İLERİ JAVA UYGULAMALARI FİNAL ÖDEVİ

Ad- Soyad: Abdulmelik OLAN
Numara: H5220069
MYO/YO/Fakülte/Enstitü (School): Meslek Yüksekokulu
Bölüm/Program (Dept.): Bilgisayar Programcılığı
Öğretim Elemanı/Instructor: Hüseyin KINAY

Koleksiyonlar Nedir?

Koleksiyonlar, birden fazla veriyi saklamak ve yönetmek amacıyla kullanılan veri yapılarıdır. Programlama dillerinde veri gruplarını tek bir yapıda tutmak ve bu veriler üzerinde çeşitli işlemler yapmak için kullanılırlar. Koleksiyonlar, verilerin organize edilmesi, erişimi, güncellenmesi ve manipülasyonu için gerekli araçları sağlar. Büyük veri kümeleri ile çalışırken, koleksiyonların etkin kullanımı performans ve kod kalitesi açısından kritik öneme sahiptir. Map, kolaksiyonların bir parçası değildir. Fakat aralarında büyük oranda benzerlikler bulunduğu için genellikle birlikte ele alınır.

Koleksiyon Türleri Nelerdir?

1-List

A) ArrayList

B) LinkedList

C) Vectör

2-Sets

A) HashSet

B)TreeSet

C) LinkedHashSet

3- StortedSet

4-Map

A) HashMap

B) TreeMap

C) LinkedHashMap

5-Stacks

6- Queue

A) LinkedList

B) PriorityQueue

Listeler (List)

Listeler, sıralı veri kümelerini saklayan koleksiyon türleridir. Tekrarlayan elemanlara sahip olabilir. ArrayList, LinkedList ve Vector sınıflarıdır.

List Arabiriminin metotları:
boolean add(indeks, eleman): Listeye, parametrede indeksi verilen elemanının o indekse eklenmesini sağlar.
boolean addAll(indeks, koleksiyon): Listeye, parametrede verilen indeksten başlayarak parametrede verilen koleksiyonun eklenmesini sağlar.
int get(indeks): Listenin, parametre indeksi verilen elemanı döner.
int indexOf(eleman): Parametrede verilen elemanın listedeki ilk indeksini döner.
int lastindexOf(eleman): Parametrede verilen elemanın listedeki son geçtiği indeksini döner.
boolean remove(index): Parametrede verilen indeks ait elemanı çıkarır.
boolean set(index, eleman): Parametrede verilen indekse verilen elemanı atar.

subList(baslangic index, bitis index): Parametrede verilen indeks aralığına ait bir liste döner.

a)ArrayList: İçeriği ve boyutu program çalışırken değiştirilebilen veri yapılarıdır. Sabit boyutlu dizilerin aksine, dinamik listeler eleman ekleme, silme gibi işlemlerle büyüyebilir veya küçülebilir.

Örnek1:

```
List<String> arrayList = new ArrayList<>();
 arrayList.add("eleman1");
 arrayList.add("eleman2");
```

Örnek2:

```
 import java.util.ArrayList;


public class Main {
  public static void main(String[] args) {
    ArrayList<Integer> myList = new ArrayList<>();
    myList.add(1);     // Listeye eleman ekleme
    myList.add(2);
    myList.add(3);
    myList.add(4);
myList.remove(Integer.valueOf(2)); // Listeden eleman silme
System.out.println(myList);      // [1, 3, 4]
  }
}
```

Örnek olarak verilebilir.

b**) LinkedList:** LinkedList bir bellek alanında 3 adet bilgi tutar. Listede tutulacak eleman, önceki elemanın tutulduğu bellek alanının referansı ve sonraki elemanın tutulduğu bellek alanının referansı. Bu yüzden tüm elemanlar birbirine zincir gibi bağlıdır. Çift yönlü bağlı liste uygulamasıdır. Ekleme ve çıkarma işlemleri hızlıdır (liste başı ve sonunda), ancak rastgele erişim yavaştır.

```
//LinkedList olusturma
List<String> linkedList = new LinkedList<>();
```

LinkedList Sınıfına ait metotlar;
boolean addFirst(eleman): Parametrede verilen elemanı listenin başına ekler.
boolean addLast(eleman): Parametrede verilen elemanı listenin sonuna ekler.
boolean removeFirst(eleman): Listenin başındaki elemanı döner ve listeden çıkarır.
boolean removeLast(eleman): Listenin sonundaki elemanı döner ve listeden çıkarır.
void getFirst(): Listenin başındaki elemanı döner.
void getLast(): Listenin sonundaki elemanı döner.

