

CEN206 Nesne Yönelimli Programlama

Hafta-10 (Nesne Yönelimli Tasarım Desenleri - İleri Kavramlar)

Bahar Dönemi, 2024-2025

İndir [BELGE-PDF](#), [BELGE-DOCX](#), [SLAYT](#), [PPTX](#)



Nesne Yönelimli Tasarım Desenleri - İleri Kavramlar

Ana Hatlar

- Daha Fazla Yaratımsal Desenler
- Daha Fazla Yapısal Desenler
- Daha Fazla Davranışsal Desenler
- Anti-Desenler
- Tasarım Deseni Seçim Kriterleri

Tekil (Singleton) Deseni

Bir sınıfın yalnızca bir örneğinin olmasını sağlar ve buna global bir erişim noktası sunar.

```
public class Tekil {
    // Özel statik örnek
    private static Tekil ornek;

    // Örneklemeyi engellemek için özel yapıcı
    private Tekil() {}

    // Örneği almak için genel metod
    public static Tekil ornekAl() {
        if (ornek == null) {
            ornek = new Tekil();
        }
        return ornek;
    }

    // İş parçacığı güvenli versiyon
    public static synchronized Tekil isParçacığıGuvenliOrnekAl() {
        if (ornek == null) {
            ornek = new Tekil();
        }
        return ornek;
    }

    // Çift kontrollü kilitleme
    public static Tekil çiftKontrollüOrnekAl() {
        if (ornek == null) {
            synchronized (Tekil.class) {
                if (ornek == null) {
                    ornek = new Tekil();
                }
            }
        }
        return ornek;
    }
}
```

Karmaşık bir nesnenin yapısını, gösteriminden ayırır ve aynı oluşturma sürecinin farklı gösterimler oluşturmaya izin verir.

```
// Ürün
class Pizza {
    private String hamur;
    private String sos;
    private String malzeme;

    public void hamurAyarla(String hamur) { this.hamur = hamur; }
    public void sosAyarla(String sos) { this.sos = sos; }
    public void malzemeAyarla(String malzeme) { this.malzeme = malzeme; }

    @Override
    public String toString() {
        return hamur + " hamurlu, " + sos + " soslu ve " + malzeme + " malzemeli pizza";
    }
}

// Soyut İnşaatçı
abstract class PizzaInsaatcisi {
    protected Pizza pizza;

    public Pizza getPizza() { return pizza; }
    public void yeniPizzaOlustur() { pizza = new Pizza(); }

    public abstract void hamurInsa();
    public abstract void sosInsa();
    public abstract void malzemeInsa();
}

// Somut İnşaatçı
class HawaiianPizzaInsaatcisi extends PizzaInsaatcisi {
    public void hamurInsa() { pizza.hamurAyarla("çapraz"); }
    public void sosInsa() { pizza.sosAyarla("hafif"); }
    public void malzemeInsa() { pizza.malzemeAyarla("jambon ve ananas"); }
}

// Yönetici
class Asci {
    private PizzaInsaatcisi pizzaInsaatcisi;

    public void pizzaInsaatcisiAyarla(PizzaInsaatcisi pizzaInsaatcisi) {
        this.pizzaInsaatcisi = pizzaInsaatcisi;
    }

    public Pizza getPizza() { return pizzaInsaatcisi.getPizza(); }

    public void pizzaInsa() {
        pizzaInsaatcisi.yeniPizzaOlustur();
        pizzaInsaatcisi.hamurInsa();
        pizzaInsaatcisi.sosInsa();
        pizzaInsaatcisi.malzemeInsa();
    }
}
```

Adaptör (Adapter) Deseni

Uyumsuz arayüzlerin birlikte çalışmasını sağlar; bir sınıfın örneğini, başka bir sınıfın arayüzüne uyan bir adaptöre sararak.

```
// Hedef arayüz
interface MedyaOynatici {
    void oynat(String sestipi, String dosyaAdı);
}

// Uyarlanacak arayüz
interface GelismisMediaOynatici {
    void vlcOynat(String dosyaAdı);
    void mp4Oynat(String dosyaAdı);
}

// Somut Uyarlanacak
class VlcOynatici implements GelismisMediaOynatici {
    @Override
    public void vlcOynat(String dosyaAdı) {
        System.out.println("Vlc dosyasını oynatıyor: " + dosyaAdı);
    }

    @Override
    public void mp4Oynat(String dosyaAdı) {
        // Hiçbir şey yapma
    }
}

// Adaptör
class MedyaAdaptor implements MedyaOynatici {
    private GelismisMediaOynatici gelismisMediaOynatici;

    public MedyaAdaptor(String sestipi) {
        if (sestipi.equalsIgnoreCase("vlc")) {
            gelismisMediaOynatici = new VlcOynatici();
        }
        // Gerekirse başka oynaticılar ekle
    }

    @Override
    public void oynat(String sestipi, String dosyaAdı) {
        if (sestipi.equalsIgnoreCase("vlc")) {
            gelismisMediaOynatici.vlcOynat(dosyaAdı);
        }
    }
}
```

```
// İstemci
class SesOynatici implements MedyaOynatici {
    private MedyaAdaptor medyaAdaptor;

    @Override
    public void oynat(String sestipi, String dosyaAdı) {
        // Mp3 için dahili destek
        if (sestipi.equalsIgnoreCase("mp3")) {
            System.out.println("Mp3 dosyasını oynatıyor: " + dosyaAdı);
        }
        // MediaAdapter diğer formatlar için destek sağlar
        else if (sestipi.equalsIgnoreCase("vlc") || sestipi.equalsIgnoreCase("mp4")) {
            medyaAdaptor = new MedyaAdaptor(sestipi);
            medyaAdaptor.oynat(sestipi, dosyaAdı);
        } else {
            System.out.println("Geçersiz medya tipi: " + sestipi);
        }
    }
}
```

