

# CEN206 Nesne Yönelimli Programlama

## Hafta-12 (UML, UMPLE ve Java Uygulamaları)

Bahar Dönemi, 2024-2025

İndir [DOC-PDF](#), [DOC-DOCX](#), [SLIDE](#), [PPTX](#)



## Ana Hatlar

- Birleşik Modelleme Dili (UML)
  - Genel Bakış ve Amaç
  - Ana Diyagram Türleri
  - Sınıf Diyagramları Detayları
  - Sıralama Diyagramları
  - Durum Diyagramları
- UMPLE
  - Model Güdümlü Geliştirmeye Giriş
  - UMPLE Dil Özellikleri
  - UMPLE ile Kod Üretimi
- UML Tasarımlarını Java'da Uygulama
  - UML'den Koda
  - En İyi Uygulamalar

- **Unified Modeling Language** (Birleşik Modelleme Dili)
- Yazılım sistemleri için standart görsel modelleme dili
- Nesne yönelimli modelleme için ortak bir kelime dağarcığı sağlar
- Grady Booch, James Rumbaugh ve Ivar Jacobson ("Üç Amigo") tarafından geliştirilmiştir
- Object Management Group (OMG) tarafından sürdürülmektedir
- Güncel sürüm: UML 2.5.1 (Aralık 2017)

## UML'nin Amacı

- Sistem mimarisini ve tasarımını görselleştirmek
- Sistem yapısını ve davranışını belirlemek
- Tasarım kararlarını belgelemek
- Paydaşlar arasında iletişimi kolaylaştırmak
- Uygulamaya rehberlik etmek

## Yapısal Diyagramlar

- **Sınıf diyagramı:** Sınıfları, arayüzleri ve aralarındaki ilişkileri gösterir
- **Nesne diyagramı:** Belirli bir zaman noktasında sınıfların örneklerini gösterir
- **Bileşen diyagramı:** Fiziksel bileşenlerin organizasyonunu gösterir
- **Dağıtım diyagramı:** Donanım topolojisini ve yazılım dağıtımını gösterir
- **Paket diyagramı:** Öğelerin mantıksal gruplandırmasını gösterir
- **Bileşik yapı diyagramı:** Bir sınıfın iç yapısını gösterir

## Davranışsal Diyagramlar

- **Kullanım durumu diyagramı:** İşlevselliği kullanıcı perspektifinden gösterir
- **Aktivite diyagramı:** İş akışını veya iş sürecini gösterir
- **Durum makinesi diyagramı:** Durumları ve geçişleri gösterir
- **Sıralama diyagramı:** Nesneler arasındaki etkileşimleri zaman içinde gösterir
- **İletişim diyagramı:** Nesneler arasındaki bağlantılara odaklanan etkileşimleri gösterir

# Sınıf Diyagramları

En yaygın kullanılan UML diyagramı, şunları gösterir:

- Sınıflar ve özellikleri
- Sınıflar arasındaki ilişkiler
- Arayüzler ve uygulamaları
- Kalıtım hiyerarşileri
- İlişkiler, bağımlılıklar ve daha fazlası



Sınıf Diyagramı Örneği

# Sınıf Gösterimi

 Sınıf Gösterimi

- Tek yönlü veya çift yönlü olabilir

## Birleştirme (Aggregation)

- "Sahip olma" ilişkisi (zayıf sahiplik)
- Sahibin ucunda elmas
- Parça bütünden bağımsız olarak var olabilir


## Kompozisyon (Composition)

- Birleştirmenin güçlü formu (güçlü sahiplik)
- Sahibin ucunda dolu elmas
- Parçanın yaşam döngüsü bütüne bağlıdır

## Genelleme/Kalıtım (Generalization/Inheritance)

- "Bir tür" ilişkisi
- Üst sınıfa doğru işaret eden üçgen

# Sınıf İlişkileri Örneği

 Sınıf İlişkileri Örneği



## Sıralama Diyagramları

Nesneler arasındaki etkileşimlerin zaman içindeki sırasını gösterir:

- Nesneler ve yaşam çizgileri
- Nesneler arasında alışverişi yapılan mesajlar
- Etkileşimlerin zaman sıralama düzeni
- Nesnelerin oluşturulması ve yok edilmesi
- Nesnelerin aktivasyonu ve deaktivasyonu



Sıralama Diyagramı Örneği

- Bir nesneyi zaman içinde temsil eder
- Dikey kesikli çizgi
- Aktivasyon çubukları içerebilir

