

Bölüm 7: Giriş Çıkış

Java ile Nesne Yönelimli Programlama

Akışlar

Metin ve ikili dosyalar

Karakter kodlama

File ve Scanner

Reader / Writer

Tamponlama

Dosya kopyalama

Rastgele erişim

Serileştirme

NIO: tampon, kanal, seçici

Sercan KÜLCÜ · Giresun Üniversitesi · Elektrik-Elektronik Mühendisliği

GİRİŞ

Öğrenme Çıktıları

Akışı açıklamak

Girdi/çıkış akışını ve `System.in` , `System.out` , `System.err` akışlarını anlatmak; klavyeden `Scanner` ile okumak.

Metin ile ikiliyi ayırmak

Metin ve ikili dosya farkını, ASCII, ISO-8859-9 ve UTF-8 kodlamalarını, bir sayının dosyada kaç bayt tuttuğunu hesaplamak.

Dosya okumak ve yazmak

`File` , `Scanner` , `BufferedReader` , `PrintWriter` , `FileWriter` ile metin dosyası işlemek; kaynakları `try` -with-resources ile kapatmak.

Doğru akışı seçmek

Bayt (`InputStream` / `OutputStream`) ve karakter (`Reader` / `Writer`) akışlarını, tamponlu ve sarmalayıcı akışları ayırt etmek.

Kopyalamak, konumlanmak

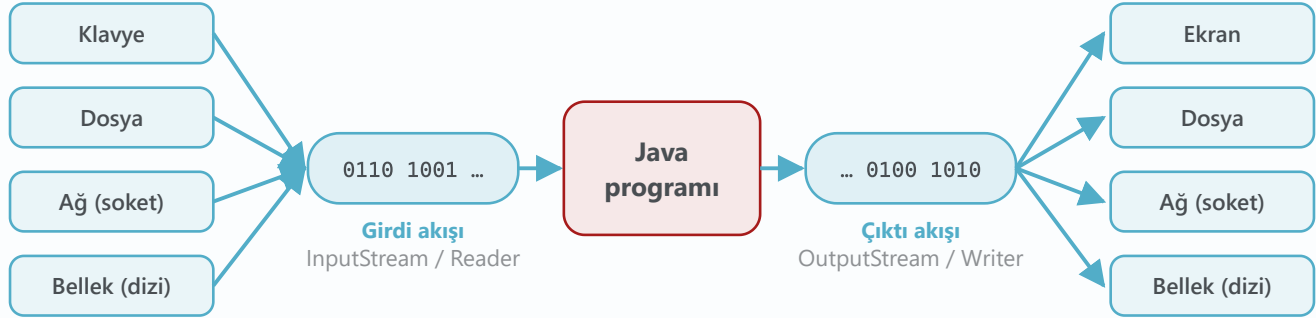
Bayt, blok ve satır düzeyinde kopyalamak; `RandomAccessFile` ile istenen konuma atlamak.

Nesne saklamak ve NIO

Nesneleri serileştirmek (`Serializable` , `transient`); NIO tamponu, kanalı ve seçicisinin ne işe yaradığını açıklamak.

Akış (Stream)

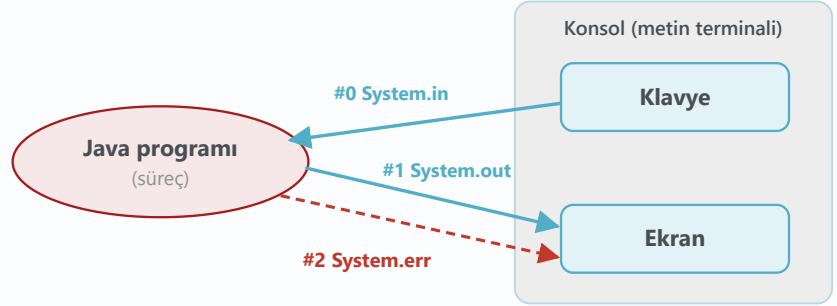
- **Akış** (stream), veriyi bir kaynaktan bir hedefe **sırayla**, parça parça taşıyan yapıdır. Veri karakter, sayı veya ham **bayt** olabilir.
- **Girdi akışı** (input stream): verinin programa **girdiği** akış. Örnek: `System.in` (klavye).
- **Çıktı akışı** (output stream): verinin programdan **çıktığı** akış. Örnek: `System.out` (ekran).
- Benzetme: su borusu. Bir uçtan giren su öbür uçtan **aynı sırayla** çıkar; borunun öbür ucunda musluk mu depo mu olduğu sizi ilgilendirmez.



Kaynak ister klavye, ister dosya, ister ağ olsun, program aynı `read()` / `write()` metotlarını kullanır. Java G/Ç (giriş/çıkış, I/O) kütüphanesinin gücü bu ortak arayüzdedir.

Klavye ve Ekran: Standart Akışlar

- **Geçici veri işleme:** klavye ve ekran verisi saklanmaz; program kapanınca kaybolur.
- **Anında akış:** kullanıcı yazdıkça girdi gelir, program yazdıkça ekranda görünür.
- Her Java programı açılırken üç hazır akış alır:



`System.err` da ekrana yazar ama ayrı bir kanaldır: çıktı bir dosyaya yönlendirilse bile (`java Uygulama > sonuc.txt`) hata mesajları ekranda kalır.

