Bölüm 6: Liste

Veri Yapıları

Liste Veri Yapısı

- Java List arayüzü, Java Collections Framework'ün bir üyesidir.
- Liste, birden çok öğeyi bir arada saklamak için kullanılan bir veri yapısıdır.
- Listeler, verilerin sıralı bir şekilde depolanmasını sağlar.

Liste Kavramları

- **Öğe (Element):** Listede depolanan her bir veri parçası.
- **İndeks (Index):** Liste içindeki her öğenin sırasını belirleyen sayısal değer. İndeks genellikle 0'dan başlar.
- **Boş Liste (Empty List):** Hiçbir öğe içermeyen bir liste.
- **Uzunluk (Length):** Listenin içinde bulunan öğe sayısı.
- **Dizi (Array):** Liste öğelerini depolamak için kullanılan veri yapısı.

Liste Özellikleri

- **Sıralıdır:** Listeler öğelerin eklenme sırasına göre sıralanır.
- **Değiştirilebilirdir (Mutable):** Öğeler eklenip çıkarılabilir, güncellenebilir.
- **İndeksleme (Indexing):** Her öğe, bir sayısal indeksle ulaşılabilir.
- **Döngülerle Kullanılabilir:** Liste öğeleri üzerinde döngülerle işlemler yapılabilir.

Liste İşlemleri

- **Ekleme (Append):** Yeni bir öğeyi listenin sonuna ekler.
- **Silme (Remove):** Belirli bir öğeyi listeden çıkarır.
- **İndeksleme (Indexing):** Belirli bir öğeye indeksle erişim sağlar.
- **Dilimleme (Slicing):** Liste içindeki bir aralığı seçer.
- **Uzunluk (Length):** Listenin öğe sayısını döndürür.

Karmaşıklık Analizi

- **Ekleme**: O(1)
- **Silme**: O(n)
- **İndeksleme**: O(1)
- **Dilimleme**: O(k)
- **Uzunluk**: O(1)

Java'da Liste (List) Veri Yapısı

- Liste (List), birden fazla öğeyi sıralı bir şekilde saklamak için kullanılır.
- Java'da, java.util paketi içindeki List arayüzünü veya bu arayüzü gerçekleyen sınıfları kullanarak listeler oluşturulabilir.

Liste Arayüzünün Ana Metodları

- Eleman Ekleme (Add)
  - **add(E e):** Liste sonuna bir eleman ekler.
  - **add(int index, E element):** Belirli bir indekse eleman ekler.
- Eleman Silme (Remove)
  - **remove(Object o):** Belirli bir elemanı listeden kaldırır.
  - **remove(int index):** Belirli bir indeksteki elemanı kaldırır.
- Eleman Erişim (Get)
  - **get(int index):** Belirli bir indeksteki elemanı döndürür.
- Liste Uzunluğu (Size)
  - **size():** Listenin uzunluğunu döndürür.
- Döngülerle Kullanım
  - Liste elemanları üzerinde döngülerle işlem yapabiliriz.

List Demo

- List, yinelemeli öğeler eklenmesine olanak tanır.

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

public class ListDemo {
  public static void main(String[] args) {
    List<String> isimler = new ArrayList<>();
    isimler.add("Ali");
    isimler.add("Ali");
    isimler.add("Ali");
  }
}
```

List Demo

- List, 'null' öğeler içerebilir.

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

public class ListDemo {
  public static void main(String[] args) {
    List<String> isimler = new ArrayList<>();
    isimler.add(null);
    isimler.add("Ali");
    isimler.add(null);
  }
}
```

List Demo

- List arayüzü, Java 8'de birçok varsayılan yöntem aldı.
- Bu varsayılan yöntemler, listeleri yönetmek için kullanılabilecek kullanışlı araçlardır.
- Örneğin,
  - replaceAll() listedeki tüm öğeleri başka bir öğeyle değiştirmek için,
  - sort() yöntemi, listedeki öğeleri sıralamak için,
  - spliterator() yöntemi, listeyi bölmek için kullanılabilir.

List Demo - replaceAll

```
List<String> isimler = new ArrayList<>();
isimler.add("Ali");
isimler.add("Ahmet");
isimler.add("Mehmet");

isimler.replaceAll(isim -> isim.toUpperCase());

System.out.println(isimler); // [ALI, AHMET, MEHMET]
```

List Demo - sort

```
List<Integer> sayilar = new ArrayList<>();
sayilar.add(5);
sayilar.add(3);
sayilar.add(1);

sayilar.sort((sayi1, sayi2) -> Integer.compare(sayi1,
sayi2));

System.out.println(sayilar); // [1, 3, 5]
```

List Demo spliterator

```
List<String> kelimeler = new ArrayList<>();
kelimeler.add("Java");
kelimeler.add("Python");
kelimeler.add("C++");

Spliterator<String> kelimeSpliterator = kelimeler.spliterator();

while (kelimeSpliterator.tryAdvance(kelime ->
System.out.println(kelime))) {
  // do something with the word
}
```