## Mesaj (Message)

- Yaşam çizgileri arasındaki iletişim
- Senkron çağrılar için düz ok
- Asenkron çağrılar için kesikli ok
- Mesaj dönüşleri için dolu ok ucu ile ok

## Birleşik Parçalar (Combined Fragments)

- Koşullu davranışı tanımlar
- Alt (alternatifler), opt (isteğe bağlı), loop (döngü) vb. içerir
- Bir grup mesajı çevreler

## Durum Makinesi Diyagramları

Bir nesnenin mevcut durumuna göre olaylara nasıl tepki verdiğini gösterir:

- Durumlar
- Durumlar arası geçişler
- Geçişleri tetikleyen olaylar
- Geçişler sırasında gerçekleştirilen eylemler
- Giriş/çıkış eylemleri
- İç içe durumlar

 Durum Makinesi Diyagramı Örneği

- UML'yi programlama dilleriyle birleştirir
- UML'yi doğrudan kodun içine gömer
- Modellerden kod üretir
- Java, C++, PHP ve Ruby'yi destekler
- Açık kaynaklı ve web tabanlı araçlar mevcuttur

## UMPLE Felsefesi

- **Model-Kod İkililiği:** Modeller ve kod aynı yapıdır
- **Aşamalı Benimseme:** İhtiyaç duyulduğu kadar az veya çok kullanılabilir
- **Çoklu Görünümler:** Aynı sistemin farklı görünümüleri üretilebilir
- **Yürütülebilir Modeller:** Modeller doğrudan yürütülebilir

## UMPLE Araçları

- **UmpleOnline:** Web tabanlı düzenleyici ve kod üretici
- **Umple Eclipse Eklentisi:** Eclipse IDE ile entegre olur

# UMPLE Temel Sözdizimi

```
class Student {  
    // Tiplerle öznitelikler  
    Integer id;  
    String name;  
  
    // Çokluklu ilişkiler  
    * -- 1 University;  
  
    // Durum makinesi  
    status {  
        Active {  
            suspend -> Suspended;  
        }  
        Suspended {  
            reinstate -> Active;  
            expel -> Expelled;  
        }  
        Expelled {}  
    }  
  
    // Metotlar (hedef dilde)  
    void registerForCourse(Course c) {  
        // Hedef dildeki uygulama  
    }  
}
```

```

Integer age;
Date birthDate;

// Varsayılan değerlerle
const String country = "Türkiye";
Boolean isActive = true;

// Kısıtlamalarla
[age > 0]
[email ~= /^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$/]
}

```

## İlişkiler (Associations)

```

// Bire-çok ilişki
class Professor {
    // Bir profesör birçok ders verebilir
    1 -- * Course;
}

// Çoka-çok ilişki
class Student {
    // Öğrenciler birçok derse kayıt olabilir
    * -- * Course;
}

```

# UMPLE Durum Makineleri

```
class TrafficLight {  
    // Durum makinesi tanımı  
    status {  
        Red {  
            // Giriş ve çıkış eylemleri  
            entry / { turnOnRedLight(); }  
            exit / { turnOffRedLight(); }  
  
            // Koruma ve eylemle geçiş  
            timer [timeInState() > 60] -> Green { resetTimer(); }  
        }  
  
        Yellow {  
            entry / { turnOnYellowLight(); }  
            exit / { turnOffYellowLight(); }  
            timer [timeInState() > 5] -> Red;  
        }  
  
        Green {  
            entry / { turnOnGreenLight(); }  
            exit / { turnOffGreenLight(); }  
            timer [timeInState() > 45] -> Yellow;  
        }  
    }  
  
    // Durum makinesi tarafından gerekli olan metotlar  
    private void turnOnRedLight() { /* uygulama */ }  
    private void turnOffRedLight() { /* uygulama */ }  
    // Diğer metotlar...  
}
```

# UMPLE ile Kod Üretimi

## Java Kod Üretimi

```
class Car {  
    String make;  
    String model;  
    Integer year;  
  
    * -- 1 Manufacturer;  
  
    status {  
        Stopped {  
            startEngine -> Running;  
        }  
        Running {  
            stopEngine -> Stopped;  
            accelerate -> Accelerating;  
        }  
        Accelerating {  
            releaseGas -> Running;  
            brake -> Braking;  
        }  
        Braking {  
            stop -> Stopped;  
            releaseBreak -> Running;  
        }  
    }  
}
```