Akış	Java nesnesi	Tipi
standart girdi	<code>System.in</code>	<code>InputStream</code>
standart çıktı	<code>System.out</code>	<code>PrintStream</code>
standart hata	<code>System.err</code>	<code>PrintStream</code>

İlk Örnek: Okuma ve Yazma

```
import java.util.Scanner;

public class AkisOrnek {
    public static void main(String[] args) {
        // Giriş akışından veri okuma
        Scanner scanner = new Scanner(System.in);
        int kullanıcıGirdisi = scanner.nextInt();
        // Çıkış akışına veri yazma
        System.out.println("Merhaba, Dünya!");
        System.out.println("Girdiğiniz sayı: " + kullanıcıGirdisi);
        System.err.println("Bu satır hata akışına yazıldı.");
    }
}
```

KONSOL (GİRDİ: 42)

```
42
Merhaba, Dünya!
Girdiğiniz sayı: 42
Bu satır hata akışına yazıldı.
```

- `new Scanner(System.in)` : klavye akışını okuyan bir **tarayıcı** oluşturur. `Scanner`, `java.util` paketindedir; `import` şarttır.
- `nextInt()` kullanıcı bir sayı yazıp Enter'a basana kadar **bekler** (bloklanır).
- `System.out.println` çıktı akışına bir satır yazar.
- `System.err.println` hata akışına yazar; çoğu IDE bu satırı **kırmızı** gösterir.
- Kullanıcı sayı yerine `abc` yazarsa `InputMismatchException` fırlatılır.

Scanner ile Klavyeden Okuma: Adım Adım

```
1 import java.util.Scanner;
2
3 public class Konso10kuma {
4     public static void main(String[] args) {
5         ▶ Scanner klavye = new Scanner(System.in);
6         System.out.print("Adınız: ");
7         String ad = klavye.nextLine();
8         System.out.print("Yaşınız: ");
9         int yas = klavye.nextInt();
10        int seneye = yas + 1;
11        System.out.println("Merhaba " + ad + "!");
12        System.out.println("Seneye " + seneye + " yaşındasın.");
13        klavye.close();
14    }
15 }
```

System.in tamponu

▲
sıradaki

Scanner nesnesi System.in akışına
bağlanır. Klavye tamponu boş.

Adım 1 / 9

ÇIKTI

DEĞİŞKENLER

klavye = Scanner(System.in)

Scanner ile Klavyeden Okuma: Adım Adım

```
1 import java.util.Scanner;
2
3 public class Konso10kuma {
4     public static void main(String[] args) {
5         Scanner klavye = new Scanner(System.in);
6         ▶ System.out.print("Adınız: ");
7         String ad = klavye.nextLine();
8         System.out.print("Yaşınız: ");
9         int yas = klavye.nextInt();
10        int seneye = yas + 1;
11        System.out.println("Merhaba " + ad + "!");
12        System.out.println("Seneye " + seneye + " yaşındasın.");
13        klavye.close();
14    }
15 }
```

System.in tamponu



sıradaki

print (println değil) soruyu yazar;
kullanıcı cevabı **aynı satıra** yazar.

Adım 2 / 9

ÇIKTI

Adınız:

DEĞİŞKENLER

klavye = Scanner(System.in)

Scanner ile Klavyeden Okuma: Adım Adım

```
1 import java.util.Scanner;
2
3 public class Konso10kuma {
4     public static void main(String[] args) {
5         Scanner klavye = new Scanner(System.in);
6         System.out.print("Adınız: ");
7         ▶ String ad = klavye.nextLine();
8         System.out.print("Yaşınız: ");
9         int yas = klavye.nextInt();
10        int seneye = yas + 1;
11        System.out.println("Merhaba " + ad + "!");
12        System.out.println("Seneye " + seneye + " yaşındasın.");
13        klavye.close();
14    }
15 }
```

System.in tamponu

A y ş e Y ı l m a z ↵

▲
sıradaki

Kullanıcı `Ayşe Yılmaz` yazıp Enter'a basar. `nextLine()` boşluk dâhil **bütün satırı** alır, satır sonunu (↵) atar.

Adım 3 / 9

ÇIKTI

Adınız: Ayşe Yılmaz

DEĞİŞKENLER

ad = "Ayşe Yılmaz"

Scanner ile Klavyeden Okuma: Adım Adım

```
1 import java.util.Scanner;
2
3 public class Konso10kuma {
4     public static void main(String[] args) {
5         Scanner klavye = new Scanner(System.in);
6         System.out.print("Adınız: ");
7         String ad = klavye.nextLine();
8         ▶ System.out.print("Yaşınız: ");
9         int yas = klavye.nextInt();
10        int seneye = yas + 1;
11        System.out.println("Merhaba " + ad + "!");
12        System.out.println("Seneye " + seneye + " yaşındasın.");
13        klavye.close();
14    }
15 }
```

System.in tamponu

A y ş e Y ı l m a z ↵

▲
sıradaki

İkinci soru yazılır.

Adım 4 / 9

ÇIKTI

Adınız: Ayşe Yılmaz
Yaşınız:

DEĞİŞKENLER

ad = "Ayşe Yılmaz"

Scanner ile Klavyeden Okuma: Adım Adım

```
1 import java.util.Scanner;
2
3 public class Konso1Okuma {
4     public static void main(String[] args) {
5         Scanner klavye = new Scanner(System.in);
6         System.out.print("Adınız: ");
7         String ad = klavye.nextLine();
8         System.out.print("Yaşınız: ");
9         int yas = klavye.nextInt();
10        int seneye = yas + 1;
11        System.out.println("Merhaba " + ad + "!");
12        System.out.println("Seneye " + seneye + " yaşındasın.");
13        klavye.close();
14    }
15 }
```

System.in tamponu

A y ş e Y ı l m a z ↵ 1 9 ↵

▲
sıradaki

nextInt() rakamları okuyup int
19'a çevirir. Dikkat: satır sonu (↵)
tamponda **kalır**.