List Demo - get

- List indeksleri, diziler gibi 0'dan başlar.

```
List<String> isimler = new ArrayList<>();
isimler.add("Ali");
isimler.add("Ahmet");
isimler.add("Mehmet");

String ilkIsim = isimler.get(0);
System.out.println(ilkIsim);
```

List Demo - Generics

- List, Generics'i destekler.

```
List<String> isimler = new ArrayList<>();
isimler.add("Ali");

List<Integer> sayilar = new ArrayList<>();
sayilar.add(5);
```

Iterator Arayüzü

- Koleksiyonun elemanları üzerinde gezmeye izin veren bir arayüzdür.
- Iterator metodları
  - hasNext() - Koleksiyonda başka öğe olup olmadığını true döndürür, yoksa false döndürür.
  - next() - Koleksiyondaki bir sonraki öğeyi döndürür veya başka öğe yoksa NoSuchElementException fırlatır.

Iterator Demo

```
Iterator<String> isimlerIterator = isimler.iterator();

while (isimlerIterator.hasNext()) {
  String isim = isimlerIterator.next();
  System.out.println(isim);
}
```

ListIterator Arayüzü

- Collection arayüzü, verilerin bir koleksiyonunu temsil eder.
- ListIterator, List koleksiyonları için özel bir yineleyici arayüzüdür. Normal Iterator arayüzünün ötesinde aşağıdakiler gibi ek özellikler sağlar:
  - Bir List'de iki yönde yineleme yapma yeteneği.
  - Yineleme yaparken bir List'deki öğeleri ekleme, kaldırma ve değiştirme yeteneği.

ListIterator Metodları

- hasNext() - Listede başka öğe varsa true döndürür, yoksa false döndürür.
- next() - Listedeki bir sonraki öğeyi döndürür.
- previous() - Listedeki bir önceki öğeyi döndürür.
- add() - Listeye geçerli öğeden önce yeni bir öğe ekler.
- set() - Listedeki geçerli öğeyi yeni bir öğeyle değiştirir.
- remove() - Geçerli öğeyi listeden kaldırır.
- hasPrevious() - Listede bir önceki öğe varsa true, yoksa false döndürür.
- nextIndex() - Listedeki bir sonraki öğenin indeksi döndürür.
- previousIndex() - Listedeki bir önceki öğenin indeksi döndürür.

ListIterator Demo

```
ListIterator<String> isimlerIterator = isimler.listIterator();
// Listeyi ileri yönde yineleyin.
while (isimlerIterator.hasNext()) {
  String isim = isimlerIterator.next();
  System.out.println(isim);
}
// Listeyi ters yönde yineleyin.
while (isimlerIterator.hasPrevious()) {
  String isim = isimlerIterator.previous();
  System.out.println(isim);
}
```

Liste Arayüzünü Uygulayan Popüler Sınıflar

- ArrayList
- LinkedList
- Vector
- Stack
- CopyOnWriteArrayList
- Arrays.asList()

Liste Arayüzünü Uygulayan Popüler Sınıflar

- **ArrayList:**
  - İhtiyaca göre büyüyebilen veya küçülebilen dinamik dizi.
  - Öğelere hızlı rastgele erişim sağlar.
  - Sık sık ekleme veya silme gerektirmeyen senaryolar için uygundur.
- LinkedList:
- Vector:
- Stack:
- CopyOnWriteArrayList:
- Arrays.asList():

Liste Arayüzünü Uygulayan Popüler Sınıflar

- ArrayList:
- **LinkedList:**
  - Çift yönlü bağlı liste uygular, her öğe önceki ve sonraki öğelere bağlıdır.
  - Sık sık ekleme veya silme gerektiren senaryolar için uygundur.
  - Hızlı ekleme ve silme sağlar,
  - ancak ArrayList'e kıyasla rastgele erişim daha yavaştır.
- Vector:
- Stack:
- CopyOnWriteArrayList:
- Arrays.asList():

Liste Arayüzünü Uygulayan Popüler Sınıflar

- ArrayList:
- LinkedList:
- **Vector:**
  - ArrayList'e benzer, ancak senkronizedir (synchronized)
  - Çoklu iş parçacıklarında kullanıldığında güvenlidir.
  - Senkronizasyon nedeniyle performans sorunu yaşanabilir.
- Stack:
- CopyOnWriteArrayList:
- Arrays.asList():

Liste Arayüzünü Uygulayan Popüler Sınıflar

- ArrayList:
- LinkedList:
- Vector:
- **Stack:**
  - Yığın veri yapısını uygular, özel bir Liste uygulamasıdır.
  - Bir yığında kullanılan standart push ve pop işlemlerini destekler.
- CopyOnWriteArrayList:
- Arrays.asList():