# Üretilen Java Kodu (Kısmi)

```
public class Car {
    //-----
    // ÜYE DEĞİŞKENLER
    //-----
    private String make;
    private String model;
    private Integer year;

    // Durum makinesi değişkenleri
    public enum Status { Stopped, Running, Accelerating, Braking }
    private Status status;

    //-----
    // CONSTRUCTOR
    //-----
    public Car(String aMake, String aModel, Integer aYear, Manufacturer aManufacturer) {
        make = aMake;
        model = aModel;
        year = aYear;
        boolean didAddManufacturer = setManufacturer(aManufacturer);
        if (!didAddManufacturer) {
            throw new RuntimeException("Üretici nedeniyle araba oluşturulamadı");
        }
        setStatus(Status.Stopped);
    }

    // Durum makinesi metotları, getter'lar, setter'lar, vb.
}
```

## UMPLE Online Demo

- UMPLE, model geliştirme için çevrimiçi bir ortam sağlar
- Gerçek zamanlı görselleştirme ve kod üretimi yapabilir
- UML kavramlarını hızlıca göstermek için kullanılabilir



UMPLE Online Ekran Görüntüsü

Ziyaret edin: <http://try.umple.org>

# UML'den Java Uygulamasına

## Sınıfları Uygulama

UML Özelliği	Java Uygulaması
Sınıf	<code>public class SınıfAdı</code>
Soyut sınıf	<code>public abstract class SınıfAdı</code>
Arayüz	<code>public interface ArayüzAdı</code>
Öznitelikler	Uygun erişim belirleyicileri ile alanlar
İşlemler	Uygun imzalara sahip metotlar
Görünürlük	<code>public</code> , <code>private</code> , <code>protected</code> veya paket-özel

## Kalıtım/Genelleme

```
// UML: Child, Parent'tan miras alır
public class Parent {
    // Parent üyeleri
}

public class Child extends Parent {
    // Child üyeleri
}
```

## Uygulama (Arayüz)

```
// UML: Class, Interface'i uygular
public interface MyInterface {
    void doSomething();
}

public class MyClass implements MyInterface {
    @Override
    public void doSomething() {
        // Uygulama
    }
}
```

# İlişkilendirme (Association)

```
// UML: A sınıfının B sınıfına bir referansı var
public class A {
    private B b; // Bire-bir

    public A(B b) {
        this.b = b;
    }

    public B getB() {
        return b;
    }

    public void setB(B b) {
        this.b = b;
    }
}

// UML: C sınıfının D sınıfına birçok referansı var
public class C {
    private List<D> dList; // Bire-çok

    public C() {
        this.dList = new ArrayList<>();
    }

    public void addD(D d) {
        dList.add(d);
    }

    // İlişkiyi yönetmek için diğer metotlar
}
```

`private final` Part part; // 'final' güçlü sahipliği vurgular

CEN206 Nesne Yönelimli Programlama

```
public Whole() {  
    // Bütün oluşturulduğunda parça da oluşturulur  
    this.part = new Part();  
}  
  
// Part için setter yok - oluşturulduktan sonra değiştirilemez  
public Part getPart() {  
    return part;  
}  
}
```

## Birleştirme (Aggregation)

```
// UML: Container sınıfı Part'lara sahiptir (birleştirme)  
public class Container {  
    private Part part; // Final değil - bağımsız olarak var olabilir  
  
    public Container(Part part) {  
        this.part = part;  
    }  
}
```

// Part değiştirilebilir veya null olarak ayarlanabilir

# Davranışsal Öğeleri Uygulama

## Durum Makineleri

```
public class Document {  
    // Durum enumeration'ı  
    public enum State {  
        DRAFT, REVIEW, APPROVED, PUBLISHED  
    }  
  
    private State currentState;  
  
    public Document() {  
        currentState = State.DRAFT;  
    }  
  
    public void submitForReview() {  
        if (currentState == State.DRAFT) {  
            currentState = State.REVIEW;  
            System.out.println("Belge inceleme için gönderildi");  
        } else {  
            System.out.println("Gönderilemiyor - DRAFT durumunda değil");  
        }  
    }  
  
    public void approve() {  
        if (currentState == State.REVIEW) {  
            currentState = State.APPROVED;  
            System.out.println("Belge onaylandı");  
        } else {  
            System.out.println("Onaylanamıyor - REVIEW durumunda değil");  
        }  
    }  
}
```

// Daha fazla durum geçiş metodu...

