Bölüm 1: Giriş

Veri Yapıları

Dijital veri | Veri tipleri | Java bellek modeli | Jenerikler | Soyut veri tipleri | Veri yapısı seçimi | Temel veri yapıları

Sercan KÜLCÜ · Giresun Üniversitesi · Bilgisayar Mühendisliği

GİRİŞ

Öğrenme Çıktıları

Temel kavramlar

Veri tipi, soyut veri tipi ve veri yapısı arasındaki farkı açıklamak.

Yapı seçimi

Bir problem için uygun veri yapısını gereksinimlere göre seçmek.

Java ile ADT

Bir ADT'yi arayüz (interface), bir veri yapısını sınıf olarak yazmak; referans ve jenerikleri doğru kullanmak.

Temel yapılar

Temel veri yapılarını düzenleme biçimleri ve tipik kullanımlarıyla tanımak.

Dijital Veri (Digital Data)

- Sayı, metin, görüntü ve ses gibi bilgilerin elektronik ortamda 0 ve 1'lerle temsil edilmiş biçimidir.
- **Örnekler**: Metin belgeleri, resim dosyaları, veritabanı, ses kayıtları.



Analog ses → ADC → 0 ve 1'ler → bilgisayar → DAC → analog ses

DİJİTAL VERİ

Dijital Verinin Özellikleri

Sayısal Olması

Veriler 0 ve 1 (ikili) olarak temsil edilir.

`1011 0010`

Kolay Saklama

Elektronik ortamda saklanabilir ve kolayca paylaşılabilir.

`disk · bulut · ağ`

İşlemeye Uygun

Bilgisayarlar üzerinde hızlı bir şekilde işlenebilir.

`~10⁹ işlem / saniye`

Bilgisayar yalnızca 0 ve 1'leri tanır. Veri yapıları, bu bitleri **anlamlı ve verimli** bir düzende tutmanın yollarıdır.

DİJİTAL VERİ

Dijital Verinin Temsil Biçimleri

Metin

ASCII, Unicode gibi karakter kodlamaları.

`'A' → 65 → 01000001`

Sayısal

Tam sayılar, kayan nokta sayıları.

`13 → 00001101 · 3,14 → IEEE 754`

Görüntü

Piksellerin renk değerleri.

`kırmızı piksel → (R, G, B) = (255, 0, 0)`

Ses

Ses dalgalarının dijital örneklemi.

`CD kalitesi → saniyede 44.100 örnek`

Dijital Veri Tipleri

Tür	Açıklama	Örnek
Yapısal	Belirli bir şemaya göre düzenlenmiş veri	İlişkisel veritabanı tabloları, CSV
Yarı Yapısal	Etiketlerle kısmen düzenlenmiş veri	XML, JSON
Metin	İnsanlar tarafından okunabilen metinler	TXT, e-posta
Resim ve Grafik	Görsel veriler	JPG, PNG
Ses ve Müzik	Ses kayıtları	MP3, WAV

VERİ YAPILARINA GİRİŞ

Veri Yapıları ve Algoritmalar

Algoritma

- Belirli bir görevi gerçekleştirmek için adım adım hazırlanan bir plan veya talimatlar bütünüdür.
- Bilgisayar biliminde temel bir yapı taşıdır.

Veri Yapısı

- Verilerin organize edilmesini sağlayan ve depolanma biçimlerini tanımlayan yapılardır.
- Verilerin etkili yönetimi ve işlenmesi için temel altyapıyı sağlar.

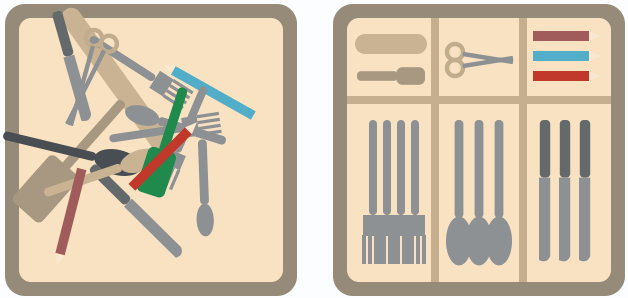
- Etkili algoritmalar ve veri yapıları, yazılımın verimli çalışmasını sağlar.
- Problemlere sistemli ve etkili çözümler sunar.

Algoritmalar + Veri Yapıları = Programlar

Niklaus Wirth'in 1976 tarihli kitabının adı

Veri Yapıları

- Verileri bilgisayarda düzenlemenin ve saklamanın bir yoludur.
- Temel amacı, verilere hızlı ve etkili bir şekilde erişmeyi sağlamaktır.
- Veri yapılarını anlamak, yazılım geliştirme sürecinde çok önemlidir.
- Doğru veri yapısı seçimi, programın performans ve verimliliğini artırır.

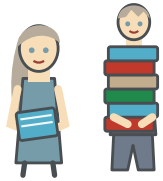


Aynı eşyalar: solda dağınık, sağda düzenli.
Aradığını hangisinde daha hızlı bulursun?

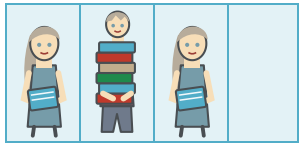
Temel Yapı Taşları

- **Nesneler**: Nitelikleri, isimlendirilmiş alanlara sahip tek bir birimde gruplar.
- **Diziler**: Varlıkları, bellekte ardışık yuvalarda toplu olarak saklar.
- **Bağlantılar**: Varlıklar arasında isteğe bağlı yönlü ilişkilere olanak tanır.

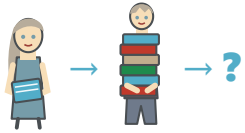
Nesneler



Diziler



Bağlantılar



İlkel (Primitive) Veri Tipleri

Java tipi	Boyut	Değerler
`byte`	8 bit	−128 … 127
`short`	16 bit	−32.768 … 32.767
`int`	32 bit	yaklaşık ±2,1 milyar
`long`	64 bit	yaklaşık $\pm 9{,}2 \cdot 10^{18}$
`float`	32 bit	ondalıklı (≈7 basamak)
`double`	64 bit	ondalıklı (≈15 basamak)
`boolean`	—	true / false
`char`	16 bit	UTF-16 kod birimi

- Tam sayılar, ondalıklı sayılar, mantıksal değerler ve karakterler.
- Basit ve temel veri tipleridir.
- Bellekte sabit bir boyuta sahiptirler.
- İşlemci tarafından doğrudan işlenebilirler.