Adım 5 / 9

ÇIKTI

```
Adınız: Ayşe Yılmaz
Yaşınız: 19
```

DEĞİŞKENLER

ad = "Ayşe Yılmaz"

yas = 19

Scanner ile Klavyeden Okuma: Adım Adım

```
1 import java.util.Scanner;
2
3 public class Konso10kuma {
4     public static void main(String[] args) {
5         Scanner klavye = new Scanner(System.in);
6         System.out.print("Adınız: ");
7         String ad = klavye.nextLine();
8         System.out.print("Yaşınız: ");
9         int yas = klavye.nextInt();
10        ► int seneye = yas + 1;
11        System.out.println("Merhaba " + ad + "!");
12        System.out.println("Seneye " + seneye + " yaşındasın.");
13        klavye.close();
14    }
15 }
```

System.in tamponu

A y ş e Y ı l m a z ↵ 1 9 ↵

▲
sıradaki

Okunan değer artık sıradan bir `int` : 19
+ 1 = 20.

Adım 6 / 9

ÇIKTI

```
Adınız: Ayşe Yılmaz
Yaşınız: 19
```

DEĞİŞKENLER

ad = "Ayşe Yılmaz"

yas = 19

seneye = 20

Scanner ile Klavyeden Okuma: Adım Adım

```
1 import java.util.Scanner;
2
3 public class Konso10kuma {
4     public static void main(String[] args) {
5         Scanner klavye = new Scanner(System.in);
6         System.out.print("Adınız: ");
7         String ad = klavye.nextLine();
8         System.out.print("Yaşınız: ");
9         int yas = klavye.nextInt();
10        int seneye = yas + 1;
11        ► System.out.println("Merhaba " + ad + "!");
12        System.out.println("Seneye " + seneye + " yaşındasın.");
13        klavye.close();
14    }
15 }
```

System.in tamponu

A y ş e Y ı l m a z ↵ 1 9 ↵

▲
sıradaki

Çıktı akışına satırlar yazılır.

Adım 7 / 9

ÇIKTI

```
Adınız: Ayşe Yılmaz
Yaşınız: 19
Merhaba Ayşe Yılmaz!
```

DEĞİŞKENLER

ad = "Ayşe Yılmaz"

yas = 19

seneye = 20

Scanner ile Klavyeden Okuma: Adım Adım

```
1 import java.util.Scanner;
2
3 public class Konso10kuma {
4     public static void main(String[] args) {
5         Scanner klavye = new Scanner(System.in);
6         System.out.print("Adınız: ");
7         String ad = klavye.nextLine();
8         System.out.print("Yaşınız: ");
9         int yas = klavye.nextInt();
10        int seneye = yas + 1;
11        System.out.println("Merhaba " + ad + "!");
12        ▶ System.out.println("Seneye " + seneye + " yaşındasın.");
13        klavye.close();
14    }
15 }
```

System.in tamponu

A y ş e Y ı l m a z ↵ 1 9 ↵

▲
sıradaki

String birleştirme: "Seneye " + 20 + "
yaşındasın." .

Adım 8 / 9

ÇIKTI

```
Adınız: Ayşe Yılmaz
Yaşınız: 19
Merhaba Ayşe Yılmaz!
Seneye 20 yaşındasın.
```

DEĞİŞKENLER

ad = "Ayşe Yılmaz"

yas = 19

seneye = 20

Scanner ile Klavyeden Okuma: Adım Adım

```
1 import java.util.Scanner;
2
3 public class Konso1Okuma {
4     public static void main(String[] args) {
5         Scanner klavye = new Scanner(System.in);
6         System.out.print("Adınız: ");
7         String ad = klavye.nextLine();
8         System.out.print("Yaşınız: ");
9         int yas = klavye.nextInt();
10        int seneye = yas + 1;
11        System.out.println("Merhaba " + ad + "!");
12        System.out.println("Seneye " + seneye + " yaşındasın.");
13        ▶ klavye.close();
14    }
15 }
```

System.in tamponu

A y ş e Y ı l m a z ↵ 1 9 ↵

▲
sıradaki

`close()` tarayıcıyla birlikte
`System.in` 'i de kapatır; program içinde
bundan sonra klavyeden okunamaz.

