JAVA PROGRAMLAMA

OOP KAVRAMLARI

Adım Adım Java Öğrenme Rehberi

İÇİNDEKİLER

OOP KAVRAMLARI 1
İÇİNDEKİLER 2
1. JAVA'YA GİRİŞ VE İLK CLASS 3
1.1 Class ve Object Nedir? 3
Örnek: Araba Sınıfı 3
Private ve Public Erişim Belirleyiciler 4
2. ENCAPSULATION (KAPSÜLLEME) 5
2.1 Encapsulation Nedir? 5
Banka Hesabı Örneği 5
3. INHERITANCE (KALITIM) 7
3.1 Inheritance Nedir? 7
Çalışan Hiyerarşisi Örneği 7
4. POLYMORPHISM (ÇOK BİÇİMLİLİK) 8
4.1 Polymorphism Nedir? 8
4.2 Interface (Arayüz) ile Polymorphism 8
4.3 Polymorphism'in Gücü 8
4.4 Method Overloading (Metod Aşırı Yükleme) 9
5. ABSTRACTION (SOYUTLAMA) 10
5.1 Abstraction Nedir? 10
5.2 Abstract Class 10
5.3 Interface vs Abstract Class 11
6. OOP'NİN 4 TEMELİ - ÖZET 13
6.1 Encapsulation (Kapsülleme) 13
6.2 Inheritance (Kalıtım) 13
6.3 Polymorphism (Çok Biçimlilik) 13
6.4 Abstraction (Soyutlama) 13
7. JAVA ÖĞRENİRKEN PRATİK ÖNERİLER 14
7.1 Kod Yazma Alışkanlıkları 14
7.2 Öğrenme Sırası Önerisi 14
7.3 Proje Önerileri 14
SONUÇ 15

1. JAVA'YA GİRİŞ VE İLK CLASS

1.1 Class ve Object Nedir?

Class (Sınıf): Bir şeyin taslağı, planıdır. Mesela bir ev planı gibi düşünebilirsiniz. Bir evin nasıl olacağını, kaç odası olacağını, hangi özelliklere sahip olacağını planlar üzerinde görürüz.

Object (Nesne): O plandan yapılan gerçek şeydir. Plandan yapılan gerçek ev gibi. Aynı plandan birden fazla ev yapabilirsiniz, her biri farklı bir nesnedir.

Örnek: Araba Sınıfı

```
public class Araba {

    private String marka;
    private String model;
    private int yil;
    private boolean motorCalisiyorMu;

    public Araba(String marka, String model, int yil) {
        this.marka = marka;
        this.model = model;
        this.yil = yil;
        this.motorCalisiyorMu = false;
    }
    public void motoruCalistir() {

        if (!motorCalisiyorMu) {
            motorCalisiyorMu = true;
            System.out.println(marka + " " + model + " motoru çal□şt□r□ld□.");
        } else {
            System.out.println("Motor zaten çal□ş□yor!");
        }
    }
    public void motoruDurdur() {
        if (motorCalisiyorMu) {
            motorCalisiyorMu = false;
            System.out.println("Motor durduruldu.");
        }
    }
    public String getBilgi() {
        return yil + " model " + marka + " " + model;
    }
}

public class Main {
    public static void main(String[] args) {
        Araba araba1 = new Araba("Toyota", "Corolla", 2023);

        System.out.println(araba1.getBilgi());
        araba1.motoruCalistir();
        araba1.motoruCalistir();
    }
}
```

Private ve Public Erişim Belirleyiciler

- **private:** Sadece bu class içinden erişilebilir (gizli, korumalı)
- **public:** Her yerden erişilebilir (açık, herkese görünür)

Neden private kullanırız? → ENCAPSULATION (Kapsülleme) için!

2. ENCAPSULATION (KAPSÜLLEME)

2.1 Encapsulation Nedir?

Verileri gizleyip, sadece kontrollü yollarla erişim sağlamaktır. Gerçek hayattan örnek: ATM'den para çekerken bankanın kasasına giremezsiniz, sadece ATM'yi kullanarak kontrollü bir şekilde işlem yaparsınız.

