***Bil. Müh. Tasarım Kısa Özet 02/12/2021***

*ö.c.s*

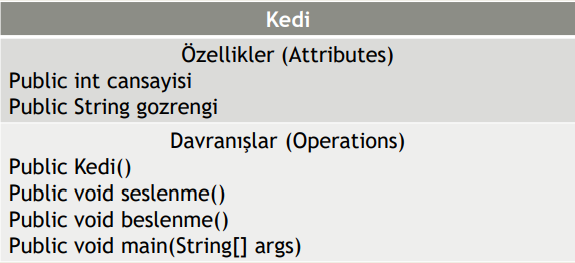
***Not:*** *UE sisteminde tüm haftaların yer aldığı bir .pdf bulunmaktadır. Bu özet tüm konulara hızlıca bir göz atabilmek için hazırlanmıştır. Slaytta yer alan hiçbir kod burada yer almamaktadır. Ayrıca bazı önemli bilgiler gözden kaçmış olabilir bu nedenle bir sorumluluk kabul etmemekteyim. Sınav öncesi konu başlıklarına bir göz atabilmek için kendime hazırladım, ihtiyacı olan yararlanabilir.*

***Hafta 1: Nesne Yönelimli Analiz ve Tasarım***

**Nesneye yönelik düşünme,** eldeki sorunları veya kavramları incelemeyi, onları parçalara ayırmayı ve bunları nesneler olarak düşünmeyi içerir. Kodunuzdaki şeyleri temsil etmek için nesneleri kullandığınızda kodunuz **düzenli** (organized), **esnek** (flexible) **ve yeniden kullanılabilir** (reusable) olur. Nesne cansız olsa bile davranışının ve özelliklerinin farkındadır.

Sınıf ve Nesne Nedir?

Sınıf nesnelerin özelliklerini, davranışlarını belirten şablonlardır. Nesne ise o sınıfın bir örneğidir. Sabun fabrikasında sabunların şablonu sınıf, sabunlar ise nesnedir.



Şekil 1: UML Diyagramı

**Yazılım tasarımı**: bir müşterinin isteklerini ve gereksinimlerini; istikrarlı, sürdürülebilir, geliştirilebilen ve daha büyük bir sistemin parçası olabilecek çalışma koduna dönüştürme işlemidir.

**Yazılım mimarının** temel amacı müşterinin ihtiyacına, müşterinin sahip olduğu bütçe dahilinde hizmet etmektir. Tüm sistemin ve sistem içerisindeki ilişkilerin belirlenmesinden sorumludur.

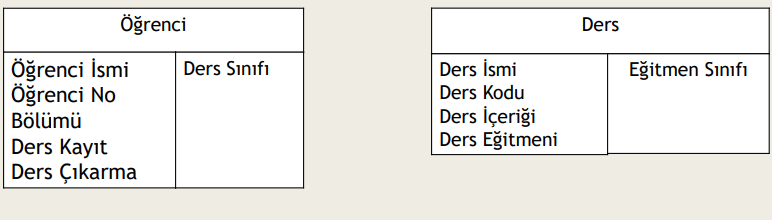
Müşteri (client) veya kullanıcı isteğine (user request) bağlı olarak bir üründe uygulanması gereken şartlara **Gereksinim** denir. Müşterinin tam olarak ne istediğini doğru bir şekilde anlamak gerekir. Gereksinimlerden sonraki adım kavramsal ve teknik tasarımdır.

**Kavramsal tasarım**da, geliştirilmekte olan yazılımın bileşenleri, bağlantıları ve sorumlulukları ana hatlarıyla belirtilir. **Teknik tasarım** bu sorumlulukların nasıl yerine getirildiğini açıklamayı amaçlar.

1. Gereksinimleri alın ve problemleri analiz edin.
2. Çözümü tanımlamak için bir algoritma oluşturun.
3. Algoritmayı sözde kod veya akış diyagramı ile gösterin.
4. Oluşturduğunuz algoritmayı koda çevirin.
5. Kodunuzu test edin

İyi bir tasarım güvenliği kolaylık ve performansla dengelemelidir. Bağlam, tasarımdaki kalitenin dengesine karar verirken önemli bilgiler sağlar. Yazılım tasarımı sonuçları da dikkate almalıdır. Maliyet, zaman ve insan gücü gibi kaynakların dengeli tasarlanması gerekir.