Adım 9 / 9

ÇIKTI

```
Adınız: Ayşe Yılmaz
Yaşınız: 19
Merhaba Ayşe Yılmaz!
Seneye 20 yaşındasın.
```

DEĞİŞKENLER

ad = "Ayşe Yılmaz"

yas = 19

seneye = 20

Dosyalar

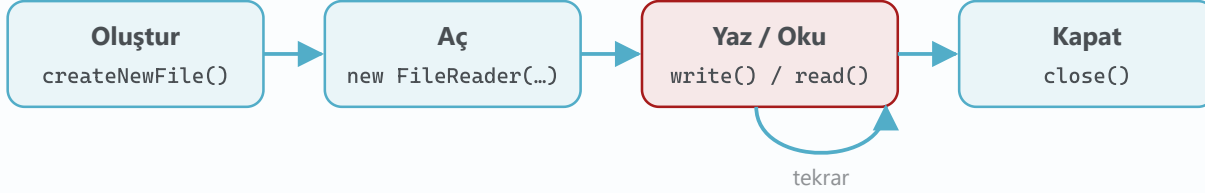
- Dosya, veriyi diskte **kalıcı** olarak saklar: program kapansa, bilgisayar yeniden başlasa da veri durur (RAM'deki değişkenler ise kaybolur).
- Her dosyadaki veri sonuçta **0'lar ve 1'ler**, yani bayt dizisidir. Farkı, bu baytların nasıl **yorumlandığı** yaratır.
- **Metin dosyaları** (text): baytlar bir karakter kodlamasına göre harflere çevrilir; insanlar okuyabilir.
- **İkili dosyalar** (binary): baytlar sayı, piksel, ses örneği, makine komutu gibi verilerdir; ancak onları bilen bir program anlamlandırır.

dosya.txt bir onaltılık (hex) düzenleyicide:

```
0000  4D 65 72 68 61 62 61 2C  Merhaba,  
0008  20 64 C3 BC 6E 79 61 21  d..nya!
```

Solda konum, ortada her baytın onaltılık değeri, sağda ASCII karşılığı. **4D** = 77 = **M**.
15 harf 16 bayt tutar: **ü** UTF-8'de iki bayttır
(**C3 BC**).

Dosya İşleme Süreçleri



Adım	Ne olur?	Java'da
Oluşturma	Diskte yeni, boş bir dosya oluşturulur.	<code>file.createNewFile()</code> ; yazmak için açmak da oluşturur
Açma	Var olan dosyaya bir akış bağlanır.	<code>new FileInputStream(ad)</code> , <code>new FileReader(ad)</code> , <code>new Scanner(file)</code>
Yazma	Veri dosyaya aktarılır.	<code>write()</code> , <code>print()</code> , <code>println()</code>
Okuma	Veri dosyadan programa alınır.	<code>read()</code> , <code>readLine()</code> , <code>nextLine()</code>
Kapatma	Tamponda kalan veri diske yazılır, işletim sistemi kaynağı bırakılır.	<code>close()</code> veya <code>try</code> -with-resources

Kapatılmayan dosya hem **veri kaybına** (tamponda bekleyen baytlar yazılmaz) hem de kaynak sızıntısına yol açar: işletim sisteminin aynı anda açık tutabileceği dosya sayısı sınırlıdır.

İlk Dosya Programı

```
import java.io.*;

public class İlkDosya {
    public static void main(String[] args) throws IOException {
        // Metin dosyasına yazma
        FileWriter writer = new FileWriter("dosya.txt");
        writer.write("Merhaba, dünya!");
        writer.close();
        // İkili dosyaya bir byte yazma
        FileOutputStream cikis = new FileOutputStream("veri.dat");
        cikis.write(127);
        cikis.close();
        // İkili dosyadan okuma
        FileInputStream stream = new FileInputStream("veri.dat");
        int veri = stream.read();
        stream.close();
        System.out.println("Okunan byte: " + veri);
        File metin = new File("dosya.txt");
        System.out.println("dosya.txt: " + metin.length() + " byte");
    }
}
```

ÇIKTI

Okunan byte: 127
dosya.txt: 16 byte

DOSYA.TXT

Merhaba, dünya!

- **FileWriter** : **metin** yazar.
- **FileOutputStream** : **ham bayt** yazar; **write(127)** tek bayt **7F**.
- **read()** : baytı 0–255 arası **int** döndürür.

Metin Dosyaları

- Metin dosyaları **yazdırılabilir karakterler** (harf, rakam, noktalama, boşluk, satır sonu) içerir.
- Not Defteri, VS Code gibi **metin düzenleyicileriyle** açılıp düzenlenebilir. Örnek: kaynak kodlar (`.java` , `.c`), `.txt` , `.csv` , `.html` .
- **Karakter kümesi / kodlama** (character set / encoding): her karakterin hangi sayıyla (baytlarla) saklanacağını belirleyen tablodur. Yazan ve okuyan **aynı kodlamayı** kullanmalıdır.

ASCII

7 bit, 128 karakter:
İngilizce harfler,
rakamlar,
noktalama, kontrol
karakterleri.

ISO-8859-1

8 bit, 256 karakter:
Batı Avrupa (Latin-1). é, ü, ö, ç var; **ş, ğ, ı yok.**

ISO-8859-9

8 bit, Latin-5:
Türkçe sürüm.
Latin-1'deki 6 harfin
yerine ğ, Ğ, ı, İ, ş, Ş
gelir.

UTF-8

Unicode'un
evrensel kodlaması:
tüm dünya dilleri,
1–4 bayt. Web'in ve
Java 18+
sürümünün
varsayılanı.

İkili Dosyalar

- Baytları karakter olarak değil, programın belirlediği **biçimde** (format) saklar: sayılar, pikseller, ses örnekleri, makine kodu.
- Bilgisayar (doğru program) için anlamlıdır; insanlar **doğrudan okuyamaz**.
- Örnekler: çalıştırılabilir dosyalar (`.exe` , Java'nın `.class` dosyaları), görüntü (`.png` , `.jpg`), müzik (`.mp3`), video (`.mp4`), veri dosyaları (`.dat`).
- Avantajı: daha **küçük** ve daha **hızlı** işlenir; sayıyı metne çevirmek gerekmez.