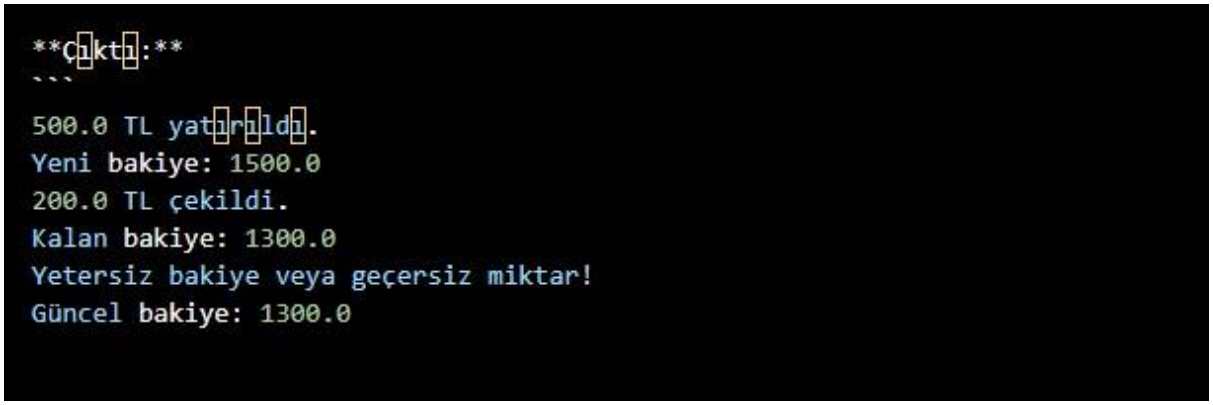
Banka Hesabı Örneği

```
public class BankaHesabi {
    private double bakiye;
    private String hesapNo;

    public BankaHesabi(String hesapNo, double ilkBakiye) {
        this.hesapNo = hesapNo;
        this.bakiye = ilkBakiye;
    }

    public void paraYatir(double miktar) {
        if (miktar > 0) {
            bakiye += miktar;
            System.out.println(miktar + " TL yat□r□ld□.");
            System.out.println("Yeni bakiye: " + bakiye);
        } else {
            System.out.println("Geçersiz miktar!");
        }
    }
    public boolean paraCek(double miktar) {
        if (miktar > 0 && miktar <= bakiye) {
            bakiye -= miktar;
            System.out.println(miktar + " TL çekildi.");
            System.out.println("Kalan bakiye: " + bakiye);
            return true;
        }
        System.out.println("Yetersiz bakiye veya geçersiz miktar!");
        return false;
    }

    public double getBakiye() {
        return bakiye;
    }
}
public class BankaTest {
    public static void main(String[] args) {
        BankaHesabi hesap = new BankaHesabi("TR123456", 1000.0);
        hesap.paraYatir(500);
        hesap.paraCek(200);
        hesap.paraCek(2000);
        System.out.println("Güncel bakiye: " + hesap.getBakiye());
    }
}
```



Encapsulation'ın Faydaları:

- Güvenlik: Veriler korunur ve istenmeyen değişikliklerden saklanır
- Kontrol: İşlemler kurallarla yapılır, geçersiz durumlar engellenir
- Esneklik: İç yapıyı değiştirseniz bile dış arayüz aynı kalır

3. INHERITANCE (KALITIM)

3.1 Inheritance Nedir?

Bir sınıfın başka bir sınıftan özellik ve davranışları miras almasıdır. Nasıl ki çocuklar anne-babadan özellikler miras alır, programlamada da sınıflar birbirinden özellik miras alabilir.

Çalışan Hiyerarşisi Örneği

```java
public class Calisan {
    protected String ad;
    protected String soyad;
    protected double maas;
    public Calisan(String ad, String soyad, double maas) {
        this.ad = ad;
        this.soyad = soyad;
        this.maas = maas;
    }
    public void bilgiGoster() {
        System.out.println("Çalışan: " + ad + " " + soyad);
        System.out.println("Maaş: " + maas + " TL");
    }
    public double yillikMaasHesapla() {
        return maas * 12;
    }
}
public class Yazilimci extends Calisan {
    private String programlamaDili;
    private int deneyimYili;
    public Yazilimci(String ad, String soyad, double maas,
                     String programlamaDili, int deneyimYili) {
        super(ad, soyad, maas);
        this.programlamaDili = programlamaDili;
        this.deneyimYili = deneyimYili;
    }
    @Override
    public void bilgiGoster() {
        super.bilgiGoster();
        System.out.println("Programlama Dili: " + programlamaDili);
        System.out.println("Deneyim: " + deneyimYili + " yıl");
    }
    public void kodYaz() {
        System.out.println(ad + " " + programlamaDili + " ile kod yazıyor...");
    }
}
public class InheritanceTest {
    public static void main(String[] args) {
        Yazilimci dev = new Yazilimci("Ali", "Yılmaz", 15000, "Java", 5);
        dev.bilgiGoster();
        dev.kodYaz();
        System.out.println("Yıllık maaş: " + dev.yillikMaasHesapla());
        System.out.println("\n---\n");
    }
}
```