Veri Tipleri

- **Veri Tipi**: Bir değer kümesini ve davranışlarını tanımlar.
- **Değer Kümesi**: Bir veri tipinin alabileceği geçerli değerler.
- **Davranışlar**: Değerler üzerinde gerçekleştirilebilecek işlemler.
- Her işlem her türde geçerli değildir.
- **Tip Güvenliği**: Veri tipleri, hataları en aza indirmek için kullanılır. Kısıtlamalar, programcılara neyin mümkün olup olmadığını söyler.

Tam Sayılar

Toplama, çıkarma, çarpma, bölme gibi işlemler.

```
7 / 2 → 3
```

Dizgiler

Karşılaştırma, birleştirme, alt dizgi alma gibi işlemler.

```
"veri" + "yapısı"
```

Geçersiz işlem

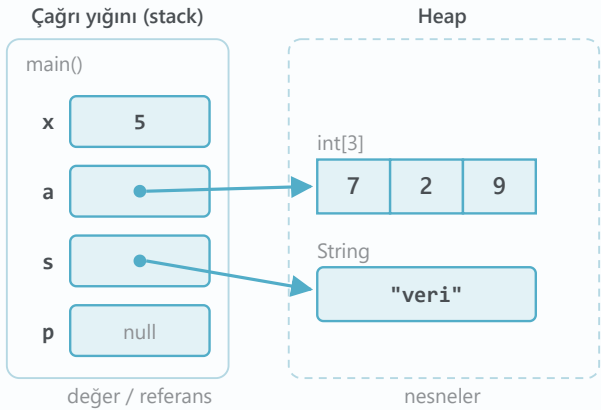
Derleyici hatayı çalıştırmadan yakalar.

```
"abc" * 2 → derleme hatası
```

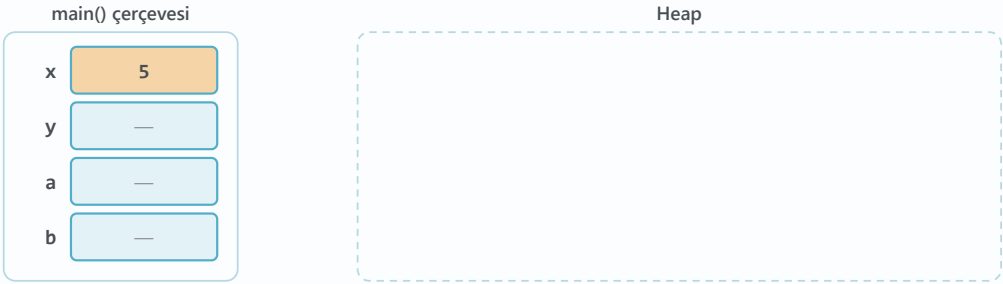
Java'da Bellek: Yığın ve Heap

- **İlkel** değişken değeri doğrudan tutar.
- **Referans** değişken, heap'teki nesnenin **adresini** tutar; `new` nesneyi heap'te oluşturur.
- Her metot çağrısı yığında bir **çerçeve** açar; yerel değişkenler oradadır, dönüşte silinir.
- Erişilemeyen nesneleri **çöp toplayıcı** temizler; `p[0]` → `NullPointerException`.

```
int x = 5;              // ilkel: değer
int[] a = {7, 2, 9};    // referans → dizi
String s = "veri";      // referans → nesne
int[] p = null;         // hiçbir şeyi göstermez
```



Kod İzleme: Değer mi, Referans mı?



x = 5: ilkel değişken, değer doğrudan çerçevededir.

Adım 1 / 8

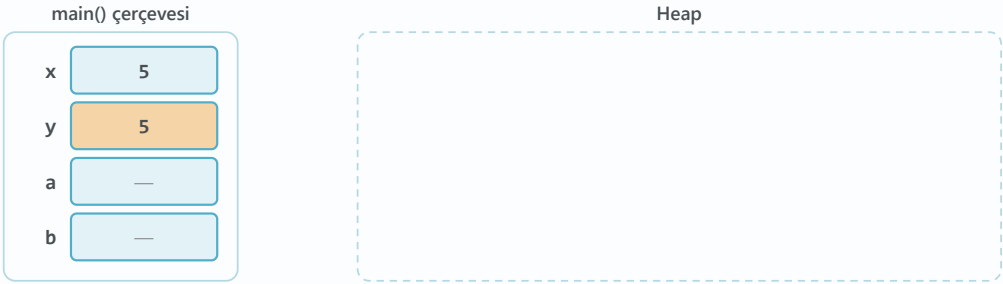
```
▶ int x = 5;
2 int y = x;
3 y = 7;
4 int[] a = {1, 2, 3};
5 int[] b = a;
6 b[0] = 9;
7 b = new int[2];
8 System.out.println(x + " " + a[0]);
```

DEĞİŞKENLER

x = 5

ÇIKTI

Kod İzleme: Değer mi, Referans mı?



y = x: değerin **kopyası** alınır; x ile y birbirinden bağımsızdır.

Adım 2 / 8

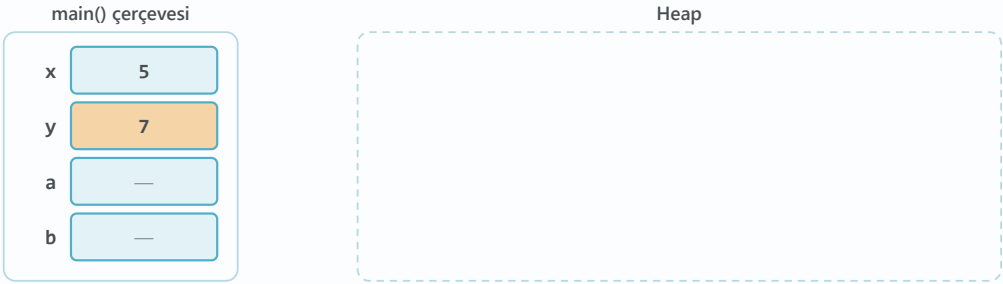
```
1 int x = 5;
▶ int y = x;
3 y = 7;
4 int[] a = {1, 2, 3};
5 int[] b = a;
6 b[0] = 9;
7 b = new int[2];
8 System.out.println(x + " " + a[0]);
```

DEĞİŞKENLER

x = 5 y = 5

ÇIKTI

Kod İzleme: Değer mi, Referans mı?



y = 7: yalnız y değişti; x hâlâ 5.

