Bölüm 6: Liste

Veri Yapıları

Iterable Arayüzü

- Koleksiyonların ve dizilerin üzerinde gezinmeyi sağlar.
- Bir nesnenin iterable (gezilebilir) olduğu durumda:
  - (1) Bu nesne için bir iterator alabilirsiniz,
  - (2) Bu nesne için bir spliterator alabilirsiniz,
  - (3) forEach metodunu kullanarak bu nesne üzerinde gezinebilirsiniz.
- `for (var x: a) {...}` ifadesi, sadece `a` bir dizi ise veya Iterable arayüzünü uygulayan bir sınıfsa çalışır.
- Iterator ile elemanlara sırasıyla erişim sağlanır.
- Spliterator, paralel işleme yeteneklerini destekler.

Collection Arayüzü

- Koleksiyonların temel işlevlerini standartlaştırır.
- Bir nesne Collection arayüzü'ne ait olduğunda:
  - (1) Bu nesneye öğe eklenebilir ve çıkartılabilir,
  - (2) Bu nesnenin boyutu ve boş olup olmadığı sorgulanabilir,
  - (3) Bir öğeyi içerip içermediği test edilebilir,
  - (4) Öğeleri akış (stream) üzerinden alınabilir,
  - (5) Öğeleri bir diziye aktarılabilir.

List Arayüzü

- List arayüzü, Java Collections Framework'ün üyesidir.
- java.util paketi içerisinde bulunur.
- List arayüzünü gerçekleyen sınıflar kullanılarak listeler oluşturulabilir.
- Liste, birden çok öğeyi bir arada saklamak için kullanılır.

Liste Arayüzü

- Liste, sıralı bir koleksiyon (aynı zamanda bir dizi) olarak bilinir.
- Döngü kullanılarak öğeler üzerinde işlemler yapılabilir.
- Öğelerin listede nereye ekleneceği kullanıcı kontrolündedir.
- Eklenen öğeler, belirli bir sırada listeye yerleştirilir.
- Kullanıcı, öğelere tam sayı indeksleriyle erişebilir.
- Liste içinde öğeleri arayabilir.

Liste Kavramları

- **Öğe (Element):** Listede depolanan, veri saklayan yapı.
- **İndeks (Index):** Her öğenin konumunu belirleyen sayısal değer.
- **Boş Liste (Empty List):** Hiçbir öğe içermeyen liste.
- **Uzunluk (Length):** Listenin içinde bulunan öğe sayısı.

Mutable ve Immutable Listeler

- Mutable (değiştirilebilir) ve immutable (değiştirilemez) listeler, veri yapılarındaki önemli bir ayrımı temsil eder.
- Değiştirilebilir listeler, içerdikleri öğelerin değiştirilebileceği listelerdir.
- Değiştirilemez listeler, bir kez oluşturulduktan sonra, listede değişiklik yapmak yerine bir kopyası oluşturulur.
- Fonksiyonel programlamanın temel mekanizmalarından biridir.
- "persistent data structure" kavramı ile ilişkilidir.
- Dayanıklı veri yapıları, değiştirilemez veri yapılarından türetilir.

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 içinde bir aralığı seçer.
- **Uzunluk (Length):** Listenin öğe sayısını döndürür.

Liste Arayüzünün Ana Metodları

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

List Demo

- Yinelemeli öğelerin 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

- '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 - 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> split = kelimeler.spliterator();

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

List Demo - get

- İndeksler 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);
```

Koleksiyonları Gezmek İçin Üç Yol

- (1) for döngüsü, koleksiyon elemanlarını indeks kullanarak gezinmek.

```
for (int i = 0; i < collection.size(); i++) {
  Element element = collection.get(i);
```

- (2) forEach metodu, koleksiyonlar üzerinde kullanılan bir fonksiyondur.

```
collection.forEach(System.out::println);
```

- (3) Iterator, koleksiyon üzerinde eleman eleman gezmek için kullanılır.