CRC (Class Responsibility Collaborator) kartları ile bileşenleri, sorumlulukları ve bağlantıları yüksek düzeyde temsil edebiliriz. CRC kartları üç bölümden oluşur, sınıf adı, sınıfın sorumlulukları ve ortak çalışanlar.



Şekil 2: Öğrenci ve Ders CRC kartı

CRC Kartlarının Avantajları:

* Karmaşık bir sistemi kolayca anlamamızı sağlar.
* CRC kartları basit bir tekniktir ve pahalı bilgi işlem kaynakları gerektirmez.
* Bir takımın tasarım yaparken beyin fırtınası yapmasını sağlar.
* Diğer resmi nesne yönelimli tasarım metodolojileriyle kullanılabilir

***Hafta 2: Nesne Yönelimli Modelleme***

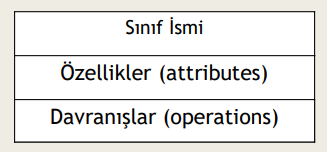
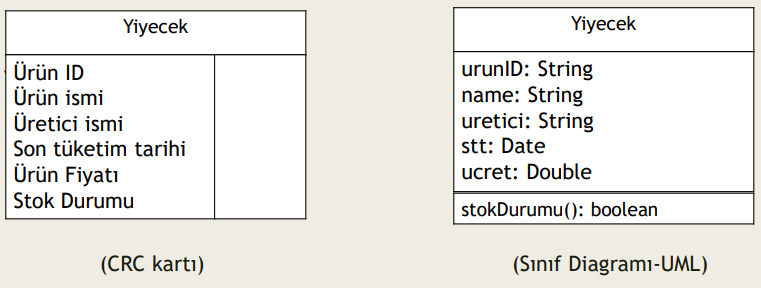
Tasarım adımı, gereksinimlerinizi anlamak ve ürünü oluşturmak arasındadır. Yinelemeli olarak hem problem alanı hem de çözüm alanı ile ilgilenir. **Yazılım tasarımı sırasında amaç**, yazılımın tüm nesnelerinin “modellerini” oluşturmak ve geliştirmektir. Yazılım modelleri, nesneler için tasarım sürecini anlamanıza ve düzenlemenize yardımcı olur ve genellikle UML ile ifade edilir. Nesne yönelimli tasarım iki adımdan-kısımdan oluşur:

* 1. Kavramsal tasarım: problemdeki temel nesneleri tanımlamak için nesne yönelimli analiz kullanır ve sorunu yönetilebilir parçalara ayırır.
* 2. Teknik tasarım, nitelik ve davranışları dahil nesnelerin ayrıntılarını daha da hassaslaştırmak için nesne yönelimli tasarım kullanır, bu nedenle geliştiricilerin çalışan yazılım olarak kullanması için yeterince açıktır.

***Hafta 3: Tasarım Prensipleri***

**Soyutlama:** bir kavramı, önemsiz ayrıntıları göz ardı eden ve kavram için gerekli olan temel unsurları vurgulayan basitleştirilmiş bir tanımlamaya bölmektir. Bir soyutlamanın temel özellikleri (characteristics) iki yolla anlaşılabilir:

1. **Temel Özellikler:** zaman içinde kaybolmayan özelliklerdir. Değerleri değişebilmesine rağmen, özniteliklerin kendileri değişmez.
2. **Temel Davranışlar:** Soyutlama da bir kavramın temel davranışlarını açıklar. Bir aslan nesnesinin avlanma, yemek yeme ve uyku gibi temel davranışları olabilir.



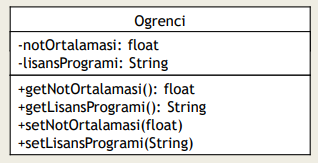
Şekil 3: UML ile Örnek Soyutlama Tasarımı

İkinci tasarım prensibi ***iyileştirme***dir. Bu prensibin arkasında üç önemli fikir vardır:

* 1) **Paketleme (bundle) özelliği**: Değerleri ve davranışları kendi kendine yetebilen bir nesneye dönüştürebilen (self-contained object) paketleme yeteneğidir.
* 2) **Açığa çıkarma (expose) özelliği:** Bir nesnenin belirli bir verisini ya da fonksiyonunu bir arayüz (interface) ile diğer nesnelerden erişilebilecek şekilde “açığa çıkarma” yeteneğidir.
* 3) **Kısıtlama (restrict) özelliği:** Sadece nesne içindeki belirli fonksiyonlara ve verilere kısıtlı erişme yeteneğidir