IlkDosya.class hex düzenleyicide (ilk 32 bayt):

```
0000  CA FE BA BE 00 00 00 41  ....A
0008  00 5C 0A 00 02 00 03 07  .\.....
0010  00 04 0C 00 05 00 06 01  ....
0018  00 10 6A 61 76 61 2F 6C  ..java/1
```

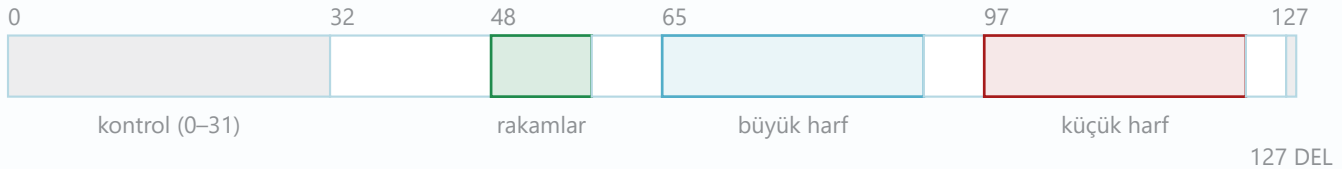
Aynı dosya Not Defteri ile açılınca:

```
Ëþº%...A\.....java/lang/Object...<init>..
..()V.....java/io/Writer.....dosya.txt.....
.....(
```

Her `.class` dosyası `CA FE BA BE` baytlarıyla başlar: JVM dosyanın türünü bu "sihirli sayıdan" tanır.

ASCII Tablosu

- **ASCII** (American Standard Code for Information Interchange): bilgisayarlar arasında metin değişimini standartlaştırmak için 1963'te tanımlanan karakter kodlamasıdır.
- **7 bit** kullanır: $2^7 = 128$ karakter (0–127). Her karakter bir **sayı** ile temsil edilir.
- **Temel karakterler:** harf, rakam, noktalama, temel semboller (32–126). **Kontrol karakterleri:** iletimde özel işlevli, ekrana basılmayan karakterler (0–31 ve 127): satır sonu, sekme, zil...



Karakter	A	a	0	boşluk	\n (LF)	\t sekme
Onluk	65	97	48	32	10	9
Onaltılık	41	61	30	20	0A	09
İkilik	01000001	01100001	00110000	00100000	00001010	00001001

Büyük ve küçük harf arasındaki fark her zaman 32'dir (A = 65, a = 97): tek bir bitin farkı.

Java'da char ve Kod Değeri

```
public class AsciiOrnek {  
    public static void main(String[] args) {  
        char harf = 'A';  
        int kod = harf; // char -> int: 65  
        System.out.println(harf + " = " + kod);  
        System.out.println((int) 'a'); // 97  
        System.out.println((char) 97); // a  
        System.out.println((char) (harf + 2)); // C  
        System.out.println('7' - '0'); // 7 (rakamın değeri)  
        System.out.println(Integer.toBinaryString('A'));  
    }  
}
```

ÇIKTI

```
A = 65  
97  
a  
C  
7  
1000001
```

- **char** 16 bitlik bir **sayıdır**; ilk 128 değeri ASCII ile aynı.
- **int** → **char** için **(char)** dönüşümü gerekir.
- **'7' - '0'** = 55 - 48 = 7.

Geniřletilmiř ASCII ve Trke Karakterler

- **Geniřletilmiř ASCII** (extended ASCII): 8 bit (1 bayt) kullanarak $2^8 = 256$ karakter temsil eder.
- **0–127**: standart ASCII karakterleri, her tabloda aynıdır. **128–255**: zel karakterler, semboller ve ulusal harfler.
- Tek bir "geniřletilmiř ASCII" yoktur: st yarı **kod sayfasına** (code page) gre deęiřir. Trke iin **ISO-8859-9** (Latin-5) ve **Windows-1254** kullanılır.

Bayt	D0	DD	DE	F0	FD	FE	FC	E7
ISO-8859-1 (Batı Avrupa)	Đ	Ÿ	Ɔ	ř	ý	Ɔ	ü	
ISO-8859-9 (Trke)	Ğ	İ	Ş	ğ	ı	ş	ü	



Aynı bayt (FE) bir tabloda ř, tekinde Ɔ'dir. Dosyayı yanlış kod sayfasıyla amak "řimdi" gibi bozuk metinler retir. zm: tm dilleri kapsayan Unicode / UTF-8.

Unicode ve UTF-8

- **Unicode**, dünyadaki her karaktere tek bir numara (**kod noktası**, code point) verir: `A` = 65, `ş` = 351 (U+015F), `€` = 8364.
- **UTF-8** bu numarayı **1–4 bayt** ile yazar: ASCII karakterleri yine 1 bayt (geriye uyumlu), Türkçe harfler 2 bayt, € 3 bayt. **UTF-16** çoğu karakter için 2 bayt kullanır; Java'nın `char` tipi UTF-16 birimidir.

Karakter	Kod noktası	ASCII	ISO-8859-9	UTF-8	UTF-16
A	65	41	41	41	00 41
7	55	37	37	37	00 37
ş	351	3F	FE	C5 9F	01 5F
ğ	287	3F	F0	C4 9F	01 1F
ı	305	3F	FD	C4 B1	01 31
€	8364	3F	3F	E2 82 AC	20 AC

`3F` = `?` : kodlamada karşılığı olmayan karakter soru işaretine dönüşür (veri kaybı). Bu tablo `"ş".getBytes(StandardCharsets.UTF_8)` gibi çağrılarla Java'da üretildi.

