

JAVA METHODS

Java'da methodlar aynı işlemi yapan kod bloklarıdır. Programımız içerisinde bir kod bloğunu birden fazla yerde kullanmak istediğimizde metotları kullanırız. Başka dillerdeki fonksiyon olarak adlandırdığımız kod bloklarına Java'da metot denir. Yapmasını istediğimiz işlemlerden herhangi bir değer döndürmesini veya doğrudan bu işlemleri gerçekleştirip bitmesini isteyebiliriz

Örneğin önce girilen sayının karekökünü alıp daha sonra ekrana çıktı olarak vermesini isteyebiliriz. Bu işlemleri metotlar sayesinde daha kolay ve daha düzenli bir şekilde yaparız.

JAVA 'DA METOT OLUŞTURMA

Java'da metot oluşturmak için belirli bir kurala uymamız gerekiyor .

Bu kural: **Erişim seviyesi** + **Dönüş tipi** + **Metot adı** ve oluşturacağımız metot türüne göre son olarak **parametre** eklenebilir.

```
erişimSeviyesi DonusTipi metotAdi(parametreListesi) {  
  
    // Metot gövdesi  
  
}
```

Dönüş Tipi : Oluşturduğumuz metodun bize geri dönüş tipini belirttiğimiz yerdir. Bu alana String, int, double , boolean , char gibi tipler gelebilir. Oluşturacağımız metot bir değer döndürmüyor ise void olarak tanımlanmalıdır.

Erişim Seviyesi : Metodun erişim düzeyini belirler (public, private, protected, default). Bu konuyu daha sonra ayrıntılı bir şekilde göreceğiz.

Parametre Listesi : Metodun alacağı parametrelerin veri tipleri ve adlarıdır. Parametre yoksa boş bırakılmalıdır.

Metot Adı : Oluşturduğumuz metodun adıdır. Adlandırma kurallarına da aşağıda değineceğim.

DÖNÜŞ TİPLERİNE GÖRE METOT ÖRNEKLERİ

```
package ileri_java;

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

public class Methods {

    public void mesaj() {
        System.out.println("Bu metot geriye bir şey döndürmüyor..");
    }

    public int toplama(int a, int b) {
        return a + b;
    }

    public String selamlama(String isim) {
        return "Merhaba, " + isim;
    }

    public boolean isPositive(int number) {
        return number > 0;
    }

    public List<String> getList() {
        List<String> list = new ArrayList<>();
        list.add("Karpuz");
        list.add("Çilek");
        list.add("Kiraz");
        return list;
    }

    public static void main(String[] args) {
        Methods methods = new Methods();

        methods.mesaj();
        System.out.println(methods.toplama(5, 10));
        System.out.println(methods.selamlama("Serhat"));
        System.out.println(methods.isPositive(-5));
        System.out.println(methods.getList());
    }
}
```

BU KODLARDA FARKLI GERİ DÖNÜŞ TİPLERİYLE METOTLAR OLUŞTURDUM.

```
Bu metot geriye bir şey döndürmüyor..
15
Merhaba, Serhat
false
[Karpuz, Çilek, Kiraz]
```

METOTLARDA ERİŞİM SEVİYELERİ

Erişim seviyeleri, hassas verilerin ve işlemlerin dışarıdan doğrudan erişimini engelleyerek güvenliği artırır. Bu, sadece belirli metotların veya sınıfların kritik verilere erişmesini sağlar. Java'da metotları tanımlarken uygulamamızın güvenliği için bazı önlemler almamız gerekebilir.

1-) public: Bir metot veya değişken public olarak tanımlandığında, bu metot veya değişkene her yerden erişilebilir. Bu, diğer sınıfların ve paketlerin bu metodu veya değişkeni kullanabileceği anlamına gelir. Genel kullanıma açık ve sınıfın bir parçası olarak sunulan temel API'ler için uygundur.

2-) protected : Protected olarak tanımlanan metot veya değişkene, aynı paket içindeki diğer sınıflar ve bu sınıfı miras alan alt sınıflar tarafından erişilebilir. Alt sınıfların ihtiyaç duyabileceği ancak genel kullanıma açılmaması gereken metotlar ve değişkenler için uygundur.

3-) private : Private olarak tanımlanan metot veya değişken sadece tanımlandığı sınıf içinde erişilebilir. Bu, diğer sınıfların ve hatta alt sınıfların bile bu metot veya değişkene erişemeyeceği anlamına geliyor. Sadece sınıf içi işlemler için gerekli olan, dışarıya açılmaması gereken metotlar ve değişkenler için uygundur. Sadece Getter ve Setter metodlarıyla erişilebilir.