Önemli Kavramlar:

- • **extends:** Kalıtım için kullanılan anahtar kelime
- • **super():** Üst sınıfın constructor'ını çağırır
- • **@Override:** Üst sınıftaki metodu yeniden yazıyoruz demek
- • **protected:** Alt sınıfların erişebileceği erişim belirleyici

4. POLYMORPHISM (ÇOK BİÇİMLİLİK)

4.1 Polymorphism Nedir?

Aynı isimli metodun farklı şekillerde çalışabilmesidir. Gerçek hayattan örnek: "Aç" komutu - Köpeğe "aç" derseniz havlar, kapıya "aç" derseniz kapı açılır, dosyaya "aç" derseniz dosya açılır. Aynı komut, farklı nesnelerde farklı davranışlar gösterir!

4.2 Interface (Arayüz) ile Polymorphism

Interface bir sözleşme gibidir. "Bu metodları mutlaka implement et" der.

```
public interface Sekil {

    double alanHesapla();
    double cevreHesapla();
    void ciz();
}
public class Daire implements Sekil {
    private double yaricap;

    public Daire(double yaricap) {
        this.yaricap = yaricap;
    }

    @Override
    public double alanHesapla() {
        return Math.PI * yaricap * yaricap;
    }

    @Override
    public double cevreHesapla() {
        // Daire çevresi = 2 x π x r
        return 2 * Math.PI * yaricap;
    }

    @Override
    public void ciz() {
        System.out.println("🔵 Daire çiziliyor...");
        System.out.println("   Yarıçap: " + yaricap);
    }
}
```

4.3 Polymorphism'in Gücü

```java
Sekil[] sekiller = new Sekil[4];
        sekiller[0] = new Daire(5.0);
        sekiller[1] = new Dikdortgen(4.0, 6.0);
        sekiller[2] = new Ucgen(3.0, 4.0, 5.0);
        sekiller[3] = new Daire(3.0);

        System.out.println("=== TÜM ŞEKİLLERİ İŞLE ===\n");

        // Aynı kod, farklı nesnelerde farklı çalışır
        for (Sekil sekil : sekiller) {
            sekil.ciz();  // Her şekil kendine göre çizer
            System.out.println("   Alan: " +
                String.format("%.2f", sekil.alanHesapla()));
            System.out.println("   Çevre: " +
                String.format("%.2f", sekil.cevreHesapla()));
            System.out.println();
        }
```

4.4 Method Overloading (Metod Aşırı Yükleme)

Aynı isimde, farklı parametreli metodlar yasmak

```java
public class Hesap {

    // Aynı isim, farklı parametre sayısı
    public int topla(int a, int b) {
        return a + b;
    }
    public int topla(int a, int b, int c) {
        return a + b + c;
    }
    // Aynı isim, farklı parametre tipi
    public double topla(double a, double b) {
        return a + b;
    }

}
```

5. ABSTRACTION (SOYUTLAMA)

5.1 Abstraction Nedir?

Detayları gizleyip, sadece önemli kısımları göstermek. Gerçek hayat örneği: Araba kullanırken motor nasıl çalışır bilmenize gerek yok. Sadece gaza basarsınız!