Deneyin: Karakter Kodlama

Metin:

Işık 7€

Karakter	Kod noktası	ASCII	ISO-8859-9	UTF-8	UTF-16
I	73	49	49	49	00 49
ş	351	3F	FE	C5 9F	01 5F
ı	305	3F	FD	C4 B1	01 31
k	107	6B	6B	6B	00 6B
_	32	20	20	20	00 20
7	55	37	37	37	00 37
€	8364	3F	3F	E2 82 AC	20 AC
Toplam		7 byte	7 byte	11 byte	14 byte

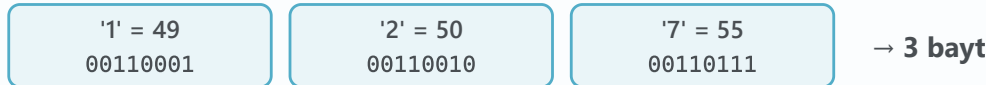
3 karakter ASCII dışında: ASCII ile kaydedilirse **kaybolur**. UTF-8 hepsini korur.

Kırmızı **3F**: o kodlamada karşılığı yok, Java **getBytes** ile yazarken **?** koyar. "Toplam" satırı, bu metni o kodlamayla kaydeden bir dosyanın kaç bayt olacağını gösterir.

Bir Dosyaya Sayı (127) Yazma

- Bir sayının dosyada nasıl saklanacağı veri tipine, kullanım amacına ve dosya biçimine bağlıdır.
- **Metin olarak:** sayı önce karakterlere çevrilir; her rakam bir ASCII baytı tutar. Dikkat: '1' karakterinin kodu 49'dur, 1 değil.
- **İkili olarak:** sayının ikilik değeri (0111 1111) tipin boyutu kadar bayta doğrudan yazılır. Sayısal veri için daha verimlidir.

Metin dosyası (ASCII): her rakam bir karakter



İkili dosya: sayının kendisi



Metin mi, İkili mi? Karşılaştırma

127 nasıl yazıldı?	Boyut
ASCII / UTF-8 metin dosyası	3 bayt
UTF-16 metin dosyası (BOM'suz)	6 bayt
İkili, <code>byte</code>	1 bayt
İkili, <code>short</code>	2 bayt
İkili, <code>int</code>	4 bayt

- **Metin:** insanlar okuyabilir, her düzenleyiciyle açılır, platformlar arası kolay taşınır. Kayıt dosyaları, ayarlar, CSV tabloları.
- **İkili:** sayısal veriyi daha küçük ve hızlı saklar; ama yazan programın biçimini bilmek gerekir. Resim, ses, veri tabanı dosyaları.
- `int` her zaman 4 bayttır: 7 de, 2 000 000 000 de. Metinde ise boyut basamak sayısına bağlıdır (7 → 1 bayt, 2 000 000 000 → 10 bayt).

metin (UTF-8): 3 bayt

31	32	37
'1'	'2'	'7'

ikili int (writeInt): 4 bayt

00	00	00	7F
7F = 127			

Kullanım alanları: ASCII / metin, insanların okuyacağı veriler için; ikili dosyalar, sayısal verilerin verimli saklanması için kullanılır.

Java araçları: `DataOutputStream` ilkel tipleri **ikili** yazar (`writeByte` , `writeShort` , `writeInt` ...), `DataInputStream` aynı sırayla geri okur; `OutputStreamWriter` karakterleri verilen kodlamayla bayta çevirir.

Kodla Doğrulama: 127

```
public class Sayi127 {  
    static void metinYaz(String ad, Charset kod) throws IOException {  
        Writer w = new OutputStreamWriter(new FileOutputStream(ad), kod);  
        w.write(String.valueOf(127));           // "127": 3 karakter  
        w.close();  
    }  
    static void ikiliYaz(String ad, int boyut) throws IOException {  
        FileOutputStream fos = new FileOutputStream(ad);  
        DataOutputStream d = new DataOutputStream(fos);  
        if (boyut == 1) d.writeByte(127);       // 01111111  
        else if (boyut == 2) d.writeShort(127); // 00000000 01111111  
        else d.writeInt(127);                   // 4 byte  
        d.close();  
    }  
    public static void main(String[] args) throws IOException {  
        metinYaz("metin.txt", StandardCharsets.US_ASCII);  
        metinYaz("metin16.txt", StandardCharsets.UTF_16BE);  
        ikiliYaz("b.dat", 1); ikiliYaz("s.dat", 2); ikiliYaz("i.dat", 4);  
        String[] adlar = {"metin.txt", "metin16.txt",  
                           "b.dat", "s.dat", "i.dat"};  
        for (String ad : adlar)  
            System.out.println(ad + ": " + new File(ad).length());  
    }  
}
```

ÇIKTI (BAYT)

```
metin.txt: 3  
metin16.txt: 6  
b.dat: 1  
s.dat: 2  
i.dat: 4
```

java.io.File Sınıfı

- `File` nesnesi diskteki bir **yolu** (path) temsil eder: bir dosya ya da klasör. Dosyanın **içeriğini** okumaz, yazmaz; bunu akışlar yapar.
- Dosya sistemine erişip şu soruları cevaplar:
- Yol var mı yok mu? Bir dosya mı, klasör mü?
- Okunabilir, yazılabilir, çalıştırılabilir mi? Boyutu ve son değiştirilme zamanı ne?
- Yeni dosya veya klasör **oluşturabilir**, var olanı **silebilir**, bir klasörün **içeriğini listeleyebilir**.
- `new File("ornek.txt")` diskte hiçbir şey oluşturmaz; yalnızca yolu tutar.