Soyutlama ilkesi, belirli bir bağlamda bir kavramla ilgili hangi niteliklerin ve davranışların belirleneceğine yardımcı olur**. Kapsülleme ilkesi** bunu bir adım daha ileri götürür ve bu özelliklerin aynı sınıfta bir araya getirilmesini (paketlenmesini-bundled) sağlar. Kapsülleme tasarım prensibi üç fikir içerir:

* Verileri işleyen (değiştiren) veriler ve fonksiyonlar bir self-contained içerisine paketlenir (bundle).
* Nesnenin verileri ve fonksiyonları diğer nesneler tarafından erişilebilir hale getirilebilir (expose).
* Nesnenin verileri ve fonksiyonları yalnızca nesnenin içinde sınırlandırılır. (restrict).



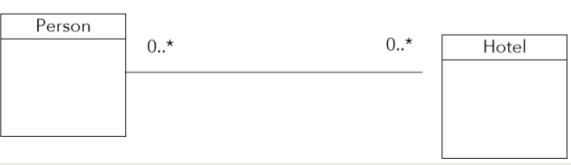
Şekil 4: UML içerisinde erişim (+) ve kısıtlama (-)

***Hafta 4:***

**Ayrıştırma** tasarım ilkesi bir bütün alır ve farklı bölümlere ayırır. Aynı zamanda tersini yapar ve farklı fonksiyonlara sahip ayrı parçalar alır ve bir bütün oluşturmak için bunları birleştirir. Üç tür ilişki vardır:

* **Birliktelik (Association):** İki nesne arasında her zaman bir bağlantı vardır.
* **Münasebet (Aggregation):**Parça-bütün (whole-part) ilişki vardır. Ancak biri diğersiz yaşayabilir.
* **Oluşum (Composition):** Parça-bütün ilişkisi vardır. Ancak biri diğerisiz yaşayamaz.

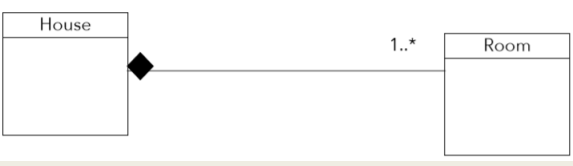
Birliktelik (Association): İki nesne arasında, bir süre birbirleriyle etkileşime girebilecek gevşek bir ilişki olduğunu gösterir. Birbirlerine bağımlı değillerdir ve bir nesne imha edilirse, diğeri var olmaya devam edebilir. Örnek: bir müşteri ve otel olabilir. İki sınıf arasındaki ilişkidir.



Münasebet (Aggregation): Parçalar tüme ait olsa da, bağımsız olarak da var olabilirler. Örneğin yolcu uçağı ve kabin ekibi… Bir bütün sınıf daha küçük sınıflarla ilişkilidir.



Oluşum (Composition): Bir bütün, parçaları olmadan var olamaz ve bütün, eğer yok edilirse, o zaman, parçalar da yok edilir. Örneğin ev ve oda ilişkisi eğer ev yok olursa odalar da yok olur. Bütün parçaların oluşumundan veya yok olmasından ilişkilidir.



**Genelleştirme** tasarım ilkesi, iki veya daha fazla sınıf arasında tekrarlanan, ortak veya paylaşılan özellikleri alır ve bunları daha genel başka bir sınıfa dönüştürür. Böylece kod tekrar kullanılabilir. **Özelleşme** ise genelleme işleminin tersi bir süreçtir, mevcut bir sınıftan yeni alt sınıflar oluşturmak demektir.

Public: Her yerden erişilebilir.

Private: Sadece sınıf içinden erişilebilir.

Protected: Sınıf içinde ve türetilen sınıflar içinden erişilebilir.

