Java Veri Yapıları

Giriş

- Veri yapıları dersi olmadığı için çok üstünde durmayacağız, ancak Java dilinde en azından bilindik veri yapılarının nasıl tanımlandığını ve kullanıldığını görelim.
- Linear (doğrusal) ve non-linear yapılar
  - Dizi, Stack, Queue, LinkedList
  - Trees, Graphs, Heap, HashSet, HashMap
- Bunların hepsini görmeyeceğiz.

Diziler

- Python'un aksine **static typed** bir dil olduğu için burada dizi statik bir yapı.
  - Yani içinde kaç eleman olduğunu söyleyerek tanımlamak gerekiyor.
  - `String[] dizi = new String[4];`
  - `int[] dizi = new int[4];`
  - `int[] dizi = {1,2,3,4};`
  - `String[] dizi = {"a","b"};`
  - `int dizi[] = new int[4];` gibi de tanımlanabilir.

Diziler

- Bellekte sıralı olduğu için **sort** işlemi kolaydır.
- Rastgele herhangi bir elemana rahatlıkla erişilebilir.
- Değer düzenlemek kolaydır.
    - Ancak yeni değer ekleyemeyiz.
    - Dizi oluştururken kaplayacağı yeri söyleriz, ona o kadar yer ayrılır.

Dezavantajlar

- Eleman eklemek ve silmek zordur.
- Dizinin boyutunu iyi hesaplamadıysak boşuna yer işgal edebilir.

Örnek

- Bir dizi oluşturup 1-10 arasındaki sayıları ekleyelim.

Stack

- LIFO: Last in First Out (Son giren ilk çıkar)
- Yalnızca en tepe elemana ulaşabiliriz. Diğerlerine ulaşamayız.
- Ekleme ve çıkarma yalnızca tepeden olur.
  - Üst üste konmuş sandalyeler gibi düşünün.
- 4 tane operasyonu var:
  - push(elm): eleman ekler
  - pop(): tepe elemanı cıkartır
  - isEmpty(): boş mu diye bakar
  - peek(): tepe elemanı gösterir, ama stack'ten çıkarmaz.

Stack kullanımı

```
Stack<Integer> myStack = new Stack<Integer>();
        //pop, push, seek, search

myStack.push(1);
System.out.println(myStack);
myStack.push(2);
System.out.println(myStack);
myStack.pop();
System.out.println(myStack);
System.out.println(myStack.empty());

System.out.println(myStack);
myStack.peek();
System.out.println(myStack);
System.out.println(myStack.search(1));
```

Queue

- FIFO: First in First Out (İlk giren ilk çıkar)
  - Bir bilet kuyruğu düşünülebilir
- Eklemeler **sondan** yapılır, çıkarmalar **baştan**.
- 4 operasyon
  - enqueue(elm): eleman ekler
  - dequeue(): tepe elemanı çıkarır
  - peekfirst(): tepe elemana bakar
  - peeklast(): son elemana bakar
- Java'da Queue implement etmek için birden fazla yol vardır, ancak onun detaylarına girmeyeceğiz.

Hashmap

- Key, value ikilileri olarak verileri tutuyor.
- Index vererek ulaşabiliyoruz.
- Daha sonra bunların kullanımları ile ilgili detaylı bilgiler vereceğiz.

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

map.put("harry", 10);
map.put("ron", 20);
map.put("hermione", 30);

System.out.println("Size is: " + map.size());
System.out.println(map);
Integer a = map.get("harry");
System.out.println(a);
```

Array Operasyonları

- Dizilerden elemanlara ulaşmak için index'leri kullanıyoruz. `myArray[2]`
- Array uzunluğu için `myArray.length`
- Bir array'i kopyalamak istersek;
  - `int[] myArr = oldArr.clone();`
- `Arrays` isminde bir util'i var Java'nın. Onu da kullanabiliriz.
  - Bunu ekledikten sonra noktaya basarak gelenlere bakabiliriz. (Çok fazla var.)
  - toString() : Array'i direkt olarak yazdırır
    - `Arrays.toString(myArr);`
  - fill(): `Arrays.fill(myArr, 0);` Tüm diziyi 0 larla doldurur (int array ise)

Dinamik Array (ArrayList)

- Java'da düz Array'ler statik olduğu için, bazen dinamik array kullanmak isteriz.
- O zaman ArrayList kullanacağız.
- Dizilerdeki operasyonların dışında ek operasyonlar var.
    - Python'daki list'i böyle düşünebiliriz.
- `ArrayList<String> myList = new ArrayList<String>();`

ArrayList operasyonları

```
ArrayList<String> myList = new ArrayList<String>();
myList.add("elma");
myList.add("armut");
System.out.println(myList);
```

```
for(String meyve:myList) {
    System.out.println(meyve);
}
```

ArrayList

- Dizilerde index verdiğimiz gibi vermiyoruz. Burada daha farklı
  - `myArr[1]` yerine `myList.get(1)`
- Eğer belirli bir yere bir veri eklemek istersek o zaman da şöyle yapacağız
  - `myList.set(1, "portakal");`
- Sort etmek için Collections kullanıyoruz
  - `Collections.sort(myList)`