Metot	Döndürdüğü
<code>exists()</code>	yol var mı (<code>boolean</code>)
<code>isFile()</code> / <code>isDirectory()</code>	dosya mı / klasör mü
<code>canRead()</code> , <code>canWrite()</code> , <code>canExecute()</code>	izinler
<code>createNewFile()</code>	oluşturuldu mu (varsa <code>false</code>)
<code>mkdir()</code> / <code>mkdirs()</code>	klasör (ara klasörlerle)
<code>delete()</code>	silindi mi (dolu klasör silinmez)
<code>list()</code> / <code>listFiles()</code>	<code>String[]</code> / <code>File[]</code>
<code>length()</code>	boyut (bayt, <code>long</code>)
<code>lastModified()</code>	1970'ten beri milisaniye
<code>getName()</code> , <code>getAbsolutePath()</code>	ad, tam yol

File ile Dosya ve Klasör İşlemleri

```
public class DosyaBilgi {  
    public static void main(String[] args) throws IOException {  
        File klasor = new File("ornek");  
        File dosya = new File("ornek/dosya.txt");  
        System.out.println("var mı: " + dosya.exists());  
        System.out.println("mkdir: " + klasor.mkdir());  
        System.out.println("oluşturdu: " + dosya.createNewFile());  
        System.out.println("dosya mı: " + dosya.isFile());  
        System.out.println("klasör mü: " + dosya.isDirectory());  
        System.out.println("okunur: " + dosya.canRead());  
        System.out.println("yazılır: " + dosya.canWrite());  
        System.out.println("boyut: " + dosya.length());  
        String[] icerik = klasor.list();  
        System.out.println("klasörde: " + icerik.length + " öğe");  
        long zaman = dosya.lastModified();  
        System.out.println("zaman > 0: " + (zaman > 0));  
        System.out.println("silindi: " + dosya.delete());  
        System.out.println("var mı: " + dosya.exists());  
        klasor.delete();  
    }  
}
```

ÇIKTI

```
var mı: false  
mkdir: true  
oluşturdu: true  
dosya mı: true  
klasör mü: false  
okunur: true  
yazılır: true  
boyut: 0  
klasörde: 1 öğe  
zaman > 0: true  
silindi: true  
var mı: false
```

- Yol ayırıcı olarak `/` her sistemde çalışır; `File.separator` da kullanılabilir.
- Göreli yol, programın çalıştığı klasöre göreler.

Scanner ile Dosya Okuma

```
public class ScannerDosya {  
    public static void main(String[] args) {  
        Scanner scanner = null;  
        try {  
            scanner = new Scanner(new File("ornekDosya.txt"));  
            // Dosyanın içeriğini okuma ve ekrana yazdırma  
            while (scanner.hasNextLine()) {  
                String satir = scanner.nextLine();  
                System.out.println(satir);  
            }  
        } catch (FileNotFoundException e) {  
            System.err.println("Dosya bulunamadı: " + e.getMessage());  
        } finally {  
            if (scanner != null)  
                scanner.close();  
        }  
    }  
}
```

ÇIKTI

```
Java  
Giriş Çıkış  
Akışlar
```

- Dosya yoksa kontrol edilen bir istisna fırlatılır: `catch` zorunlu.
- `hasNextLine()` : satır var mı?
`nextLine()` : satırı al.
- Dosya açılmazsa `scanner` `null` kalır: `finally` 'deki kontrol şart.

`Scanner` dosya içeriğini okumak ve işlemek için kullanılır; klavye ve `String` gibi diğer girdileri de okur. Gereken importlar: `java.io.File`, `java.io.FileNotFoundException`, `java.util.Scanner`.

try-with-resources (Java 7+)

```
public class ScannerTwr {
    public static void main(String[] args) {
        try (Scanner scanner = new Scanner(new File("notlar.txt"))) {
            int toplam = 0, adet = 0;
            while (scanner.hasNextInt()) {
                toplam += scanner.nextInt();
                adet++;
            }
            double ort = (double) toplam / adet;
            System.out.println(adet + " not, ortalama " + ort);
        } catch (FileNotFoundException e) {
            System.err.println("Dosya bulunamadı: " + e.getMessage());
        }
    }
}
```

ÇIKTI (NOTLAR.TXT: 70 85 / 90 / 45 60)

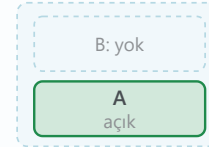
5 not, ortalama 70.0

- Parantezde açılan kaynak blok bitince **otomatik kapatılır**; hata olsa bile.
- `finally` ve `null` kontrolü gerekmez.
- Kaynak `AutoCloseable` olmalı: akışlar, `Scanner`, kanallar.
- Birden çok kaynak `;` ile ayrılır, **ters sırayla** kapanır.
- `nextInt()` sayıları boşluk ve satır sonuna bakmadan okur.

try-with-resources: Kapanış Sırası

```
1 class Kaynak implements AutoCloseable {
2     String ad;
3     Kaynak(String ad) { this.ad = ad; System.out.println(ad
4         + " açıldı"); }
5     public void close() { System.out.println(ad + "
6         kapandı"); }
7 }
8
9 public class KapanisSirasi {
10     public static void main(String[] args) {
11         try (Kaynak a = new Kaynak("A"); Kaynak b = new
12             Kaynak("B")) {
13             System.out.println("blok çalışıyor");
14             throw new IllegalStateException("hata!");
15         } catch (IllegalStateException e) {
16             System.out.println("yakalandı: " + e.getMessage());
17         } finally { System.out.println("finally"); }
18     }
19 }
```

açık kaynaklar (son açılan üstte)



Parantezdeki ilk kaynak oluşturulur; yapıcı mesaj yazar. `a` artık açık bir kaynaktır.