Adım 3 / 8

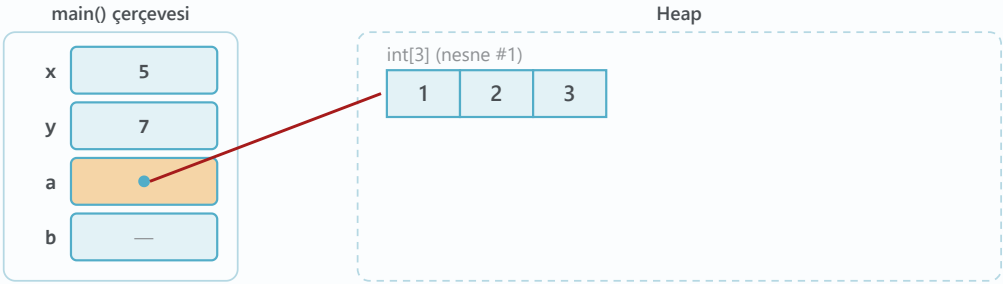
```
1 int x = 5;
2 int y = x;
▶ y = 7;
4 int[] a = {1, 2, 3};
5 int[] b = a;
6 b[0] = 9;
7 b = new int[2];
8 System.out.println(x + " " + a[0]);
```

DEĞİŞKENLER

x = 5 y = 7

ÇIKTI

Kod İzleme: Değer mi, Referans mı?



Dizi **heap'te** oluşturulur; **a** yalnızca ona bir referans (ok) tutar.

Adım 4 / 8

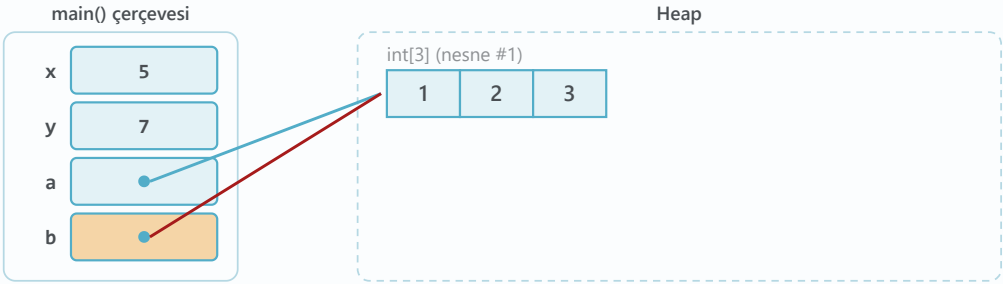
```
1 int x = 5;
2 int y = x;
3 y = 7;
▶ int[] a = {1, 2, 3};
5 int[] b = a;
6 b[0] = 9;
7 b = new int[2];
8 System.out.println(x + " " + a[0]);
```

DEĞİŞKENLER

`x = 5` `y = 7` `a = →#1 [1, 2, 3]`

ÇIKTI

Kod İzleme: Değer mi, Referans mı?



b = a: dizi kopyalanmaz, **referans** kopyalanır. İki ok aynı nesneyi gösterir.

Adım 5 / 8

```
1 int x = 5;
2 int y = x;
3 y = 7;
4 int[] a = {1, 2, 3};
▶ int[] b = a;
6 b[0] = 9;
7 b = new int[2];
8 System.out.println(x + " " + a[0]);
```

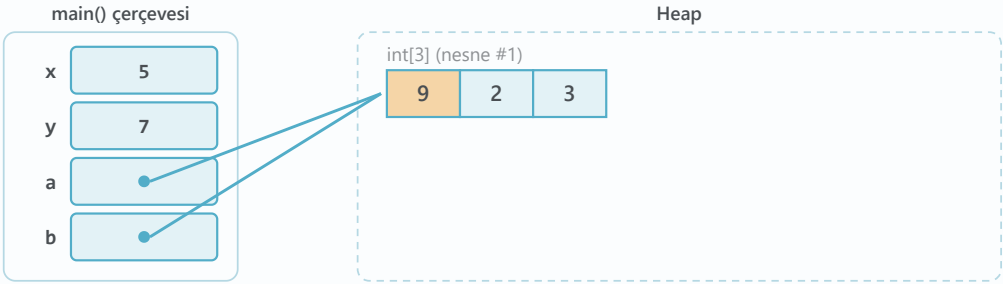
DEĞİŞKENLER

`x = 5` `y = 7` `a = →#1 [1, 2, 3]`

`b = →#1 [1, 2, 3]`

ÇIKTI

Kod İzleme: Değer mi, Referans mı?



b[0] = 9: nesnenin kendisi değişti → **a[0] de 9** görünür (takma ad, aliasing).

Adım 6 / 8

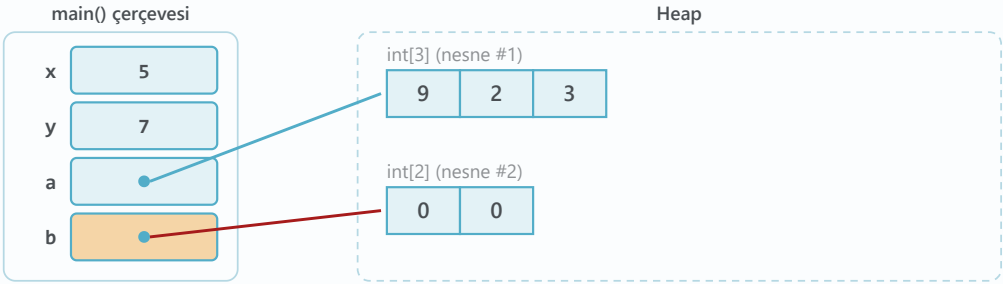
```
1 int x = 5;
2 int y = x;
3 y = 7;
4 int[] a = {1, 2, 3};
5 int[] b = a;
▶ b[0] = 9;
7 b = new int[2];
8 System.out.println(x + " " + a[0]);
```

DEĞİŞKENLER

`x = 5` `y = 7` `a = →#1 [9, 2, 3]` `b = →#1 [9, 2, 3]`

ÇIKTI

Kod İzleme: Değer mi, Referans mı?



b = new int[2]: b artık yeni nesneyi gösterir; a etkilenmez. Yeni dizi 0 ile dolar.