```
Iterator<Element> iterator = collection.iterator();
while (iterator.hasNext()) {
  Element element = iterator.next();
```

Iterator Arayüzü

- Koleksiyonda elemanlar üzerinde gezinmeyi sağlar.
- Iterator metodları:
  - hasNext(): Listede gezinirken sona gelinip gelinmediğini söyler.
  - next(): Listede gezinirken bir sonraki öğeye geçmeyi sağlar. Liste sonuna gelindiğinde NoSuchElementException fırlatır.

Iterator Demo

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

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

ListIterator Arayüzü

- ListIterator, List koleksiyonları üzerinde gezinmeyi sağlar.
- Iterator arayüzüne kıyasla ek özellikler sağlar:
  - İki yönde de gezinebilme.
  - Gezinme sırasında listeye ekleme, çıkarma ve güncelleme yapabilme.

ListIterator Metodları

- hasNext() - Listede gezinirken sona gelinip gelinmediğini söyler.
- next() - Listede gezinirken bir sonraki öğeye geçmeyi sağlar.
- hasPrevious() - Listede gezinirken başa gelinip gelinmediğini söyler.
- previous() - Listede gezinirken bir önceki öğeye geçmeyi sağlar
- add() - Listeye geçerli öğeden önce yeni bir öğe ekler.
- remove() - Geçerli öğeyi listeden çıkarır.
- set() - Listedeki geçerli öğeyi yeni bir öğeyle değiştirir.
- nextIndex() - Listedeki bir sonraki öğenin indeksini döndürür.
- previousIndex() - Listedeki bir önceki öğenin indeksini döndürür.

ListIterator Demo

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

Liste Arayüzünü Uygulayan Sınıflar

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

Liste Arayüzünü Uygulayan 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 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,
  - ArrayList'e kıyasla rastgele erişim yavaştır.
- Vector:
- Stack:
- CopyOnWriteArrayList:
- Arrays.asList():

Liste Arayüzünü Uygulayan Sınıflar

- ArrayList:
- LinkedList:
- **Vector:**
  - ArrayList'e benzer, ancak eşzamanlıdır (synchronized)
  - Çoklu iş parçacıklarında güvenle kullanılabilir.
  - Senkronizasyon sağlaması nedeniyle performans sorunu yaşanabilir.
- Stack:
- CopyOnWriteArrayList:
- Arrays.asList():

Liste Arayüzünü Uygulayan Popüler Sınıflar

- ArrayList:
- LinkedList:
- Vector:
- **Stack:**
  - Yığın veri yapısını uygular.
  - 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, listenin yeni bir kopyası oluşturulur.
  - Büyük listeler için bellek ve performans maliyetli olabilir.
- Arrays.asList():

Liste Arayüzünü Uygulayan Popüler Sınıflar

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

ArrayList

- Dinamik bir dizi oluşturmak için kullanılır.
- Standart dizilere kıyasla;
  - İşlemler uzun sürer.
  - Boyutu dinamik olarak büyütülebilir.
  - Öğe eklemek ve çıkarmak kolaydır.
  - Öğelerle daha fazla işlem yapma esnekliği sağlar.

ArrayList Kullanımı

- **add(E e):** Listeye eleman ekleme
- **remove(int index):** Verilen indeksteki elemanı listeden çıkarma
- **get(int index):** Verilen indeksteki elemana erişim
- **size():** Listenin uzunluğunu alma

LinkedList

- Bağlı liste veri yapısını uygular.
- Öğeler bellekte ardışık konumlarda saklanmaz.
- Her öğe, **veri** ve **adres** kısmı olan birbirinden farklı birer nesnedir.
- Öğeler, adres kısmı kullanılarak birbirine bağlanır.
- Her öğeye "düğüm" denir.
- Rastgele erişim performansı düşüktür. ??

LinkedList Kullanımı

- **add(E e):** Listeye eleman ekleme
- **remove(int index):** Verilen indeksteki elemanı listeden çıkarma
- **get(int index):** Verilen indeksteki elemana erişim
- **size():** Listenin uzunluğunu alma

Vector

- Boyutu dinamik olarak değişebilen bir dizi uygular.
- İhtiyaca göre büyüyebilir, küçülebilir.
- Diziyi andırır, öğelere indeks kullanılarak erişilebilir.
- Concurrent (eşzamanlı) işlemler için uygun değildir.

Vector Kullanımı

- **add(E e):** Listeye eleman ekleme
- **remove(int index):** Verilen indeksteki elemanı listeden çıkarma
- **get(int index):** Verilen indeksteki elemana erişim
- **size():** Listenin uzunluğunu alma

Stack

- Vektör sınıfını genişletir ve Yığın (Stack) veri yapısını uygular.
- Son giren ilk çıkar (last-in-first-out) ilkesine dayanarak işlem yapar.
- Geri alma (undo), yeniden uygula (redo) işlemleri için kullanışlıdır.
- İşlemler yığının en üstünde bulunan elemanı etkiler.

Stack Kullanımı

- **push(E e):** Yığına eleman ekler
- **pop():** Üstteki elemanı çıkarır
- **empty():** Yığının boş olup olmadığını söyler
- **search(Object o):** Verilen elemanın yığındaki indeksini döner
- **peek():** Yığının üstünde bulunan elemanı döndürür, yığından çıkarmaz.

CopyOnWriteArrayList

- "Yazarken Kopyala" (Copy-on-Write) stratejisini kullanır.
  - Listede değişiklik yapılacağında orijinal listeyi korur.
  - İşlemleri kopya liste üzerinde gerçekleştirir.
- Çoklu iş parçacıkları arasında güvenle kullanılabilir.
- Okuma çok hızlıdır, kilitlenme / senkronizasyon sorunları olmaz.
- Yazma maliyetlidir, her yazma işlemi için listenin kopyası oluşturulur.

Arrays.asList()

- Verilen diziyi, liste türünde bir koleksiyona dönüştürür.
- Dizi ve liste arasında verilerin paylaşıldığı bir arayüz sunar.
- Hem dizi avantajlarından hem koleksiyonların işlevselliğinden yararlanılır.

SON