C)**Vectör:** Vector, Java'nın eski bir sınıfıdır. ArrayList ile çok benzerdir. Fakat farklılıkları da bulunur. Vector senkronize bir yapıya sahiptir. Yani aynı anda birden fazla thread (iş parçacığı) tarafından erişilen durumlarda güvenliği sağlar. Senkronizasyon maliyetine sahip olduğu için de ArrayList'e göre daha yavaş olabilir, maliyetlidir ve performansı etkiler.

- std::vector (C++)

```
#include <vector>

#include <iostream>


int main() {

   std::vector<int> my_vector = {1, 2, 3};

my_vector.push_back(4);   // Vektöre eleman ekleme
```

```
my_vector.erase(my_vector.begin() + 1);  // Vektörden eleman silme

  for (int i : my_vector) {

    std::cout << i << " ";  // 1 3 4

  }
```

Kümeler (Sets)

Set, tekrarlı elemanlara izin vermeyen bir koleksiyon türüdür. En yaygın set türleri HashSet ve TreeSet'tir. Kümeler üzerinde eleman ekleme, çıkarma, birleştirme ve kesişim alma gibi işlemler gerçekleştirilir.

- **HashSet**: Hızlı erişim sağlar, elemanların sıralanmasını garanti etmez.

```
HashSet<String> ornek = new HashSet<String>();
```

- **TreeSet**: TreeSet tekrarsız elemanları “alfabetik” veya “küçükten büyüğe”(Natural Order) dizer. TreeSet, diğer set’lere göre çok daha yavaştır ve null eleman kabul etmez. SortedSet bir interface, TreeSet ise bir class’tır.

```
TreeSet<String> set=new TreeSet<String>();
```

- **LinkedHashSet:** LinkedHashSet, HashSet ile oldukça benzer bir yapıya sahiptir. Tek farkı, elemanları ekleniş sırasına göre dizmesidir. LinkedHashSet, HashSet’e nazaran daha yavaş çalışır.

```
LinkedHashSet<String> lhs = new LinkedHashSet<String>();
```

Java'da HashSet örneği:

```
Set<String> set = new HashSet<>();

set.add("Elma");
```

```
set.add("Elma"); // Tekrarlı eleman eklenmez

System.out.println(set);
```

Sıralı Kümeler (SortedSets)

Sıralı kümeler, elemanları belirli bir sıralamaya göre saklayan setlerdir.

Sortedset Metotları;

- Add: Bu sınıfa veri eklemek için Add() metodu kullanılır. Bu metod bool bir değer dönderir. Eğer eklenecek veri listede varsa metod false(yanlış)dönderir ve veriyi listeye eklemez, yoksa true(doğru) dönderir ve veriyi listeye ekler.
- GetViewBetween: SortedSet sınıfının GetViewBetween() metodu başka bir sınıfa veya diziye belli bir aralıktaki elemanları gönderir.
- Remove: SortedSet sınıfının elemanlarını siler.
- Union: SortedSet sınıflarının elemanlarının birleşimini alır. Aynı elemanlar varsa tekrarlama olmaz.
- Intersection: SortedSet sınıflarının elemanlarının kesişimini alır.
- Except: İki SortedSet sınıfı arasında birincinin diğerinde olmayan elemanlarını alır.

Map

Map, koleksiyonlara her ne kadar benzese de verileri saklama şekliyle koleksiyonlardan ayırt olmuş bir interface'dir. Koleksiyonlar verileri liste veya dizi şeklinde depoluyordu. Fakat Map verileri *key, value(anahtar, değer) çiftleri olarak tutar*.

Metotlar;

size(): Eşleme tablosundaki eşleme sayısını döner.
values(): Eşleme tablosundaki elemanlardan bir koleksiyon nesnesi döner.
clear(): Map'teki tüm eşleşmeleri kaldırır.
isEmpty(): Eşleşme tablosu boş ise yani eşleşme tablosunda eleman yoksa true döner.

entrySet(): Eşleşme tablosunda bulunan elemanlar için bir set nesnesi-küme döner.
keySet(): tabloda bulunan anahtarlar için set nesnesi döner.
put(anahar,eleman): Parametrede verilen anahtar elemanı birbirleri ile eşleştirip, eşleme tablosuna yazar.
putAll(eslemeTablosu X): Parametrede verilen eş. tab. X in tüm elemanlarını ilgili eşleme tablosuna ekler.
get(anahtar): Parametrede verilen anahtar ile eşlesen elemanı döner.
containsKey(anahtar): parametrede verilen anahtar ile eşlesen bir eleman varsa true döner.
containsValue(eleman): parametrede verilen anahtar ile eşlesen bir eleman varsa true döner.
remove(anahtar): parametrede verilen değeri eşleme tablosundan çıkarır.