Adım 7 / 8

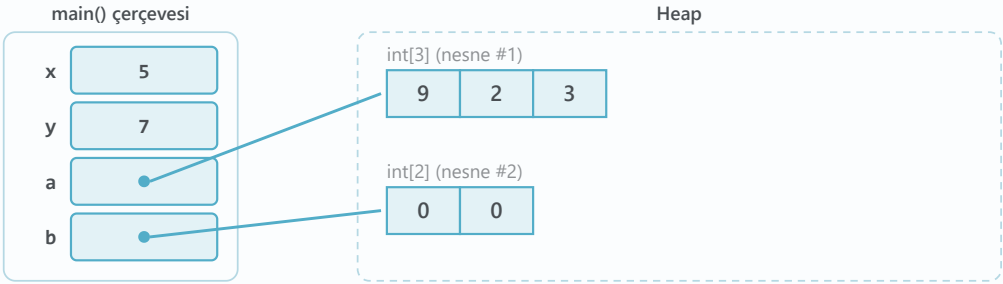
```
1 int x = 5;
2 int y = x;
3 y = 7;
4 int[] a = {1, 2, 3};
5 int[] b = a;
6 b[0] = 9;
▶ b = new int[2];
8 System.out.println(x + " " + a[0]);
```

DEĞİŞKENLER

x = 5 | y = 7 | a = →#1 [9, 2, 3] | b = →#2 [0, 0]

ÇIKTI

Kod İzleme: Değer mi, Referans mı?



Çıktı **5 9**. Kural: **=** ilkel tipte değeri, referans tipte adresi kopyalar.

Adım 8 / 8

```
1 int x = 5;
2 int y = x;
3 y = 7;
4 int[] a = {1, 2, 3};
5 int[] b = a;
6 b[0] = 9;
7 b = new int[2];
▶ System.out.println(x + " " + a[0]);
```

DEĞİŞKENLER

`x = 5` `y = 7` `a = →#1 [9, 2, 3]` `b = →#2 [0, 0]`

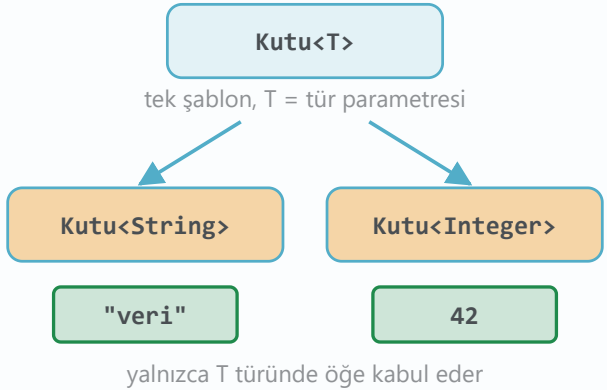
ÇIKTI

```
5 9
```

Jenerikler: Her Tür İçin Tek Yapı

```java
public class Kutu<T> {
  private T icerik;
  public void koy(T x) { icerik = x; }
  public T al() { return icerik; }
}

Kutu<String> k = new Kutu<>();
k.koy("veri");
String s = k.al();       // dönüşüm gerekmez
Kutu<Integer> m = new Kutu<>();
m.koy(42);               // int → Integer
// m.koy("x");           // DERLEME HATASI
```



- Tür denetimi **derleme zamanında**; yanlış tür çalışmadan yakalanır.
- T yalnız referans tipi olabilir: `Kutu<int>` yok, `Integer` kullanılır.
- Java koleksiyonları jeneriktir: `ArrayList<E>`, `Map<K,V>`.

İlkel Olmayan (Non-Primitive) Veri Yapıları

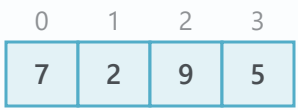
Daha karmaşık veri yapılarıdır. Örneğin:

Sıralı Diziler, listeler	**Bağlantılı** Bağlı listeler	**Hiyerarşik** Ağaçlar
Ağ (Network) Çizgeler	**İndislenmiş** Hash tabloları	**...** Kümeler, eşlemeler

- Karmaşıktır ve birden fazla ilkel veri tipini içerebilir.
- Çoğu dinamik boyutludur (diziler hariç); saklanan veri miktarı değişebilir.
- Özelleştirilmiş işlemleri gerçekleştirmek için kullanılırlar.

Farklı Veri Yapıları: Bir Bakışta

Aynı dört değer (7, 2, 9, 5), sekiz farklı düzende. Her birinin güçlü ve zayıf yönü farklıdır:



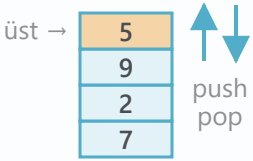
a[2] → 9 (tek adım)

Dizi
ardışık hücreler, indisle O(1) erişim, sabit boyut

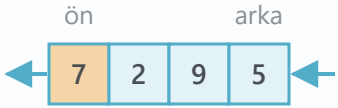


i. öğeye gitmek: O(n)