4-) default : Hiçbir erişim belirleyici belirtilmediğinde, metot veya değişken paket özel erişime sahip olur. Bu, aynı paket içindeki diğer sınıflar tarafından erişilebilir, ancak farklı paketlerden erişilemez. Aynı paket içinde kullanılacak ancak paket dışına açılmaması gereken metotlar ve değişkenler için uygundur.

METOTLARDA ERIŞİM SEVİYELERİ ÖRNEĞİ

```
package ileri_java;

public class AccessModifiers {
    public int genelDegisken;
    private int ozelDegisken;
    protected int korunanDegisken;
    int varsayilanDegisken;

    public AccessModifiers() {
        genelDegisken = 10;
        ozelDegisken = 20;
        korunanDegisken = 30;
        varsayilanDegisken = 40;
    }

    // Public erişim belirteçli metot
    public void genelMetot() {
        System.out.println("Bu bir genel metottur.");
    }

    // Private erişim belirteçli metot
    private void ozelMetot() {
        System.out.println("Bu bir özel metottur.");
    }

    // Protected erişim belirteçli metot
    protected void korunanMetot() {
        System.out.println("Bu bir korunan metottur.");
    }

    // Varsayılan (paket-özel) erişim belirteçli metot
    void varsayilanMetot() {
        System.out.println("Bu bir varsayılan metottur.");
    }
}
```

```
package ileri_java;

public class AccessModifiersTest {
    public static void main(String[] args) {
        AccessModifiers obj = new AccessModifiers();

        System.out.println("Genel değişken: " + obj.genelDegisken);
        System.out.println("Özel değişken: " + obj.ozelDegisken); // Bu bir hata verecek çünkü
        // değişken private erişemiyoruz.
        System.out.println("Korunan değişken: " + obj.korunanDegisken);
        System.out.println("Varsayılan değişken: " + obj.varsayilanDegisken);

        obj.genelMetot();
        obj.ozelMetot(); // Bu bir hata verecek çünkü metot private buna da erişemiyoruz.
        obj.korunanMetot();
        obj.varsayilanMetot();
    }
}
```

_BU İKİ AYRI SAYFADAKİ KODLARDA PRIVATE DEĞİŞKENİNE BAŞKA SINIFTAN ERİŞİLEMEYECEĞİNİ GÖSTERDİM.

METOD ADLANDIRMA

Java'da metod isimlendirme kuralları genel olarak şu prensiplere dayanır:

- 1-)camelCase İsimlendirme: Metod isimleri küçük harfle başlar ve sonraki kelimelerin baş harfleri büyük yazılır. (calculateInterest, getEmployeeName). Sınıf isimlerindeyse PascalCase kullanılmalıdır yani her kelime büyük harfle başlanmalı. (AccessModifiers)
- 2-) Tanımlayacağımız metotlara isim verirken sayı ile başlamamalıyız.
- 3-) Bir metotun içerisinde ayrı bir metot oluşturulamaz.
- 4-) Metod isimlerinde özel karakterler boşluklar vb olamaz.

JAVA METOT TÜRLERİ

Parametresiz Metot: Parametre almadan bir görevi gerçekleştiren metot.

Parametre Alan Metot: Bir veya daha fazla parametre alarak belirli bir işlemi gerçekleştiren metot.

Değer Döndüren Metot: İşlem sonucunda bir değer döndüren ve bu değer genellikle başka bir işlemde kullanılan metot.

Void Metot: İşlem sonucunda bir değer döndürmeyen metottur.

Birden Fazla Değer Döndüren Metot: Tek bir işlem sonucunda birden fazla değer döndürebilen metot.

Static Metot: Bir nesne örneği oluşturulmadan doğrudan sınıf adıyla çağrılabilen metot.

Recursive Metot: Kendi kendini çağırabilen ve genellikle bir döngü yerine kullanılan metot.

Overloaded Metot: Aynı isme sahip ancak farklı parametreler alan ve farklı işlemleri gerçekleştiren birden fazla metot.

Final Metot: Alt sınıflar tarafından yeniden tanımlanması veya değiştirilmesi engellenmiş olan metot. Override edilemezler.

