



Bölüm 3: Özyineleme

JAVA ile Nesne Yönelimli Programlama



Özyineleme (Recursion)

- Fonksiyonun görevini küçük parçalara bölerek kendini çağırması.
- Tekrarlama süreci, kendini benzer bir şekilde tekrar etme işlemidir.
- Bazı problemleri daha anlaşılır ve temiz bir şekilde çözmeye yarar.
- *Recursion refers to a procedure that calls itself, or to a constituent that contains a constituent of the same kind.*



Faktöriyel Alma

```
int faktoriyel(int n) {  
    if (n == 0 || n == 1) {  
        return 1;  
    } else {  
        return n * faktoriyel(n - 1);  
    }  
}
```



Faktöriyel Alma

faktoriyel(4)

$$= 4 * \text{faktoriyel}(3)$$

$$= 4 * (3 * \text{faktoriyel}(2))$$

$$= 4 * (3 * (2 * \text{faktoriyel}(1)))$$

$$= 4 * (3 * (2 * (1 * \text{faktoriyel}(0))))$$

$$= 4 * (3 * (2 * (1 * 1)))$$

$$= 4 * (3 * (2 * 1))$$

$$= 4 * (3 * 2)$$

$$= 4 * 6$$

$$= 24$$



Toplam Bulma

```
int toplam(int n) {  
  
    if (n == 1) {  
        return 1;  
    } else {  
        return n + toplam(n - 1);  
    }  
}
```



Fibonacci Serisi

- 0 ile başlar, ardından 1 gelir. $F(0) = 0$, $F(1) = 1$.
- Genel formül: $F(n) = F(n-1) + F(n-2)$

```
int fibonacci(int n) {  
    if (n <= 1) {  
        return n;  
    } else {  
        return fibonacci(n - 1) + fibonacci(n - 2);  
    }  
}
```



Lucas Serisi

- 2 ile başlar, ardından 1 gelir. $L(0) = 2$, $L(1) = 1$.
- Genel formül: $L(n) = L(n-1) + L(n-2)$

```
int lucas(int n) {  
    if (n == 0) {  
        return 2;  
    } else if (n == 1) {  
        return 1;  
    } else {  
        return lucas(n - 1) + lucas(n - 2);  
    }  
}
```



Catalan Serisi

- 1 ile başlar. $C(0) = 1$, $C(1) = 1$.
- Genel formül: $C(n) = (2n)! / ((n + 1)! * n!)$

```
long catalan(int n) {  
    if (n <= 1) {  
        return 1;  
    }  
    long sonuc = 0;  
    for (int i = 0; i < n; i++) {  
        sonuc += catalan(i) * catalan(n - i - 1);  
    }  
    return sonuc;  
}
```




Bell Serisi

- 1 ile başlar. $B(0) = 1, B(1) = 1$.
- Genel formül: $B(n) = \sum_{k=0}^{n-1} [C(n, k) * B(k)]$

```
int bell(int n) {
    if (n == 0) { return 1; }
    int sayi = 0;
    for (int k = 0; k < n; k++) {
        sayi += katsayi(n - 1, k) * bell(k);
    }
    return sayi;
}

int katsayi(int n, int k) {
    return faktoriyel(n) / (faktoriyel(k) * faktoriyel(n - k));
}
```



Özyinelemeli Fonksiyon Yazma

- **Kural 1** Temel Durum (Base Case):
 - Sonsuz döngüye girmemek için temel bir duruma ihtiyaç vardır.
 - Örneğin, n sayısı 0 veya 1 olduğunda fonksiyon sonlanmalı.
- **Kural 2** İlerleme Sağlama:
 - Her özyinelemeli çağrının, temel duruma doğru ilerlemesi gerekir.
 - Her özyinelemeli çağrıda parametre değerini azaltmak veya artırmak.
- Sonsuz özyineleme ve yetersiz ilerleme gibi hatalardan kaçınmalı.
- Her özyinelemeli çağrı problemin küçük bir alt kümesine odaklanmalı.



Rakamlar Toplamı

```
int rakamlarToplami(int n) {  
    if (n < 0) {  
        return rakamlarToplami(-n);  
    } else if (n < 10) {  
        return n;  
    } else {  
        return rakamlarToplami(n / 10) + (n % 10);  
    }  
}
```



Rakamlar Toplamı

$$\begin{aligned} & \text{rakamlarToplami}(-48729) \\ &= \text{rakamlarToplami}(48279) \\ &= \text{rakamlarToplami}(4827) + 9 \\ &= (\text{rakamlarToplami}(482) + 7) + 9 \\ &= ((\text{rakamlarToplami}(48) + 2) + 7) + 9 \\ &= (((\text{rakamlarToplami}(4) + 8) + 2) + 7) + 9 \\ &= (((4 + 8) + 2) + 7) + 9 \\ &= ((12 + 2) + 7) + 9 \\ &= (14 + 7) + 9 \\ &= 21 + 9 \\ &= 30 \end{aligned}$$