Bağlı liste
referanslarla bağlı düğümler, esnek boyut

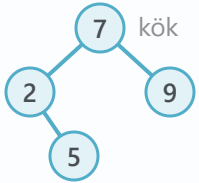


Yığın
LIFO: yalnız üstten ekle/çıkar

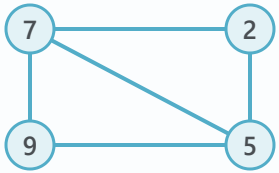


önden çık · arkadan gir

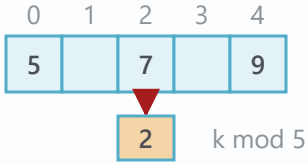
Kuyruk
FIFO: ilk giren ilk çıkar



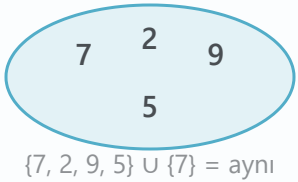
Ağaç
hiyerarşi: kök, çocuk, yaprak



Çizge
düğüm + kenar, birçoğa-birçok ilişki



Hash tablosu
h(k) ile kovaya; ortalama O(1)



{7, 2, 9, 5} ∪ {7} = aynı

Küme / Eşleme
tekil öğe / anahtar → değer

Soyut Veri Tipleri ve Veri Yapıları

Soyut Veri Tipi (ADT)

- Listeler, yığınlar, kuyruklar, kümeler, eşlemeler, çizgeler…
- Hangi işlemlerin yapılabileceğini (davranışı) belirler.
- Gerçekleme ayrıntılarına odaklanmaz.

Veri Yapısı (Data Structure)

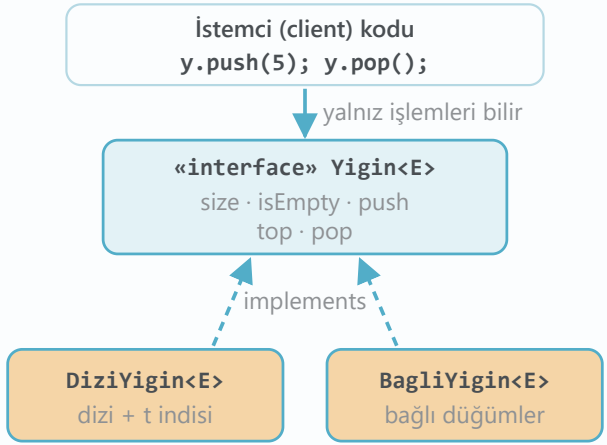
- Bağlı listeler, diziler, komşuluk matrisleri…
- Soyut veri tiplerini uygulayan somut yapılardır.
- Bellek ve performans detaylarına odaklanır.



ADT'den Java'ya: Arayüz (`interface`)

```
public interface Yigin<E> {
  int size();          // öğe sayısı
  boolean isEmpty();   // boş mu?
  void push(E e);      // üste ekle
  E top();             // üsttekini döndür
  E pop();             // üsttekini çıkar
}
```

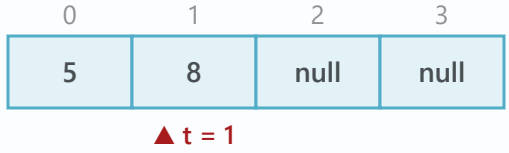
- **interface** = ADT'nin sözleşmesi: *ne* yapılır.
- **class** = veri yapısı: *nasıl* yapılır.
- Aynı ADT'nin birden çok gerçeklemesi olabilir.



`Yigin<Integer> y = new DiziYigin<>(10);` — gerçeklemeyi değiştirmek için yalnız bu satır değişir; istemci kodu aynı kalır.

Bir Gerçekleme: `DiziYigin`

```java
public class DiziYigin<E> implements Yigin<E> {
  private E[] veri;          // öğeler
  private int t = -1;        // üst öğenin indisi
  public DiziYigin(int kapasite) {
    veri = (E[]) new Object[kapasite];
  }
  public int size() { return t + 1; }
  public boolean isEmpty() { return t == -1; }
  public E top() {
    if (isEmpty()) return null;
    return veri[t];
  }
  // push ve pop
}
```



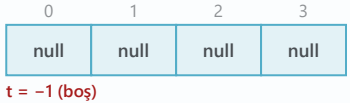
`size() = t + 1 = 2`

İşlem	Süre
size, isEmpty, top	O(1)
push, pop	O(1)
Bellek	O(N) (kapasite)

- Java'da jenerik dizi doğrudan oluşturulamaz: `(E[]) new Object[..]`.
- Kapasite sabit: dolunca `push` hata fırlatır.

Kod İzleme: `push` / `pop`

```
1  public void push(E e) {
2    if (size() == veri.length)
3      throw new IllegalStateException("dolu");
4    veri[++t] = e;
5  }
6  public E pop() {
7    if (isEmpty()) return null;
8    E e = veri[t];
9    veri[t--] = null;      // GC için boşalt
10   return e;
11 }
▶ y.push(5); y.push(8);
13 System.out.println(y.pop());
14 y.push(3);
```



y, kapasitesi 4 olan **boş** bir DiziYigin (t = −1). `push(5)` çağrılır.

Adım 1 / 16

DEĞİŞKENLER

`t = -1` `veri = [null, null, null, null]`

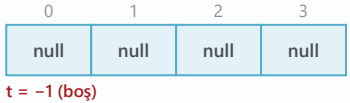
ÇAĞRI YIĞINI

`main()`

ÇIKTI

Kod İzleme: `push` / `pop`

```
1  public void push(E e) {
2    if (size() == veri.length)
3      throw new IllegalStateException("dolu");
4    veri[++t] = e;
5  }
6  public E pop() {
7    if (isEmpty()) return null;
8    E e = veri[t];
9    veri[t--] = null;      // GC için boşalt
10   return e;
11 }
12 y.push(5); y.push(8);
13 System.out.println(y.pop());
14 y.push(3);
```



size() = 0 ≠ 4 → dolu değil.

Adım 2 / 16

DEĞİŞKENLER

`t = -1` `veri = [null, null, null, null]` `e = 5`

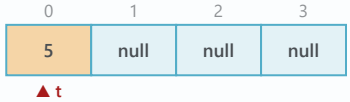
ÇAĞRI YIĞINI

- `push(5)`
- `main()`

ÇIKTI

Kod İzleme: `push` / `pop`

```
1  public void push(E e) {
2    if (size() == veri.length)
3      throw new IllegalStateException("dolu");
▶    veri[++t] = e;
5  }
6  public E pop() {
7    if (isEmpty()) return null;
8    E e = veri[t];
9    veri[t--] = null;      // GC için boşalt
10   return e;
11 }
12 y.push(5); y.push(8);
13 System.out.println(y.pop());
14 y.push(3);
```



++t → t = 0, sonra **veri[0] = 5**. Önce artır, sonra yaz.