5.2 Abstract Class

Abstract class hem abstract metodları hem normal metodları içerebilir. Abstract class'tan direkt nesne oluşturamazsınız.

```java
public abstract class Odeme {
    protected double tutar;
    protected String aciklama;

    public Odeme(double tutar, String aciklama) {
        this.tutar = tutar;
        this.aciklama = aciklama;
    }
    // ABSTRACT METOD - Gövde yok!
    // Alt sınıflar MUTLAKA implement etmeli
    public abstract boolean odemeYap();
    public abstract String odemeTipi();

    // Bu metod değişmez, herkeste aynı
    public void faturaBilgisi() {
        System.out.println("  Ödeme Türü: " + odemeTipi());
        System.out.println("  Tutar: " + tutar + " TL");
    }
}
public class KrediKartiOdeme extends Odeme {
    private String kartNumarasi;
    private String cvv;

    public KrediKartiOdeme(double tutar, String aciklama,
                           String kartNumarasi, String cvv) {
        super(tutar, aciklama);  // Üst sınıfın constructor'ı
        this.kartNumarasi = kartNumarasi;
        this.cvv = cvv;
    }
    // Abstract metodu IMPLEMENT ediyoruz
    @Override
    public boolean odemeYap() {
        System.out.println("💳 Kredi kartı ile ödeme yapılıyor...");

        if (kartNumarasi.length() == 16 && cvv.length() == 3) {
            System.out.println("✅ Ödeme başarılı!");
            return true;
        }
        return false;
    }
    @Override
    public String odemeTipi() {
        return "Kredi Kartı";
    }

}
```

5.3 Interface vs Abstract Class

Interface:

- Sadece abstract metodlar (Java 8 öncesi)
- Çoklu implement edilebilir (bir class birden fazla interface implement edebilir)
- Constructor yok
- %100 abstraction sağlar

Abstract Class:

- Hem abstract hem normal metodlar içerebilir

- Sadece tek inherit edilir
- Constructor olabilir
- Kısmi abstraction sağlar

6. OOP'NİN 4 TEMELİ - ÖZET

6.1 Encapsulation (Kapsülleme)

Ne: Verileri gizleme ve kontrollü erişim sağlama

Nasıl: private değişkenler + public getter/setter metodları

Neden: Güvenlik, kontrol, esneklik

6.2 Inheritance (Kalıtım)

Ne: Bir sınıfın başka sınıftan özellik miras alması

Nasıl: extends anahtar kelimesi ile

Neden: Kod tekrarını azaltır, hiyerarşik yapı oluşturur

6.3 Polymorphism (Çok Biçimlilik)

Ne: Aynı metodun farklı şekillerde çalışabilmesi

Nasıl: Interface implementation, method overriding/overloading

Neden: Esneklik, genişletilebilirlik, temiz kod

6.4 Abstraction (Soyutlama)

Ne: Detayları gizleyip önemli kısımları gösterme

Nasıl: Abstract class veya Interface kullanarak

Neden: Karmaşıklığı azaltır, bakımı kolaylaştırır

7. JAVA ÖĞRENİRKEN PRATİK ÖNERİLER

7.1 Kod Yazma Alışkanlıkları

- Her gün düzenli kod yazın, tutarlılık önemlidir
- Küçük projelerle başlayın, karmaşık projelere yavaş geçin
- Her yeni kavramı mutlaka kodlayarak öğrenin
- Hata mesajlarını okuyun ve anlamaya çalışın
- Debug yaparak kodun nasıl çalıştığını takip edin

7.2 Öğrenme Sırası Önerisi

1. Temeller: Değişkenler, veri tipleri, operatörler, döngüler, koşullar

2. OOP Kavramları: Class, Object, Encapsulation, Inheritance, Polymorphism, Abstraction

3. Collections: ArrayList, HashMap, HashSet gibi veri yapıları

4. Exception Handling: try-catch, throw, throws, custom exceptions

5. Generics: Tip güvenli programlama

6. Modern Java: Lambda expressions, Stream API, Optional

7. Design Patterns: Singleton, Factory, Builder, Observer, Strategy

7.3 Proje Önerileri

Başlangıç: Hesap makinesi, not defteri, öğrenci kayıt sistemi

Orta Seviye: Kütüphane yönetim sistemi, banka uygulaması, e-ticaret sepeti

İleri Seviye: Çok kullanıcılı chat uygulaması, web servisi, oyun geliştirme

SONUÇ

Bu dokümanda Java programlama dilinin temellerini ve Object-Oriented Programming (OOP) kavramlarını öğrendiniz. Encapsulation, Inheritance, Polymorphism ve Abstraction olmak üzere OOP'nin 4 temel direğini kod örnekleriyle inceledik.

Unutmayın: Programlama öğrenmek bir maraton, sprint değil. Her gün düzenli pratik yaparak ve gerçek projeler geliştirerek ilerleyebilirsiniz. Her kavramı mutlaka kodlayarak pekiştirin ve kendi projelerinizi geliştirin.