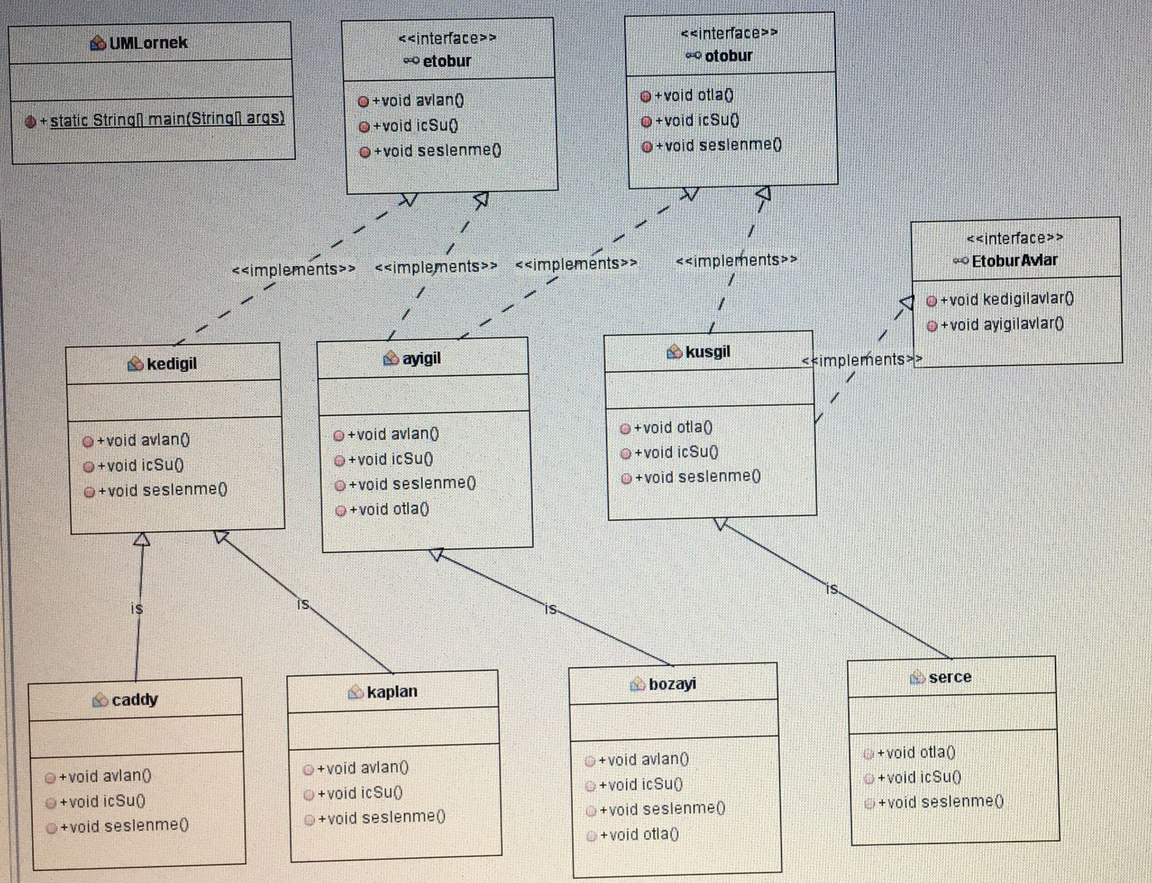
Hafta 5:

**Kalıtım (Inheritance),** ilgili sınıfların tek bir üst sınıfa genelleştirilmesine izin verir ve yine de alt sınıfların aynı nitelik ve davranış kümesini korumasını sağlar. Bu koddaki fazlalığı kaldırır ve değişikliklerin uygulanmasını kolaylaştırır. Java da yalnızca bir uygulama kalıtımına izin verilir. C++ çoklu kalıtımı destekler.

Java, **arayüz (interface)** kalıtımını sunarak tek uygulama kalıtımı sınırlamasını çözer. Bir arayüzü kullanmak için “**implements**” kelimesini kullanırız. Ara yüz yalnızca metodu belirtir (yapılandırıcısı, metot gövdesi ve nitelikleri olmadan). Bir alt sınıf ise bir üst sınıfı **extends** ederek üst sınıfın uygulama ayrıntılarını da devir alabilir.

**Soru?** 1. Etobur ve Otobur diye iki arayüzümüz olacak. 2. Kedigil, Ayigil ve Kusgiller diye soyut sınıflarımız olacak ve diğer somut sınıflar bu sınıflardan türeyecekler. 3. Caddy, Kaplan nesneleri Kedigillerden türeyecek. Ve dolayısıyla Etobur olacaklar ve avlanma yetenekleri olacak. 4. BozAyi nesnesi Ayigil soyut sınıfından türeyecek. Ayigil soyut nesnesi de iki ayrı arayüzü uyguladığı için (interface, implements) iki farklı yeteneğe sahip olacak. Yani hem otçul hem etçil olacak. 5. Serce nesnemiz de Kusgiller soyut sınıfından tureyecek ve Kusgiller sınıfı da Otoburu uyguladığı için (implements) Otobur olacak ve Etobur hayvanlar tarafından avlanabilecek.

