sistemimizde düşük eşleşme istiyoruz. Niye?

Bir sistem hayal edin. Mesela Controller’dan gidelim. Sistemdeki TÜM UI tek bir controller’a bağlı olsun. Adına da SysC diyelim. SysC haliyle birçok sınıfa bağlı olacak, mesaj alışverişinde bulunacaklar.

Peki SysC’yi değiştirirsek? Hop, tüm sınıfları bir elden geçirmek gerekebilir. Ya da daha kötüsü, alt sistemlerden birisini değiştirdik ve SysC’de bir değişiklik yapmamız gerekti. Hop, zincirleme olarak bu sefer SysC’deki değişiklikten dolayı öbür sistemleri de değiştirmek gerekti. Bir sürü iş.

Low Coupling diyor ki, herkesi sadece işi olan sınıflarla eşleyin ve bu eşleşmeleri mümkün surette azaltın, böylece sisteminiz daha stabil, daha kolay kullanılan ve kontrol edilebilen bir hâl alır.

High Cohesion

Nesnelerin odaklanmış ve anlaşılabilir olmasını nasıl sağlayabiliriz

Low Coupling’in büyük kankası olan High Cohesion, yüksek yoğunluk anlamına geliyor. Burada yoğunluktan kasıt, odaklanmak diyebiliriz. Kabaca, bir sınıfa çok fazla görev vermemek gerekiyor. Verilen görevlerin de birbiriyle ilgili olması gerekiyor. Özelleştiriyoruz yani. Bir sınıf SaleLineItem’ları yaratıyor, kullanıcı isimlerini tutuyor ve satış sonunda bilgileri database’e atıyorsa, yanlış bir şeyler var demektir.

Fark ettiğiniz üzere Low Coupling ve High Cohesion, diğer üçü gibi sınıflara verilen sorumluluklardan ziyade, genel tasarım prensipleri. O yüzden sınavda ilk üçü için, ilgili sorumluluğa ait sınıftan ok çıkıp “bunu creator’a göre seçtim, çünkü öbürünü kapsıyor” diye açıklarken, LC ve HC’yi ya bu notların içinde ya da ayrıca açıklamanız gerekecektir.