Adım 3 / 16

DEĞİŞKENLER

`t = 0` `veri = [5, null, null, null]` `e = 5`

ÇAĞRI YIĞINI

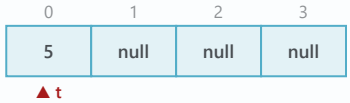
`push(5)`

`main()`

ÇIKTI

Kod İzleme: `push` / `pop`

```
1  public void push(E e) {
2    if (size() == veri.length)
3      throw new IllegalStateException("dolu");
4    veri[++t] = e;
5  }
6  public E pop() {
7    if (isEmpty()) return null;
8    E e = veri[t];
9    veri[t--] = null;      // GC için boşalt
10   return e;
11 }
▶ y.push(5); y.push(8);
13 System.out.println(y.pop());
14 y.push(3);
```



`push(8)` çağrılır.

Adım 4 / 16

DEĞİŞKENLER

`t = 0` `veri = [5, null, null, null]`

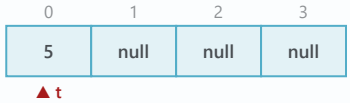
ÇAĞRI YIĞINI

`main()`

ÇIKTI

Kod İzleme: `push` / `pop`

```
1  public void push(E e) {
▶    if (size() == veri.length)
3      throw new IllegalStateException("dolu");
4    veri[++t] = e;
5  }
6  public E pop() {
7    if (isEmpty()) return null;
8    E e = veri[t];
9    veri[t--] = null;      // GC için boşalt
10   return e;
11 }
12 y.push(5); y.push(8);
13 System.out.println(y.pop());
14 y.push(3);
```



size() = 1 ≠ 4 → dolu değil.

Adım 5 / 16

DEĞİŞKENLER

`t = 0` `veri = [5, null, null, null]` `e = 8`

ÇAĞRI YIĞINI

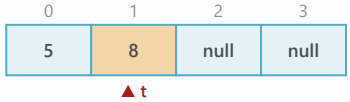
`push(8)`

`main()`

ÇIKTI

Kod İzleme: `push` / `pop`

```
1  public void push(E e) {
2    if (size() == veri.length)
3      throw new IllegalStateException("dolu");
▶    veri[++t] = e;
5  }
6  public E pop() {
7    if (isEmpty()) return null;
8    E e = veri[t];
9    veri[t--] = null;      // GC için boşalt
10   return e;
11 }
12 y.push(5); y.push(8);
13 System.out.println(y.pop());
14 y.push(3);
```



++t → t = 1, sonra **veri[1] = 8**. Önce artır, sonra yaz.

Adım 6 / 16

DEĞİŞKENLER

`t = 1` `veri = [5, 8, null, null]` `e = 8`

ÇAĞRI YIĞINI

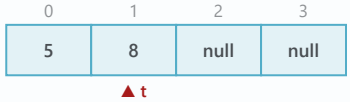
`push(8)`

`main()`

ÇIKTI

Kod İzleme: `push` / `pop`

```
1  public void push(E e) {
2    if (size() == veri.length)
3      throw new IllegalStateException("dolu");
4    veri[++t] = e;
5  }
6  public E pop() {
7    if (isEmpty()) return null;
8    E e = veri[t];
9    veri[t--] = null;      // GC için boşalt
10   return e;
11 }
12 y.push(5); y.push(8);
▶ System.out.println(y.pop());
14 y.push(3);
```



`pop()` çağrılır; dönen değer yazdırılacak.

Adım 7 / 16

DEĞİŞKENLER

`t = 1` `veri = [5, 8, null, null]`

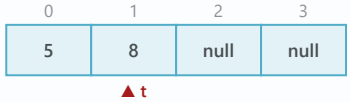
ÇAĞRI YIĞINI

`main()`

ÇIKTI

Kod İzleme: `push` / `pop`

```
1  public void push(E e) {
2    if (size() == veri.length)
3      throw new IllegalStateException("dolu");
4    veri[++t] = e;
5  }
6  public E pop() {
▶    if (isEmpty()) return null;
8    E e = veri[t];
9    veri[t--] = null;      // GC için boşalt
10   return e;
11 }
12 y.push(5); y.push(8);
13 System.out.println(y.pop());
14 y.push(3);
```



t = 1 → boş değil, devam.

Adım 8 / 16

DEĞİŞKENLER

`t = 1` `veri = [5, 8, null, null]`

ÇAĞRI YIĞINI

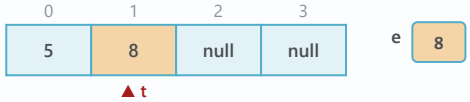
`pop()`

`main()`

ÇIKTI

Kod İzleme: `push` / `pop`

```
1  public void push(E e) {
2    if (size() == veri.length)
3      throw new IllegalStateException("dolu");
4    veri[++t] = e;
5  }
6  public E pop() {
7    if (isEmpty()) return null;
▶    E e = veri[t];
9    veri[t--] = null;      // GC için boşalt
10   return e;
11 }
12 y.push(5); y.push(8);
13 System.out.println(y.pop());
14 y.push(3);
```



e = veri[1] = 8: en son eklenen öğe (LIFO).

Adım 9 / 16

DEĞİŞKENLER

`t = 1` `veri = [5, 8, null, null]` `e = 8`

ÇAĞRI YIĞINI

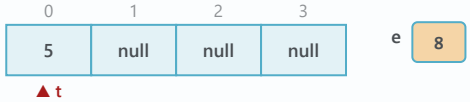
`pop()`

`main()`

ÇIKTI

Kod İzleme: `push` / `pop`

```
1  public void push(E e) {
2    if (size() == veri.length)
3      throw new IllegalStateException("dolu");
4    veri[++t] = e;
5  }
6  public E pop() {
7    if (isEmpty()) return null;
8    E e = veri[t];
▶    veri[t--] = null;      // GC için boşalt
10   return e;
11 }
12 y.push(5); y.push(8);
13 System.out.println(y.pop());
14 y.push(3);
```



Hücre **null** yapılır (nesne serbest kalsın), **t--** → t = 0.