METOT TÜRLERİ ÖRNEKLERİ

```
package ileri_java;

public class MethodTypes {

    // Parametresiz Metot
    public void parametresizMetot() {
        System.out.println("Bu bir parametresiz metottur.");
    }

    // Parametre Alan Metot
    public void parametreAlanMetot(int sayi) {
        System.out.println("Parametrelili metot, sayı: " + sayi);
    }

    // Değer Döndüren Metot
    public int degerDonenMetot() {
        return 42;
    }

    // Birden Fazla Değer Döndüren Metot
    public String[] birdenFazlaDegerDondurenMetot() {
        String[] degerler = {"Değer1", "Değer2", "Değer3"};
        return degerler;
    }

    // Static Metot
    public static void statikMetot() {
        System.out.println("Bu bir statik metottur.");
    }

    // Recursive Metot
    public void tekrarliMetot(int n) {
        if (n > 0) {
            System.out.println("Tekrarlı metot, n = " + n);
            tekrarliMetot(n - 1);
        }
    }

    // Overloaded Metot
    public void overloadedMetot() {
        System.out.println("Bu bir parametresiz overloaded metottur.");
    }

    public void overloadedMetot(int sayi) {
        System.out.println("Parametrelili overloaded metot, sayı: " + sayi);
    }

    // Final Metot
    public final void finalMetot() {
        System.out.println("Bu bir final metottur.");
    }

    public static void main(String[] args) {
        MethodTypes obje = new MethodTypes();

        // Parametresiz Metot çağırısı
        obje.parametresizMetot();

        // Parametre Alan Metot çağırısı
        obje.parametreAlanMetot(10);

        // Değer Döndüren Metot çağırısı
        int sonuc = obje.degerDonenMetot();
        System.out.println("Değer döndüren metottan dönen deger: " + sonuc);

        // Birden Fazla Değer Döndüren Metot çağırısı
        String[] degerler = obje.birdenFazlaDegerDondurenMetot();
        System.out.println("Birden fazla deger döndüren metottan dönen degerler:");
        for (String deger : degerler) {
            System.out.println(deger);
        }

        // Static Metot çağırısı
        statikMetot();

        // Recursive Metot çağırısı
        obje.tekarliMetot(5);

        // Overloaded Metot çağırısı
        obje.overloadedMetot();
        obje.overloadedMetot(20);

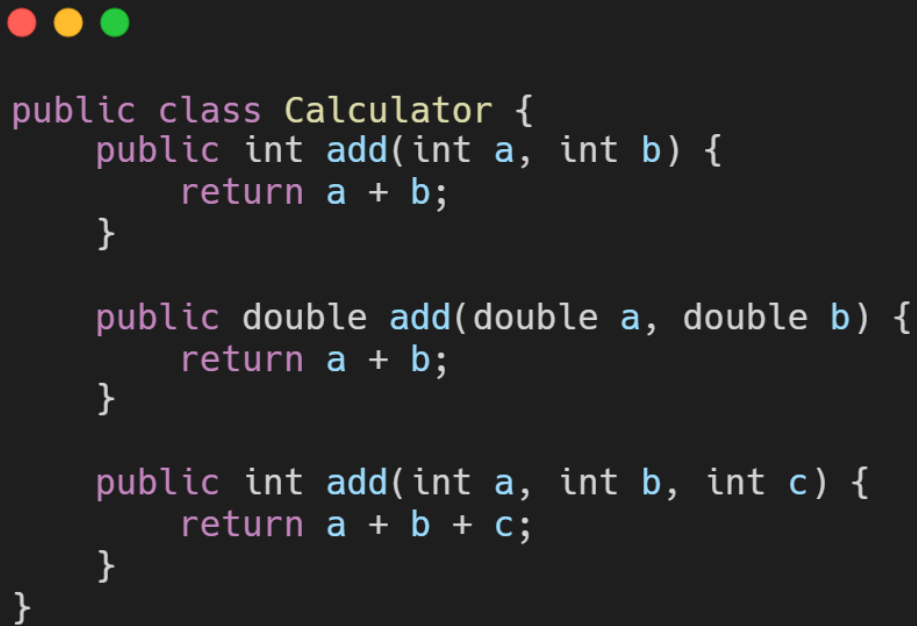
        // Final Metot çağırısı
        obje.finalMetot();
    }
}
```

ÜSTTEKİ KODDA TÜM METOT TÜRLERİNE BİRER ÖRNEK VERDİM. VE ÇIKTI AŞAĞIDAKİ GİBİ OLDU ;

```
Bu bir parametresiz metottur.  
Parametrelili metot, sayı: 10  
Deger döndüren metottan dönen deger: 42  
Birden fazla deger döndüren metottan dönen degerler:  
Değer1  
Değer2  
Değer3  
Bu bir statik metottur.  
Tekrarlı metot, n = 5  
Tekrarlı metot, n = 4  
Tekrarlı metot, n = 3  
Tekrarlı metot, n = 2  
Tekrarlı metot, n = 1  
Bu bir parametresiz overloaded metottur.  
Parametrelili overloaded metot, sayı: 20  
Bu bir final metottur.
```