Liste Arayüzünü Uygulayan Popüler Sınıflar

- ArrayList:
- LinkedList:
- Vector:
- Stack:
- **CopyOnWriteArrayList:**
  - Senkronizasyon yükü olmadan iş parçacıkları arası güvenlik sağlar.
  - Listenin sık gezildiği, nadiren değiştirildiği senaryolar için tasarlanmıştır.
  - Liste güncellendiğinde yeni bir kopya oluşturur,
  - Büyük listeler için hafıza ve performans açısından maliyetli olabilir.
- Arrays.asList():

Liste Arayüzünü Uygulayan Popüler Sınıflar

- ArrayList:
- LinkedList:
- Vector:
- Stack:
- CopyOnWriteArrayList:
- **Arrays.asList():**
  - Bir diziyi bir List'e dönüştürür.
  - Elde edilen List, sabit boyutludur ve değiştirilemez

ArrayList

- ArrayList, Java Koleksiyon Çerçevesi (Collection Framework) içinde uygulanan bir sınıftır.
- Dinamik bir dizi oluşturmak için kullanılır.
- Standart dizilere kıyasla;
  - Daha yavaş,
  - Boyutu dinamik olarak büyütülebilir.
  - Eleman eklemek veya çıkarmak kolaydır.
  - Elemanlarla daha fazla işlem yapma esnekliği sağlar.

ArrayList Kullanımı

- **add(E e):** Eleman ekleme
- **remove(int index):** Belirli bir indeksteki elemanı çıkarma
- **get(int index):** Belirli bir indeksteki elemana erişim
- **size():** Listenin uzunluğunu alma

LinkedList

- LinkedList, Java Koleksiyon Çerçevesi'nde uygulanan bir sınıftır.
- Bağlı liste veri yapısını uygular.
- Öğelerin ardışık konumlarda saklanmadığı bir lineer veri yapısıdır.
- Her öğe, **veri** ve **adres** kısmı olan ayrı bir nesnedir.
- Öğeler, işaretçi ve adresler kullanılarak birbirine bağlıdır.
- Her öğeye "düğüm" denir.
- Rastgele erişim performansı düşüktür, çünkü elemanlar bağlıdır ve indeksleme maliyetlidir.

LinkedList Kullanımı

- **add(E e):** Eleman ekleme
- **remove(int index):** Belirli bir indeksteki elemanı kaldırma
- **get(int index):** Belirli bir indeksteki elemana erişim
- **size():** Listenin uzunluğunu alma

Vector

- Vector, Java Koleksiyon Çerçevesi'nde uygulanan bir sınıftır.
- Büyüyebilen bir nesneler dizisi gerçekler.
- Dinamik bir dizi gerçeklediği için ihtiyaca göre büyür veya küçülür.
- Bir diziyi andırır, tamsayı indeks kullanılarak erişilebilen bileşenleri içerir.
- Concurrent (eşzamanlı) işlemler için uygun değildir.

Vector Kullanımı

- **add(E e):** Eleman ekleme
- **remove(int index):** Belirli bir indeksteki elemanı çıkarma
- **get(int index):** Belirli bir indeksteki elemana erişim
- **size():** Listenin uzunluğunu alma

Stack

- Stack, Java Koleksiyon Çerçevesi'nde uygulanan bir sınıftır.
- Stack sınıfı, vektör sınıfını genişletir ve Yığın (Stack) veri yapısını uygular.
- Temel işlem, son giren ilk çıkar (last-in-first-out) ilkesine dayanır.
- Geri alma (undo) işlemleri için kullanışlıdır.
- Çoğu işlem sadece yığının üstündeki elemanı etkiler.

Stack Kullanımı

- **push(E e):** Eleman eklemek
- **pop():** En üstteki elemanı kaldırmak
- **empty():** Yığın boş mu?
- **search(Object o):** Belirli bir elemanın indeksini bulma
- **peek():** En üstteki elemana erişmek

CopyOnWriteArrayList

- CopyOnWriteArrayList, Java Koleksiyon Çerçevesi'nde uygulanan bir sınıftır.
- "Yazarken Kopyala" (Copy-on-Write) stratejisini kullanır.
  - Veriler üzerinde değişiklik yapıldığında orijinal veriyi kopyalar ve işlemleri kopya üzerinde gerçekleştirir.
- Çoklu iş parçacıkları arasında güvenli bir şekilde kullanılabilir.
- Okuma işlemleri çok hızlıdır, herhangi bir kilitlenme veya senkronizasyon olmadan yapılır.
- Yazma işlemleri oldukça maliyetlidir, çünkü her yazma işlemi için verinin kopyası oluşturulur.

Arrays.asList()

- Arrays.asList(), Java'da bir dizi (array) nesnesini liste (list) türünde bir koleksiyona dönüştüren bir fonksiyondur.
- Dizi ve liste arasında verilerin paylaşıldığı bir arayüz sunar.
- Dizi kullanmanın avantajlarından yararlanırken koleksiyonların işlevselliğini elde etmek istediğimizde kullanılır.
- Verileri diziye ekledikten sonra, bu verileri daha fazla koleksiyon işlevselliği kullanmak için bir Liste'ye dönüştürmek istediğimizde kullanılır.

SON