Adım 10 / 16

DEĞİŞKENLER

`t = 0` `veri = [5, null, null, null]` `e = 8`

ÇAĞRI YIĞINI

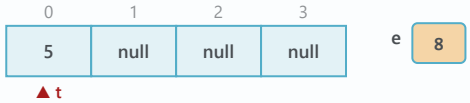
`pop()`

`main()`

ÇIKTI

Kod İzleme: `push` / `pop`

```
1  public void push(E e) {
2    if (size() == veri.length)
3      throw new IllegalStateException("dolu");
4    veri[++t] = e;
5  }
6  public E pop() {
7    if (isEmpty()) return null;
8    E e = veri[t];
9    veri[t--] = null;      // GC için boşalt
▶    return e;
11 }
12 y.push(5); y.push(8);
13 System.out.println(y.pop());
14 y.push(3);
```



return 8: çerçeve kapanır.

Adım 11 / 16

DEĞİŞKENLER

`t = 0` `veri = [5, null, null, null]` `e = 8`

ÇAĞRI YIĞINI

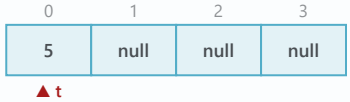
`pop()`

`main()`

ÇIKTI

Kod İzleme: `push` / `pop`

```
1  public void push(E e) {
2    if (size() == veri.length)
3      throw new IllegalStateException("dolu");
4    veri[++t] = e;
5  }
6  public E pop() {
7    if (isEmpty()) return null;
8    E e = veri[t];
9    veri[t--] = null;      // GC için boşalt
10   return e;
11 }
12 y.push(5); y.push(8);
▶ System.out.println(y.pop());
14 y.push(3);
```



Ekrana **8** yazılır.

Adım 12 / 16

DEĞİŞKENLER

`t = 0` `veri = [5, null, null, null]`

ÇAĞRI YIĞINI

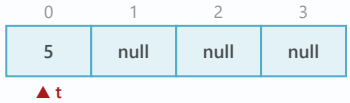
`main()`

ÇIKTI

```
8
```

Kod İzleme: `push` / `pop`

```
1  public void push(E e) {
2    if (size() == veri.length)
3      throw new IllegalStateException("dolu");
4    veri[++t] = e;
5  }
6  public E pop() {
7    if (isEmpty()) return null;
8    E e = veri[t];
9    veri[t--] = null;      // GC için boşalt
10   return e;
11 }
12 y.push(5); y.push(8);
13 System.out.println(y.pop());
▶ y.push(3);
```



`push(3)` : boşalan 1 numaralı hücre yeniden kullanılacak.

Adım 13 / 16

DEĞİŞKENLER

`t = 0` `veri = [5, null, null, null]`

ÇAĞRI YIĞINI

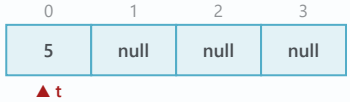
`main()`

ÇIKTI

```
8
```

Kod İzleme: `push` / `pop`

```
1  public void push(E e) {
▶    if (size() == veri.length)
3      throw new IllegalStateException("dolu");
4    veri[++t] = e;
5  }
6  public E pop() {
7    if (isEmpty()) return null;
8    E e = veri[t];
9    veri[t--] = null;      // GC için boşalt
10   return e;
11 }
12 y.push(5); y.push(8);
13 System.out.println(y.pop());
14 y.push(3);
```



size() = 1 ≠ 4 → dolu değil.

Adım 14 / 16

DEĞİŞKENLER

`t = 0` `veri = [5, null, null, null]` `e = 3`

ÇAĞRI YIĞINI

`push(3)`

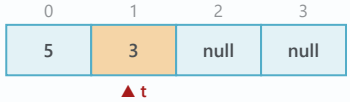
`main()`

ÇIKTI

```
8
```

Kod İzleme: `push` / `pop`

```
1  public void push(E e) {
2    if (size() == veri.length)
3      throw new IllegalStateException("dolu");
▶    veri[++t] = e;
5  }
6  public E pop() {
7    if (isEmpty()) return null;
8    E e = veri[t];
9    veri[t--] = null;      // GC için boşalt
10   return e;
11 }
12 y.push(5); y.push(8);
13 System.out.println(y.pop());
14 y.push(3);
```



++t → t = 1, sonra **veri[1] = 3**. Önce artır, sonra yaz.

Adım 15 / 16

DEĞİŞKENLER

`t = 1` `veri = [5, 3, null, null]` `e = 3`

ÇAĞRI YIĞINI

`push(3)`

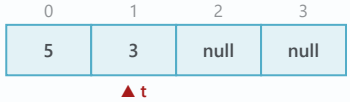
`main()`

ÇIKTI

```
8
```

Kod İzleme: `push` / `pop`

```
1  public void push(E e) {
2    if (size() == veri.length)
3      throw new IllegalStateException("dolu");
4    veri[++t] = e;
5  }
6  public E pop() {
7    if (isEmpty()) return null;
8    E e = veri[t];
9    veri[t--] = null;      // GC için boşalt
10   return e;
11 }
12 y.push(5); y.push(8);
13 System.out.println(y.pop());
▶ y.push(3);
```



Son durum: [5, 3], t = 1. Her işlem n'den bağımsız, sabit sayıda adım → **O(1)**.