Palindrom Kontrolü

```
boolean palindromMu(String s) {  
    if(s.length() <= 1) {  
        return true;  
    }  
    else if(s.charAt(0) == s.charAt(s.length() - 1)) {  
        return palindromMu(s.substring(1, s.length() - 1));  
    }  
    return false;  
}
```



Palindrom Kontrolü

```
palindromMu("racecar")  
= ('r' == 'r') and palindromMu("aceca")  
= true and palindromMu("aceca")  
= palindromMu("aceca")  
= ('a' == 'a') and palindromMu("cec")  
= true and palindromMu("cec")  
= palindromMu("cec")  
= ('c' == 'c') and palindromMu("a")  
= true and palindromMu("a")  
= palindromMu("a")  
= true
```



En Büyük Ortak Bölen Bulma

```
int gcd(int x, int y) {  
    if(y == 0) {  
        return x;  
    }  
    return gcd(y, x % y);  
}
```



En Büyük Ortak Bölen Bulma

$$\begin{aligned} \gcd(444, 93) &= \gcd(93, 72) \\ &= \gcd(72, 21) \\ &= \gcd(21, 9) \\ &= \gcd(9, 3) \\ &= \gcd(3, 0) \\ &= 3 \end{aligned}$$



Verimlilik (Efficiency)

- Özyinelemeli fonksiyonlar bazen yavaş çalışabilir.
- Özyinelemeli çağrılar yapıldıkça, sistem kısmi sonuçları bellekte biriktirir.
- Sistem yığını aşırı dolarsa, program bellek taşma hatası verir.
- Problemlerle durumlarda çözüm, döngülerle çalışan ilerlemeli çözümlerdir.
- İlerlemeli çözümler, kısmi sonuçları biriktirmeye ihtiyaç duymaz.
- ilerlemeli = iterative



Faktöriyel Alma

```
int faktoriyel(int n) {  
    int carpim = 1;  
    for (int i = 1; i <= n; i++) {  
        carpim *= i;  
    }  
    return carpim;  
}
```



Rakamlar Toplamı

```
int rakamlarToplami(int n) {  
    int toplam = 0;  
    int kalan = Math.abs(n);  
    while (kalan > 0) {  
        toplam += kalan % 10;  
        kalan /= 10;  
    }  
    return (n > 0) ? toplam : -toplam;  
}
```



Palindrom Kontrolü

```
boolean palindromMu(String s) {  
    s = s.toLowerCase();  
    int uzunluk = s.length();  
    for (int i = 0; i < uzunluk / 2; i++) {  
        if (s.charAt(i) != s.charAt(uzunluk - 1 - i)) {  
            return false;  
        }  
    }  
    return true;  
}
```



Kuyruk Özyinelemesi (Tail Recursion)

- Fonksiyonun son işlemi özyineleme çağrısıdır.
- Belirli optimizasyonları mümkün kılar.
- Yığılmayan çağrılar sayesinde bellek verimliliği sağlar.
- Optimizasyon, fonksiyon çağrılarını bir döngü gibi işlemesine olanak tanır.



Rakamlar Toplamı

```
int rakamlarToplami(int sayi, int toplam) {  
    if (sayi == 0) {  
        return toplam;  
    } else {  
        return rakamlarToplami(sayi / 10, toplam + sayi % 10);  
    }  
}
```



Faktöriyel Alma

```
int faktoriyel(int n, int birikec) {  
    if (n == 0) {  
        return birikec;  
    } else {  
        return faktoriyel(n - 1, n * birikec);  
    }  
}
```



Durum Bilgisi Taşıma (Memoization)

- Tekrarlanan hesaplamaların sonuçlarını yeniden kullanma yöntemidir.
- Sonuçlar önbelleğe alınır ve ihtiyaç duyulduğunda kullanılır.



Fibonacci Serisi

```
Map<Integer, Long> hafiza = new HashMap<>();
```

```
long fibonacci(int n) {  
    if (hafiza.containsKey(n)) {  
        return hafiza.get(n);  
    }  
    if (n == 0 || n == 1) {  
        return n;  
    }  
    long sonuc = fibonacci(n - 1) + fibonacci(n - 2);  
    hafiza.put(n, sonuc);  
    return sonuc;  
}
```



Faktöriyel

```
Map<Integer, Long> hafiza = new HashMap<>();
```

```
long faktoriyel(int n) {  
    if (hafiza.containsKey(n)) {  
        return hafiza.get(n);  
    }  
    if (n == 0 || n == 1) {  
        return 1;  
    }  
    long sonuc = n * faktoriyel(n - 1);  
    hafiza.put(n, sonuc);  
    return sonuc;  
}
```



SON