# Vaka Çalışması: Çevrimiçi Alışveriş Sistemi

 Çevrimiçi Alışveriş UML Diyagramı



# UMPLE Uygulaması

```
class Customer {
    String name;
    String email;
    String address;

    1 -- * Order;
}

class Order {
    Date orderDate;
    Float totalAmount;

    status {
        New {
            processPayment -> PaymentProcessing;
        }
        PaymentProcessing {
            paymentSuccessful -> Confirmed;
            paymentFailed -> PaymentFailed;
        }
        PaymentFailed {
            retry -> PaymentProcessing;
            cancel -> Cancelled;
        }
        Confirmed {
            ship -> Shipped;
        }
        Shipped {
            deliver -> Delivered;
        }
        Delivered {}
        Cancelled {}
    }

    * -- * Product;
}

class Product {
    String name;
    String description;
    Float price;
    Integer stockQuantity;
}
```

# Java Uygulaması (Kısmi)

```
public class Customer {
    private String name;
    private String email;
    private String address;
    private List<Order> orders;

    public Customer(String name, String email, String address) {
        this.name = name;
        this.email = email;
        this.address = address;
        this.orders = new ArrayList<>();
    }

    public void addOrder(Order order) {
        orders.add(order);
    }

    // Getter'lar, setter'lar ve diğer metotlar
}

public class Order {
    private Date orderDate;
    private float totalAmount;
    private List<Product> products;
    private OrderState state;

    public Order() {
        this.orderDate = new Date();
        this.products = new ArrayList<>();
        this.state = OrderState.NEW;
    }

    // Durum geçiş metotları, getter'lar, setter'lar ve diğer işlevler
}
```

- Her sınıfın değişmek için sadece bir nedeni olmalıdır

## 2. Alan erişimini kapsülleyin

- Private alanları public getter/setter'larla kullanın

## 3. Kalıtım yerine kompozisyonu tercih edin

- "Sahip olma" genellikle "bir tür olma"dan daha iyidir

## 4. Davranış için arayüzleri uygulayın

- Arayüzleri sözleşmeleri tanımlamak için kullanın

## 5. Tasarım desenlerini uygun şekilde kullanın

- Desenleri yaygın problemlerle eşleştirin

## 6. Varlıkları mümkün olduğunca değişmez tutun

- Özellikle değer nesneleri için

## 7. Modele karşı test edin

## UMPLE vs. Diğer UML Araçları

Özellik	UMPLE	Geleneksel UML Araçları
Kod Entegrasyonu	Kodla doğrudan entegrasyon	Koddan ayrı modeller
Öğrenme Eğrisi	Orta (bir dil uzantısı öğrenmek gibi)	Dik (tamamen farklı paradigma)
Çift Yönlü Mühendislik	Doğal (model ve kod aynıdır)	Genellikle problemlidir
Sürüm Kontrolü	Standart kaynak kontrolü araçları	Özel araçlar gerektirebilir
IDE Desteği	Değişir, iyi Eclipse desteği	Genellikle kapsamlı

1. Şunları gösteren bir UML sınıf diyagramı oluşturun:

- BankAccount (soyut sınıf)
- SavingsAccount ve CheckingAccount (somut sınıflar)
- Hesaplarla ilişkili Customer
- Hesaplarla ilgili Transaction sınıfı

2. Hesap durumu için bir durum makinesi modelleyin

3. UMPLE kullanarak Java'da uygulayın

4. Kod üretin ve uygulamayı test edin

## Öğrenme Çıktıları

- UML modellemesi pratiği yapın
- UMPLE ile deneyim kazanın

## Kaynaklar

- OMG Unified Modeling Language Specification: <https://www.omg.org/spec/UML/>
- UMPLE Kullanım Kılavuzu: <https://cruise.umple.org/umple/>
- Fowler, M. (2003). UML Distilled: A Brief Guide to the Standard Object Modeling Language. Addison-Wesley.
- Rumbaugh, J., Jacobson, I., & Booch, G. (2004). The Unified Modeling Language Reference Manual. Addison-Wesley.
- UMPLE GitHub Deposu: <https://github.com/umple/umple>

## Gelecek Hafta

Quiz 2 - UML, UMPLE, tasarım desenleri ve Java uygulamalarını kapsayacaktır.