Adım 16 / 16

DEĞİŞKENLER

`t = 1` `veri = [5, 3, null, null]`

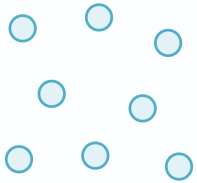
ÇAĞRI YIĞINI

`main()`

ÇIKTI

```
8
```

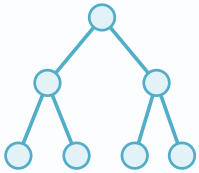
Verilerin Düzenlenmesi



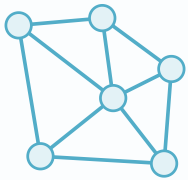
Konumsal olmayan



Sıralı



Ağaç



Çizge

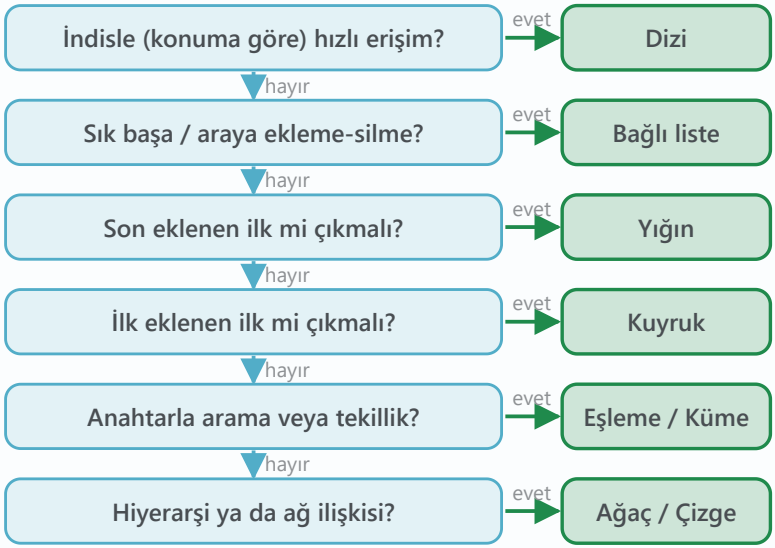
Düzen	İlişki	Günlük hayattan örnek
Konumsal olmayan	ilişkisiz	sınıftaki öğrenciler
Sıralı	bire-bir	sıra bekleyen öğrenciler
Ağaç	bire-birçok	aile ağacı
Çizge	birçoğa-birçok	yol haritası

Veri Yapısı Seçimi

Aşağıdaki sorular kritik rol oynar:

- Veri yapısı başlangıçta tamamen doldurulabilir mi?
- Veri yapısından öğeler silinebilir mi?
- Ekleme sırasının önemi var mı?
- Veri yapısındaki öğeler sıralanabilir mi?
- Öğeler belirli bir sırayla mı işlenir, yoksa rastgele erişim de gerekli mi?

Durum değişmeyecekse **statik** (static) yapılar; sık ekleme / silme varsa **dinamik** yapılar.



Temel Veri Yapıları: Genel Bakış

Yapı	Temel fikir	Tipik kullanım
Dizi	ardışık hücreler; indisle O(1) erişim, sabit boyut	tablo ve matris, diğer yapıların temeli
Bağlı liste	referanslarla bağlı düğümler; esnek boyut	sık ekleme ve silme
Yığın	LIFO: yalnız üstten ekle / çıkar	geri al (undo), çağrı yığını, parantez kontrolü
Kuyruk	FIFO: arkadan ekle, önden çıkar	yazdırma kuyruğu, görev sırası, BFS
Ağaç	hiyerarşi: kök, çocuk, yaprak	dosya sistemi, HTML belgesi (DOM), indisler
Öncelikli kuyruk	en öncelikli öğe hep önde (heap)	görev çizelgeleme, acil servis triyajı
Eşleme / hash tablosu	anahtar → değer; h(k) ile kovaya, ortalama O(1)	öğrenci no → kayıt, önbellek, kelime sayımı
Küme	tekil öğeler; hızlı üyelik testi	tekrarları ayıklama, birleşim ve kesişim
Çizge	düğüm + kenar; birçoğa-birçok ilişki	yol haritası, sosyal ağ, en kısa yol

ÖZET

Özet

Konu	Hatırlanacaklar
Dijital veri	metin, sayı, görüntü ve ses 0 ve 1'lerle, farklı kodlamalarla temsil edilir
Veri tipi	değer kümesi + işlemler; tip güvenliği hataları derleme zamanında yakalar
İlkel / referans	ilkel değişken değeri tutar; referans heap'teki nesneyi gösterir, atama yalnız referansı kopyalar
Jenerikler	`Kutu<T>`: her referans tipi için tek sınıf; `int` yerine `Integer`
ADT / veri yapısı	ADT ne yapılacağını (arayüz), veri yapısı nasıl yapılacağını (sınıf) belirler
Yapı seçimi	erişim biçimi, ekleme / silme sıklığı, sıra ihtiyacı ve ilişki türü belirler

ÖZET

Örnek Problemler

1. Metin editöründe "geri al" için hangi veri yapısı uygundur? Neden?
2. Yazıcıya gelen belgeleri geliş sırasıyla basmak için hangi yapı uygundur?
3. Öğrenci numarasıyla öğrenci bilgisine hızlı erişim için hangi yapı uygundur?
4. İki nokta arasındaki en kısa yolu bulmak için şehir haritasını nasıl modellersiniz?
5. Listedeki tekrar eden öğeleri ayıklamak için hangi yapıyı kullanırsınız?
6. Yığın ADT'si Java'da arayüz olarak mı, sınıf olarak mı yazılır? `DiziYigin` hangisidir?

▶ **Cevapları göster**

ÖZET

Alıştırmalar

1. `int[] a = {1, 2}; int[] b = a; b[1] = 7;` sonrası `a[1]` kaçtır? Neden?
2. `Kutu<int> k;` neden derlenmez? Yerine ne yazılır?
3. `'B'` karakteri ve 13 sayısı 8 bitle nasıl temsil edilir?
4. `String s = "veri"; String t = s; t = t + "!";` sonrası `s` nedir? Neden?
5. Kapasitesi 3 olan boş `DiziYigin` 'e push(1), push(2), pop(), push(3), push(4) uygulanıyor. Son durumda t ve dizi nedir?
6. Müzik çalar: sıralı çalma listesi, "son çalınanlar" ve şarkı adına göre hızlı arama için hangi yapılar uygundur?

▶ **Cevaplar / ipuçları**

Kaynaklar

- Goodrich, Tamassia ve Goldwasser, *Data Structures and Algorithms in Java*, 6. baskı, 2014.
- Wirth, *Algorithms + Data Structures = Programs*, 1976.
- Oracle, Java SE 17 API belgeleri.

Son

Bölüm 1: Giriş

Sonraki sunum: Bölüm 2: Algoritma Analizi · Çalışma soruları · Terim sözlüğü