METOTLARDA AŞIRI YÜKLEME (OVERLOADİNG)

Java'da aşırı yükleme (overloading), bir sınıfta aynı isme sahip birden fazla metodun tanımlanmasıdır. Ancak bu metodlar farklı parametre listeleriyle tanımlanır. Bu, aynı metodu farklı parametrelerle çağırabileceğiniz anlamına gelir.



```
public class Calculator {  
    public int add(int a, int b) {  
        return a + b;  
    }  
  
    public double add(double a, double b) {  
        return a + b;  
    }  
  
    public int add(int a, int b, int c) {  
        return a + b + c;  
    }  
}
```

Yukarıdaki örnekte, Calculator sınıfında add isimli üç farklı metod oluşturdum. İlk iki metod, toplama işlemi için int ve double türlerini alırken, üçüncü metod üç tane int parametre alır.

Aşırı yükleme, kodu daha modüler ve okunabilir hale getirebilir çünkü aynı işlev için farklı parametre türlerini işleyebilir. Bu şekilde, farklı parametre türlerini kullanarak aynı isimli metodları çağırabilirsiniz.

JAVADA METHOD OVERRIDING

Java'da bir alt sınıfta (subclass) üst sınıftan (superclass) gelen bir metodu değiştirmek için "override" (üzerine yazmak) işlemi kullanılır. Bu işlem, alt sınıfta aynı isimli ve imzaya (signature) sahip bir metod tanımlanarak gerçekleştirilir. Örnek:

```
class UstSinif {
    void selamla() {
        System.out.println("Merhaba, ben üst sınıftan geliyorum.");
    }
}

class AltSinif extends UstSinif {
    // Üst sınıftan gelen selamla() metodu override ediliyor.
    @Override
    void selamla() {
        System.out.println("Merhaba, ben alt sınıftan geliyorum.");
    }
}

public class Main {
    public static void main(String[] args) {
        UstSinif ust = new UstSinif();
        ust.selamla(); // Üst sınıftan gelen metodu çağırır.

        AltSinif alt = new AltSinif();
        alt.selamla(); // Alt sınıfta override edilmiş metodu çağırır.
    }
}
```

Bu kodda, UstSinif adında bir üst sınıf ve AltSinif adında bir alt sınıf tanımladım. AltSinif, UstSinif'tan kalıtım alıyor. selamla() metodu AltSinif içinde aynı isimle ve @Override anotasyonu ile tekrar tanımlanarak override ediliyor. Bu nedenle alt.selamla() çağrıldığında alt sınıftan gelen metot çalışıyor.

KONU SONU BASİT JAVA PROJEM (4 METOTLU HESAP MAKİNESİ)

```
package ileri_java;

public class Calculator {

    // Toplama metodu
    public static double add(double num1, double num2) {
        return num1 + num2;
    }

    // Çıkarma metodu
    public static double subtract(double num1, double num2) {
        return num1 - num2;
    }

    // Çarpma metodu
    public static double multiply(double num1, double num2) {
        return num1 * num2;
    }

    // Bölme metodu
    public static double divide(double num1, double num2) {
        if (num2 != 0) {
            return num1 / num2;
        } else {
            System.out.println("Bölen 0 olamaz!");
            return 0;
        }
    }

    public static void main(String[] args) {
        java.util.Scanner scanner = new java.util.Scanner(System.in);

        System.out.println("Birinci sayıyı girin: ");
        double num1 = scanner.nextDouble();

        System.out.println("İkinci sayıyı girin: ");
        double num2 = scanner.nextDouble();

        System.out.println("İşlem türünü seçin (+, -, *, /): ");
        char operation = scanner.next().charAt(0);

        double result = 0;

        switch (operation) {
            case '+':
                result = add(num1, num2);
                break;
            case '-':
                result = subtract(num1, num2);
                break;
            case '*':
                result = multiply(num1, num2);
                break;
            case '/':
                result = divide(num1, num2);
                break;
            default:
                System.out.println("Geçersiz işlem!");
                return;
        }

        System.out.println("Sonuç: " + result);
    }
}
```

TÜM KODLARIMIN YER ALDIĞI GİTHUB REPOSU :
https://github.com/serhatesen99/ileri_java_final