SUNUM: OOP TEMEL PRENSİPLERİ

Access Modifiers → Encapsulation → Abstraction → Abstract Class → Interface

---

1️⃣ ERİŞİM BELİRLEYİCİLER (ACCESS MODIFIERS)

> Encapsulation'ı anlayabilmek için önce erişim seviyelerini bilmek gerekir.

1.1 public

- Her yerden erişilebilir
- En geniş erişim seviyesi

```
public int age;
```

1.2 private

- Sadece tanımlandığı sınıf içinde erişilebilir
- Veri gizleme mekanizmasının temelidir

```
private String password;
```

Önemli:

> Private üyeler alt sınıflar tarafından doğrudan erişilemez.

1.3 protected

- Aynı paket + alt sınıflar erişebilir
- Kalıtımda kontrollü erişim sağlar

```
protected double salary;
```

1.4 default (package-private)

- Sadece aynı paket içinde erişilebilir

- Modifier yazılmaz

```
int count; // default
```

1.5 Erişim Tablosu

Modifier	Aynı Sınıf	Aynı Paket	Alt Sınıf	Her Yer
public	✓	✓	✓	✓
protected	✓	✓	✓	✗
default	✓	✓	✗	✗
private	✓	✗	✗	✗

2️⃣ KAPSÜLLEME (ENCAPSULATION)

2.1 Encapsulation Nedir?

Bir sınıfın içindeki verileri private yaparak dış dünyadan gizleyip, bu verilere kontrollü erişim sağlamaktır.

> Temel Amaç: Veri güvenliği + Kontrol + Sürdürülebilirlik

Encapsulation veri saklamak değildir. Veri üzerinde kontrol kurmaktır.

2.2 Encapsulation Olmadan (Hatalı Tasarım)

```
class BankAccount {
    public double balance;
}

BankAccount acc = new BankAccount();
acc.balance = -10000; // Kontrol yok
```

Sorun:

- Negatif bakiye engellenemez
- İş kuralları uygulanamaz
- Nesne kendi durumunu koruyamaz

2.3 Encapsulation Uygulanmış Hali

```
class BankAccount {

    private double balance;

    public double getBalance() {
        return balance;
    }

    public void deposit(double amount) {
        if(amount > 0) {
            balance += amount;
        }
    }

    public void withdraw(double amount) {
        if(amount > 0 && amount <= balance) {
            balance -= amount;
        }
    }
}
```

Artık:

- Veri doğrudan değiştirilemez
- Kurallar sınıf içinde uygulanır
- Nesne kendi iç durumunu korur

2.4 Getter ve Setter Mantığı (Detaylı)

Getter Nedir?

Private bir değişkenin değerini okumaya yarayan metottur. Ancak sadece değeri döndürmek zorunda değildir. Gerekirse:

- Formatlayarak döndürebilir
- Hesaplanmış bir değer döndürebilir
- Filtreleme uygulayabilir

Setter Nedir?

Private değişkene değer atarken kontrol yapmamızı sağlar. Amaç:

> Veriyi nesneye almadan önce doğrulamaktır.

```
class User {

    private int age;

    public int getAge() {
        return age;
    }

    public void setAge(int age) {
        if(age >= 0 && age <= 120) {
            this.age = age;
        }
    }
}
```

Getter & Setter sayesinde:

- Validation yapılır
- İş kuralları uygulanır
- Nesnenin iç durumu korunur
- Sistem değişse bile dış dünya etkilenmez

3️⃣ SOYUTLAMA (ABSTRACTION)

3.1 Abstraction Nedir?

Bir nesnenin ne yaptığını gösterip, nasıl yaptığını gizlemektir.

Kullanıcı davranışı bilir. İç detayları bilmek zorunda değildir.

3.2 Gerçek Hayat Örneği

Araba:

- Gaza basarsın → gider
- Fren yaparsın → durur

Motorun iç mekanizmasını bilmezsin. Bu abstraction'dır.

3.3 Abstraction'ın Amacı

- Karmaşıklığı azaltmak
- Sistemi modüler hale getirmek

• Kullanıcıyı teknik detaydan korumak

4️⃣ ABSTRACT SINIFLAR VE METOTLAR

4.1 Abstract Class Nedir?

• Nesnesi oluşturulamaz
• Hem abstract hem normal metot içerebilir
• Alt sınıflar için şablon görevi görür

```
abstract class Animal {

    abstract void makeSound();

    void sleep() {
        System.out.println("Hayvan uyuyor");
    }
}
```

Burada:

• makeSound() → Alt sınıf yazmak zorunda
• sleep() → Ortak davranış

4.2 Abstract Metot Nedir?

• Gövdesi yoktur
• Alt sınıfta override edilmek zorundadır

```
class Dog extends Animal {

    @Override
    void makeSound() {
        System.out.println("Hav hav");
    }
}
```

Dog sınıfı makeSound metodunu yazmak zorundadır. Yazmazsa derleme hatası oluşur.

5️⃣ INTERFACE (ARAYÜZ)

5.1 Interface Nedir?

Interface bir davranış sözleşmesidir.

> "Bu davranışı uygulamak zorundasın" der.

```
interface Flyable {
    void fly();
}
```

5.2 Kuş Örneği ile Interface Mantığı

Kuş olmak bir varlıktır → class
Uçabilmek bir davranıştır → interface

Tüm kuşlar uçamaz (örneğin penguen). Bu yüzden uçma davranışı interface olmalıdır.

Bird

```
abstract class Bird {

    void eat() {
        System.out.println("Kuş yemek yiyor");
    }
}
```

Parrot (Uçabilen)

```
class Parrot extends Bird implements Flyable {

    @Override
    public void fly() {
        System.out.println("Papağan uçuyor");
    }
}
```

Penguin (Uçamayan)

```
class Penguin extends Bird {
}
```

Penguin Flyable implement etmez. Çünkü uçma davranışına sahip değildir.

5.3 Abstract Class vs Interface

Abstract Class	Interface
Kısmi soyutlama	Davranış sözleşmesi
Tek kalıtım	Çoklu implement
Constructor var	Constructor yok
State tutabilir	Genelde state tutmaz

6️⃣ GENEL ÖZET

Access Modifiers → Erişimi kontrol eder
Encapsulation → Veriyi korur
Abstraction → Karmaşıklığı gizler
Abstract Class → Ortak yapı + kısmi soyutlama
Interface → Davranış sözleşmesi

> OOP'de amaç kodu çalıştırmak değil, kontrol edilebilir ve sürdürülebilir sistem tasarlamaktır.