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 factorial(int n) {

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

Faktöriyel Alma

factorial(4)

= 4 * factorial(3)

= 4 * (3 * factorial(2))

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

= 4 * (3 * (2 * (1 * factorial(0))))

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

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

= 4 * (3 * 2)

= 4 * 6

= 24

Toplam Bulma

```
int calculateSum(int n) {

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

Bell Serisi

- 1 ile başlar. B(0) = 1, B(1) = 1.
- Genel formül: B(n) = Σ (k=0 to n) [C(n, k) * B(k)]

```
int bell(int n) {
  if (n == 0) { return 1; }
  int bellNumber = 0;
  for (int k = 0; k < n; k++) {
    bellNumber += coefficient(n - 1, k) * bell(k);
  }
  return bellNumber;
}
int coefficient(int n, int k) {
  return factorial(n) / (factorial(k) * factorial(n - k));
}
```

Özyinelemeli Fonksiyon Yazma

- **Kural 1** Temel Durum (Base Case):
  - Fonksiyonun kendini sonsuz bir döngüye sokmaması için bir temel duruma ihtiyaç vardır.
  - Örneğin, n sayısı 0 veya 1 olduğunda fonksiyon sonlanmalı.
- **Kural 2** İlerleme Sağlama:
  - Her özyinelemeli çağrının, temel duruma doğru ilerlemesi gerekir.
  - Örneğin, 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 sumOfDigits(int n) {
  if (n < 0) {
    return sumOfDigits(-n);
  } else if (n < 10) {
    return n;
  } else {
    return sumOfDigits(n / 10) + (n % 10);
  }
}
```

Rakamlar Toplamı

sumOfDigits(-48729)
  = sumOfDigits(48279)
  = sumOfDigits(4827) + 9
  = (sumOfDigits(482) + 7) + 9
  = ((sumOfDigits(48) + 2) + 7) + 9
  = (((sumOfDigits(4) + 8) + 2) + 7) + 9
  = (((4 + 8) + 2) + 7) + 9
  = ((12 + 2) + 7) + 9
  = (14 + 7) + 9
  = 21 + 9
  = 30

Palindrom Kontrolü

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

Palindrom Kontrolü

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

gcd(444,93)

= gcd(93,72)

= gcd(72,21)

= gcd(21,9)

= gcd(9,3)

= gcd(3,0)

= 3

Verimlilik (Efficiency)

- Özyinelemeli fonksiyonlar bazen yavaş olabilir.
- Özyinelemeli çağrılar yapıldıkça, sistem kısmi sonuçları bellekte biriktirir.
- Sistem yığını aşırı dolarsa, program bellek taşma hatası verir.
- Problemli durumlarda çözüm, döngülerle çalışan iteratif çözümlerdir.
- İteratif çözümler, kısmi sonuçları biriktirmeye ihtiyaç duymaz.

Faktöriyel Alma

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

Rakamlar Toplamı

```
int sumOfDigits(int n) {
  int sum = 0;
  int remaining = Math.abs(n);
  while (remaining > 0) {
    sum += remaining % 10;
    remaining /= 10;
  }
  return (n > 0) ? sum : -sum;
}
```

Palindrom Kontrolü

```
boolean isPalindrome(String s) {
  s = s.toLowerCase();
  int length = s.length();
  for (int i = 0; i < length / 2; i++) {
    if (s.charAt(i) != s.charAt(length - 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 sumOfDigits(int number, int sumSoFar) {
  if (number == 0) {
    return sumSoFar;
  } else {
    return sumOfDigits(number / 10, sumSoFar + number % 10);
  }
}
```

Faktöriyel Alma

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

Durum Bilgisi Taşıma (Memoization)

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

Fibonacci Serisi

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

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

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

long factorial(int n) {
  if (memo.containsKey(n)) {
    return memo.get(n);
  }
  if (n == 0 || n == 1) {
    return 1;
  }
  long result = n * factorial(n - 1);
  memo.put(n, result);
  return result;
}
```

SON