**Çözüm (Hata İçerebilir!)**



İyi bir tasarımda, modüller birbiriyle uyumludur ve bu nedenle kolayca bağlanabilir ve yeniden kullanılabilir. Tasarım karmaşıklığını değerlendirmek için iki ölçüt kullanılır: **Coupling (bağlama)** ve **cohesion (yapışma).**

**Coupling (Bağlama):** Bir modül ile diğer modüller arasındaki karmaşıklığa odaklanır. Bir modül diğer modüllere çok fazla güveniyorsa, diğerlerine "*sıkıca bağlanır*" ve bu kötü bir tasarımdır.

Bir modül iyi tanımlanmış arayüzler üzerinden diğer modüllere kolayca bağlanıyorsa, diğerlerine “*gevşek bir şekilde bağlanır*” ve bu iyi bir tasarımdır. İyi bir tasarım:

* **Derece:** Bağlantı sayısıdır. Bir modül diğer modüllere sadece birkaç parametre ile gevşek bir şekilde bağlanabilmeli
* **Kolaylık:** Diğer modüllerin uygulamalarını anlamaya gerek kalmadan kolayca bağlanabilmeli
* **Esneklik:** Gelecekteki bağlamalar için modüller kolayca değiştirilebilinmeli

Tam tersi durumlarda sistemin sıkıca bağlandığını ve kötü bir tasarıma sahip old. düşünülür.

**Cohesion (Yapışma)**: Bir modülün sorumluluklarının netliğini gösterir ve karmaşıklığa odaklanır. Bir görevi yerine getiren ve başka hiçbir şey yapmayan veya açık bir amacı olan bir modülün **yüksek yapışması** (high cohesion) vardır. Bu durum iyi bir tasarımdır.

Öte yandan, eğer bir modül birden fazla amacı kapsıyorsa, bir yöntemi anlamak için bir kapsüllemenin kırılması gerekiyorsa veya modülün net olmayan bir amacı varsa, **düşük yapışma** (low cohesion) vardır. Birden fazla sorumluluğu olan modülü bölmek iyi bir fikirdir.

Bir yazılım sisteminin her bir bileşeninin başka her şeyi bilmesi gerekmez. Modüller sadece işlerini yapmak için ihtiyaç duydukları bilgiye erişebilmelidir. **Bilgi gizleme**, geliştiricilerin, üzerinde çalıştıkları modülün uygulama ayrıntılarını bilmelerine gerek kalmadan, modüller üzerinde ayrı ayrı çalışmalarını sağlar. Java’da 4 seviye ile bilgi gizleme sağlanabilir: • Public • Protected • Default • Private

***Hafta 6: Tasarım Kalıpları***

**Tasarım kalıpları,** programlama dillerinden bağımsız olarak, şimdiye kadar en iyi tasarımları bir arada toplayan ve gruplayan bir platformdur. Karşılaşılan sorunlara bütün ve standart tasarımlar sunarak ortak bir tasarım dili oluştururlar. Tasarım kalıbı ifadesini ilk olarak ***Christopher Alexander***isimli bir mimar kullanmıştır.

Aynı tasarım kalıbını defalarca uygulasanız bile hiçbir iki uygulamanız birbirinin aynısı olmayacaktır. Yani tasarım kalıpları yüksek cohesion (yapışma) ve düşük coupling (bağlama) yapıları kurgulamamıza yardımcı olur.

■ Tasarım kalıplarındaki genel kural : “Değişkenlik gösterecek tüm özellik ve davranışları bir araya getir ve ayrı kalıplarda tut!”

**Strateji Tasarım Kalıpları:** Değişkenlik gösteren algoritmaları ve davranışları bir araya getirip (encapsulation) algoritmaları geliştirebilir hale getirmeli. Böylece arayüzün arkasına toplanan ve ileride geliştirilmeye açık davranışların kontrolsüz değiştirilmesinin önüne geçilir.