Dekorator (Decorator) Deseni

CEN206 Nesne Yönelimli Programlama

Bir nesneye dinamik olarak ek sorumluluklar ekler. Dekoratorlar, işlevselliği genişletmek için alt sınıflandırmaya esnek bir alternatif sağlar.

```
// Bileşen arayüzü
interface Kahve {
    double maliyet();
    String tanim();
}

// Somut Bileşen
class BasitKahve implements Kahve {
    @Override
    public double maliyet() {
        return 1.0;
    }

    @Override
    public String tanim() {
        return "Basit kahve";
    }
}

// Soyut Dekorator
abstract class KahveDekorator implements Kahve {
    protected final Kahve dekoreEdilenKahve;

    public KahveDekorator(Kahve kahve) {
        this.dekoreEdilenKahve = kahve;
    }

    @Override
    public double maliyet() {
        return dekoreEdilenKahve.maliyet();
    }

    @Override
    public String tanim() {
        return dekoreEdilenKahve.tanim();
    }
}
```

```
// Somut Dekorator
class SutDekorator extends KahveDekorator {
    public SutDekorator(Kahve kahve) {
        super(kahve);
    }

    @Override
    public double maliyet() {
        return super.maliyet() + 0.5;
    }

    @Override
    public String tanim() {
        return super.tanim() + ", sütlü";
    }
}

// Kullanım
// Kahve kahvem = new BasitKahve();
// kahvem = new SutDekorator(kahvem);
// System.out.println(kahvem.tanim() + " " + kahvem.maliyet() + " TL");
```


Gözlemci (Observer) Deseni

Nesneler arasında bire-çok bağımlılık tanımlar; öyle ki bir nesne durumunu değiştirdiğinde, bağımlılarının tümü otomatik olarak bilgilendirilir ve güncellenir.

```
import java.util.ArrayList;
import java.util.List;

// Gözlemci arayüzü
interface Gozlemci {
    void guncelle(String mesaj);
}

// Özne
class Ozne {
    private final List<Gozlemci> gozlemciler = new ArrayList<>();
    private String durum;

    public String getDurum() {
        return durum;
    }

    public void durumAyarla(String durum) {
        this.durum = durum;
        tumGozlemcileriBilgilendir();
    }

    public void ekle(Gozlemci gozlemci) {
        gozlemciler.add(gozlemci);
    }

    public void tumGozlemcileriBilgilendir() {
        for (Gozlemci gozlemci : gozlemciler) {
            gozlemci.guncelle(durum);
        }
    }
}
```

```
// Somut Gözlemci
class SomutGozlemci implements Gozlemci {
    private String isim;

    public SomutGozlemci(String isim) {
        this.isim = isim;
    }

    @Override
    public void guncelle(String mesaj) {
        System.out.println(isim + " aldı: " + mesaj);
    }
}

// Kullanım
// Ozne ozne = new Ozne();
// Gozlemci gozlemci1 = new SomutGozlemci("Gözlemci 1");
// Gozlemci gozlemci2 = new SomutGozlemci("Gözlemci 2");
// ozne.ekle(gozlemci1);
// ozne.ekle(gozlemci2);
// ozne.durumAyarla("Yeni Durum");
```

Strateji (Strategy) Deseni

Bir algoritma ailesi tanımlar, her birini kapsüller ve birbirinin yerine kullanılabilir hale getirir. Strateji, algoritmayı kullanıcılardan bağımsız olarak değiştirmeye izin verir.

```
// Strateji arayüzü
interface OdemeStratejisi {
    void ode(int miktar);
}

// Somut Stratejiler
class KrediKartiStratejisi implements OdemeStratejisi {
    private String isim;
    private String kartNumarasi;
    private String cvv;
    private String sonKullanmaTarihi;

    public KrediKartiStratejisi(String isim, String kartNumarasi, String cvv, String sonKullanmaTarihi) {
        this.isim = isim;
        this.kartNumarasi = kartNumarasi;
        this.cvv = cvv;
        this.sonKullanmaTarihi = sonKullanmaTarihi;
    }

    @Override
    public void ode(int miktar) {
        System.out.println(miktar + " TL kredi kartı ile ödendi");
    }
}
```

```
class PayPalStratejisi implements OdemeStratejisi {
    private String emailId;
    private String sifre;

    public PayPalStratejisi(String emailId, String sifre) {
        this.emailId = emailId;
        this.sifre = sifre;
    }

    @Override
    public void ode(int miktar) {
        System.out.println(miktar + " TL PayPal kullanılarak ödendi");
    }
}

// Bağlam
class AlisverisKart {
    private List<Urun> urunler;

    public AlisverisKart() {
        this.urunler = new ArrayList<Urun>();
    }

    public void urunEkle(Urun urun) {
        this.urunler.add(urun);
    }

    public int toplamHesapla() {
        int toplam = 0;
        for (Urun urun : urunler) {
            toplam += urun.getFiyat();
        }
        return toplam;
    }

    public void ode(OdemeStratejisi odemeStratejisi) {
        int miktar = toplamHesapla();
        odemeStratejisi.ode(miktar);
    }
}
```