- HashMap: Genellikle en yaygın kullanılan map türüdür. Anahtar-değer çiftlerini depolamak için bir hash tablosu kullanır. Anahtarlar benzersiz olmalıdır ve değerler null olabilir.

```
Map<String, Integer> map = new HashMap<>();

map.put("Elma", 1);

map.put("Armut", 2);

System.out.println(map);
```

- TreeMap: Bu, anahtarları doğal bir sıraya göre saklar. Bu sıra, anahtarların karşılaştırılabilir olduğu herhangi bir tür olabilir. TreeMap, anahtarların sıralı bir şekilde depolanması gerektiğinde oldukça kullanışlıdır.

```
//TreeMap olusturma
Map<String, Integer> treeMap = new TreeMap<>();
```

- LinkedHashMap: LinkedHashMap, *HashMap* sınıfını *miras alır*. Dolayısıyla özellikleri oldukça benzerdir. Farklı olarak elemanları **sıralı** olarak ekler. HashTable ile verileri depolar ve linked list kullanarak eleman ekleme sırasını korur. *LinkedHashMap*, verileri hem hash tablosunda hem de linked list'te

depolar. Bu sayede eklenen elemanların sırasını korur. Eğer sıra önemli değilse HashMap kullanılması performans açısından daha iyidir.

```
//LinkedHashMap olusturma
Map<String, Integer> linkedHashMap = new LinkedHashMap<>();
```

StortedMap Arabirimi

Map arabirimindeki tüm metotları uygulamakla birlikte, map arabirimine ek olarak eşleme tablosundaki eşleme anahtarlarını sıra ile tutar.

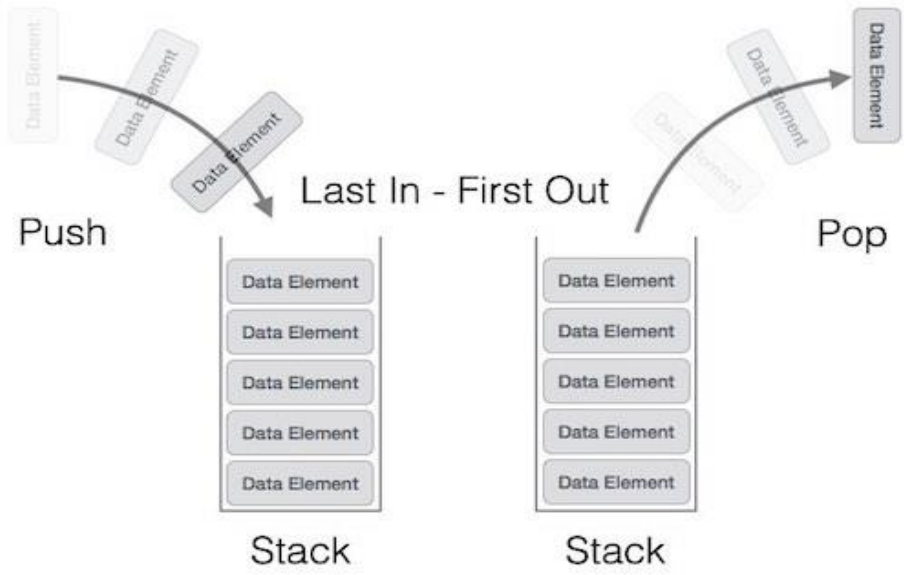
İlgili metotları;

firstKey(): Eşleme tablosundaki en küçük elemanı döner.
lastKey(): Eşleme tablosundaki en büyük elemanı döner.
HeadMap(anahtar): Eşleme tablosunun, parametrede verilen anahtardan küçük olan anahtarlar bölümünü döner.
tailMap(eleman): Eşleme tablosunun, parametrede verilen anahtardan büyük olan anahtarlar bölümünü döner.

Stack

Nesnelerin LIFO (last-input-first-output) yapısıyla depolanmasını sağlayan bir veri tipidir. Bilgisayar uygulamalarında çok sık kullanılır. Üst üste konulmuş kutular gibidir. Yani gelen kutu en üste konur.Bazı metotlar;

empty(): stack boş ise true, değilse false döner.
peek(): Stack'ın son eklenen elemanını döner.
pop(): En son elemanı döner ve listeden çıkartır.
push(eleman): Parametrede verilen elemanı stack'in sonuna ekler.
search(): Parametrede verilen elemanın stack'teki yerini döner.