Şimdiye kadar öğrendiğimiz tasarım prensipleri:

* 1. Değişkenlik gösterecek davranışların bir araya getir,
* 2. Kalıtım yerine kompozisyon kullan
* 3. Uygulamaya program yazmak yerine, arayüze yazılım yap (programming to interface)

**NOT: 6. Haftanın Slaytında Çok Fazla Kod ve Kedi Simülasyonu isimli bir çalışma vardır. Lüften göz Atınız.**

***Hafta 8:***

**Tasarım Kalıplarının Temel Kuralları:**

1. Uygulamaya değil, arayüze program yazılmalı.
2. Nesneler arasında esnek bağ oluşturulmalı, somut bağ olmamalı.
3. Değişebilir bölümler tespit edilmeli ve bir çatı altında tutulmalı
4. Kalıtım yerine kompozisyon kullanılmalı
5. Tasarım geliştirmeye açık, değişime kapalı olmalı
6. Somut sınıflar yerine soyut sınıflar kullanılmalı

**Gözlemci Tasarım Kalıbı:** Gözlemci tasarım kalıbında, aboneler, yayınlanan bilgileri otomatik olarak alırlar. Her abone tek tek bilginin alınması için çaba sarf etmez. 1-n ilişkisi vardır. (1: yayını yapanlar, n: bu yayına abone olanlar)

**Medyatör Tasarım Kalıbı:** Nesnelerin birbirleri ile iletişimininin koordinasyonunu yönetebilen tasarım kalıbı Medyatör Tasarım Kalıbıdır. Gözlemci kalıbında n sayıda nesnenin birbirleri ile iletişim problemi yaşayabilme ihtimaline karşı, Medyatör tasarım kalıbı ile birlikte kullanıldığında başarılı sonuçlar elde edilmektedir.

**Dekoratör Tasarım Kalıbı:** Sitemize decorator tasarım kalıbı ile her türlü konfigürasyonu çok kolay bir şekilde yapabilme olanağı veriyoruz. Kalıtım yönteminin tersine, konfigürasyonu belirleme işini derleme esnasında değil de çalıştırma esnasında yapıyoruz bu sayede istediğimiz değişikliği rahatlıkla yapabiliyoruz.

**Yalnızlık-Tekil Tasarım Kalıbı:** Bir nesnenin sadece ve sadece bir kez yaratılmasını garanti sağlar. Birden fazla ihtiyaç duyulması durumunda, daha önceden yaratılmış olan nesnenin kullanılmasını sağlar. **Volatile,** sadece değişkenlere uygulanır ve volatile ile değişkenin sekronize edilmiş çoklu işleme adandığını ve derleyicinin bu değişkeni optimize etmesi için uğraşmaması gerektiğini belirtiyoruz.

***Hafta 10***

1994 yılında Gang of Four (4’lü çete) olarak adlandırılan Eric Gamma, Richard Helm, Ralph Johnson ve John Vlissides ünlü “Tasarım Kalıpları (Design Patterns) kitabını yazmışlardır.

*4’Lü Çete (Gang of Four) Tasarım Kalıpları:*

- **Creational Kalıplar (Yaratımsal):** Yazılım sistemindeki nesnelerin yaratılışı ve yönetilmesi hakkında yol gösterirler.

*Singleton:* Uygulamanın yaşam süresince bir nesnenin bir kez oluşturulmasını sağlar.

- **Structural Kalıplar (Yapısal):** Yazılım sistemindeki nesnelerin birbirleriyle olan ilişkilerini gösteren kalıptır.

*Decorator:* Bir yapıya dinamik olarak yeni metotlar eklenmesini düzenler.

- **Behavioral Kalıplar (Davranışsal):** Nesne davranışlarını takip eden kalıptır.

*Observer:* Bir nesnede meydana gelen değişikliklerde içinde bulundurduğu listede bulunan nesnelere haber gönderen tasarım desenidir.

*Mediator:* Çalışmaları birbirleri ile aynı arayüzden türeyen nesnelerin durumlarına bağlı olan nesnelerin davranışlarını düzenler.

*Strategy*: : Bir işlemin birden fazla şekilde gerçekleştirile bilineceği durumları düzenler.

**NOT: 5+7+11 OLMAK ÜZERE TOPLAM 23 TASARIM KALIBI VARDIR. DİĞER TASARIM KALIPLARI SLAYTTA YER ALMAKTADIR.**

Kaynak: BILGISAYAR MUHENDISLIĞI TASARIM Dr.Öğr.Üyesi Betül AY (BMT-TumHaftalar-Sunum.pdf)