Anti-desenler, etkisiz ve riskli olma eğiliminde olan tekrarlanan sorunlara yönelik yaygın çözümlerdir.

Yaygın Anti-Desenler

- **Tanrı Nesnesi (God Object):** Çok fazla şey bilen veya yapan bir sınıf
- **Spagetti Kodu (Spaghetti Code):** Yapılandırılmamış ve bakımı zor kod
- **Tekil Kötüye Kullanım (Singleton Abuse):** Tekil desenini aşırı kullanma
- **Altın Çekiç (Golden Hammer):** Sorundan bağımsız olarak tanıdık bir çözüm kullanmak
- **Tekerleği Yeniden İcat Etmek (Reinventing the Wheel):** Standart çözümler varken özel çözümler yaratmak
- **Erken Optimizasyon (Premature Optimization):** Darboğazları belirlemeden önce optimizasyon yapmak

Tasarım Deseni Seçim Kriterleri

Bir tasarım deseni seçerken şunları göz önünde bulundurun:

1. **Problem Bağlamı:** Çözmeye çalıştığınız belirli sorun nedir?
2. **Desen Sonuçları:** Bu deseni kullanmanın avantaj ve dezavantajları nelerdir?
3. **Alternatif Desenler:** Bu sorunu ele alabilecek başka desenler var mı?
4. **Uygulama Dili:** Bazı desenler belirli dillerde daha doğal olabilir
5. **Ekip Aşinalığı:** Ekibiniz bu deseni aşıyor mu?
6. **Bakım Yapılabilirlik:** Deseni, kodu daha bakımı yapılabilir hale getirecek mi?
7. **Performans Kaygıları:** Deseni performansı etkileyecek mi?

Gerçek Projelerde Desenleri Uygulamak

En İyi Uygulamalar

- Tasarım desenlerini uymadıkları yerlere zorlamayın
- Basit başlayın, gerektiğinde desenlere doğru yeniden düzenleyin
- Belirli bir deseni neden seçtiğinizi belgeleyin
- Sadece bileşenleri değil, tüm sistemi göz önünde bulundurun
- Desen kombinasyonları, tek tek desenlerden daha güçlü olabilir
- Desen uygulamalarını kapsamlı bir şekilde test edin

Güvenli Tasarım Desenleri

Güvenlik, yazılım tasarımında temel bir husus olmalıdır.

Önemli güvenlik tasarım desenleri şunlardır:

- **Güvenli Fabrika (Secure Factory):** Güvenlik kontrolleriyle nesne oluşturmaya merkezileştirme
- **Güvenli Vekil (Secure Proxy):** Hassas nesnelere erişimi kontrol etme
- **Güvenli Tekil (Secure Singleton):** Tek örneklerle güvenli erişim sağlama
- **Araya Giren Doğrulayıcı (Intercepting Validator):** Tüm girişleri merkezi doğrulayıcılarla doğrulama

Daha fazla güvenlik kontrolü: <https://www.cisecurity.org/controls/cis-controls-list>

Ek Kaynaklar

Mimaride ilk Tasarım Deseni kitabı:

<https://www.amazon.com/Pattern-Language-Buildings-Construction-Environmental/dp/0195019199>

Dörtlü Çete (Gang of Four - GoF) Tasarım Desenleri Kitabı:

<https://www.amazon.com/gp/product/0201633612/>

SOLID İlkeleri Kaynakları:

- <https://www.monterail.com/blog/solid-principles-cheatsheet-printable>
- https://www.monterail.com/hubfs/PDF_content/SOLID_cheatsheet.pdf
- <https://www.freecodecamp.org/news/solid-principles-explained-in-plain-english/>

Liskov Yerine Geçme İlkesi Örnekleri:

<https://code-examples.net/en/q/a476f2>

Bağımlılık Enjeksiyonu Kaynakları:

- <http://www.dotnet-stuff.com/tutorials/dependency-injection/understanding-and-implementing-inversion-of-control-container-ioc-container-using-csharp>
- <https://stackify.com/dependency-injection/>
- <https://www.tutorialsteacher.com/ioc/inversion-of-control>
- https://www.wikiwand.com/en/Dependency_injection
- <https://www.baeldung.com/inversion-control-and-dependency-injection-in-spring>

Gelecek Hafta

UML ve UMPLE ile devam edeceğiz, tasarımlarımızı modellemeye ve modellerden kod üretmeye odaklanacağız.