Queue

Queue, Türkçe karşılığına baktığımızda *kuyruk'a* karşılık gelir. Elemanları *FIFO (first in first out)* mantığına göre sıralı bi şekilde depolar. Yani ilk eklenen eleman ilk çıkar. Tam olarak bir müşteri kuyruğu olarak düşünebiliriz.

Bu sınıfa ait metotlar**;**
add(eleman): Parametrede verilen elemanı kuyruğa ekler. İşlemin başarısız olması durumunda hata fırlatır.
offer(eleman): Parametrede verilen elemanı kuyruğa ekler. İşlemin başarısız olması durumunda null döner.
poll(): Kuyruğun başındaki elemanı kuyruktan çıkartır.
peek(): Kuyrukta sıradaki elemana ulaşmak için kullanılır.

LinkedList: LinkedList'in özel bir durumu var. Hem List hem de Queue arayüzlerini uygular. Queue olarak kullanıldığında FIFO sırasını korur.

LinkedList'in bu çift yönlü özelliği, uygulamalarda dinamik verileri listelemek ve Queue operasyonları için verileri sırasal bir şekilde işlemek için kullanılmasını sağlar. Listelerde ve Queue'larda birden çok işlem yapmanın gerekli olduğu senaryolarda LinkedList kullanmak, performans ve esneklik açısından faydalı olabilir. Ancak, çok büyük veri kümelerinde indeks tabanlı erişim hızı ArrayList'e göre daha düşük olabilir. Bu yüzden uygulamanın ihtiyaçlarına ve veri yapısının özelliklerine dikkat etmek önem gerektirir.

```
//LinkedList'in queue ile olusturulmasi
Queue<String> queue = new LinkedList<>();
```

PriorityQueue: Elemanlarını önceliklerine göre sıralar. Bunun için *doğal sıralama* kullanır. Elemanlar, FIFO'ya göre değil önceliklerine göre kuyruktan çıkarılır. Yüksek, orta ve düşük öncelikli elemanları sıralayıp önce yüksek öncelikli elemanı çıkarır.

```
//PriorityQueue olusturulmasi
Queue<String> priorityQueue = new PriorityQueue<>();
```

Koleksiyonlar'a Ait Örnek Kod:

```
import java.util.ArrayList;

import java.util.Scanner;


public class Main {

    public static void main(String[] args) {

        ArrayList<String> todoList = new ArrayList<>();

        Scanner scanner = new Scanner(System.in);

        int choice;

        do {

            System.out.println("1. Add To-Do Item");

            System.out.println("2. List To-Do Items");

            System.out.println("0. Exit");

            System.out.print("Enter your choice: ");

            choice = scanner.nextInt();

            scanner.nextLine(); // consume newline
```

```
            switch (choice) {
                case 1:
                    System.out.print("Enter the description of the to-do item: ");
                    String description = scanner.nextLine();
                    todoList.add(description);
                    break;
                case 2:
                    System.out.println("To-Do List:");
                    if (todoList.isEmpty()) {
                        System.out.println("No items in the list.");
                    } else {
                        for (int i = 0; i < todoList.size(); i++) {
                            System.out.println((i + 1) + ". " + todoList.get(i));
                        }
                    }
                    break;
                case 0:
                    System.out.println("Exiting...");
                    break;
                default:
                    System.out.println("Invalid choice. Try again.");
            }
```

```
        } while (choice != 0);

        scanner.close();

    }

}
```

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package javacollectionframework.collections.list.arraylist;

import java.util.ArrayList;
import java.util.Collections;
import java.util.Iterator;
import java.util.List;

/**
 *
 * @author cemdirman
 */
public class TestArrayList {

    public static void main(String[] args) {
        List sayilar = new ArrayList();
        sayilar.add(1905);
        sayilar.add(1992);
        sayilar.add(23);
        sayilar.add(3);
        sayilar.add(47);
        sayilar.add(57);
        sayilar.add(5, 24);

        Iterator iterator = sayilar.iterator();
        iterator.forEachRemaining(System.out::println);

        System.out.println("--------Sıralı----------");
```

```
        Collections.sort(sayilar);
        for (Iterator it = sayilar.iterator(); it.hasNext();) {
            System.out.println(it.next());
        }

        System.out.println("--------Örnek 2----------");

        List<Character> harfler = new ArrayList();
        harfler.add('A');
        harfler.add('B');
        harfler.add('C');
        harfler.add('D');
        harfler.add('E');
        harfler.add('F');
        harfler.add('G');

        for (Iterator<Character> it = harfler.iterator(); iterator.hasNext();) {
            System.out.println(it.next());
        }
        System.out.println("------------------------------");
        Iterator<Character> iterator1 = harfler.iterator();
        iterator1.forEachRemaining(System.out::println);
    }
}
```