Adım 1 / 8

DEĞİŞKENLER

`a = Kaynak("A")`

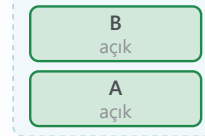
ÇIKTI

A açıldı

try-with-resources: Kapanış Sırası

```
1 class Kaynak implements AutoCloseable {  
2     String ad;  
3     Kaynak(String ad) { this.ad = ad; System.out.println(ad  
+ " açıldı"); }  
4     public void close() { System.out.println(ad + "  
    kapandı"); }  
5 }  
6 public class KapanisSirasi {  
7     public static void main(String[] args) {  
8         try (Kaynak a = new Kaynak("A"); Kaynak b = new  
Kaynak("B")) {  
9             System.out.println("blok çalışıyor");  
10            throw new IllegalStateException("hata!");  
11        } catch (IllegalStateException e) {  
12            System.out.println("yakalandı: " + e.getMessage());  
13        } finally { System.out.println("finally"); }  
14    }  
15 }
```

açık kaynaklar (son açılan üstte)



İkinci kaynak **b** açılır. Kaynaklar **soldan sağa** sırayla oluşturulur.

Adım 2 / 8

DEĞİŞKENLER

a = Kaynak("A")

b = Kaynak("B")

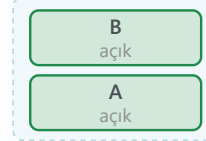
ÇIKTI

```
A açıldı  
B açıldı
```

try-with-resources: Kapanış Sırası

```
1 class Kaynak implements AutoCloseable {  
2     String ad;  
3     Kaynak(String ad) { this.ad = ad; System.out.println(ad  
+ " açıldı"); }  
4     public void close() { System.out.println(ad + "  
    kapandı"); }  
5 }  
6 public class KapanisSirasi {  
7     public static void main(String[] args) {  
8         try (Kaynak a = new Kaynak("A"); Kaynak b = new  
    Kaynak("B")) {  
9             System.out.println("blok çalışıyor");  
10            throw new IllegalStateException("hata!");  
11        } catch (IllegalStateException e) {  
12            System.out.println("yakalandı: " + e.getMessage());  
13        } finally { System.out.println("finally"); }  
14    }  
15 }
```

açık kaynaklar (son açılan üstte)



Blok gövdesi çalışır; iki kaynak da açıktır.

Adım 3 / 8

DEĞİŞKENLER

a = Kaynak("A")

b = Kaynak("B")

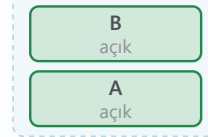
ÇIKTI

```
A açıldı  
B açıldı  
blok çalışıyor
```

try-with-resources: Kapanış Sırası

```
1 class Kaynak implements AutoCloseable {  
2     String ad;  
3     Kaynak(String ad) { this.ad = ad; System.out.println(ad  
4         + " açıldı"); }  
5     public void close() { System.out.println(ad + "  
6         kapandı"); }  
7 }  
8 public class KapanisSirasi {  
9     public static void main(String[] args) {  
10         try (Kaynak a = new Kaynak("A"); Kaynak b = new  
11             Kaynak("B")) {  
12             System.out.println("blok çalışıyor");  
13             ▶ throw new IllegalStateException("hata!");  
14         } catch (IllegalStateException e) {  
15             System.out.println("yakalandı: " + e.getMessage());  
16         } finally { System.out.println("finally"); }  
17     }  
18 }
```

açık kaynaklar (son açılan üstte)



IllegalStateException

İstisna fırlatılır. `catch` 'e geçmeden önce kaynaklar **otomatik** kapatılacak.

Adım 4 / 8

DEĞİŞKENLER

a = Kaynak("A")

b = Kaynak("B")

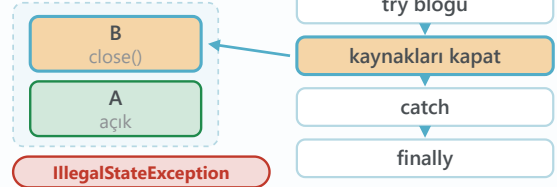
ÇIKTI

```
A açıldı  
B açıldı  
blok çalışıyor
```

try-with-resources: Kapanış Sırası

```
1 class Kaynak implements AutoCloseable {  
2     String ad;  
3     Kaynak(String ad) { this.ad = ad; System.out.println(ad  
    + " açıldı"); }  
4     ▶ public void close() { System.out.println(ad + "  
    kapandı"); }  
5 }  
6 public class KapanisSirasi {  
7     public static void main(String[] args) {  
8         try (Kaynak a = new Kaynak("A"); Kaynak b = new  
    Kaynak("B")) {  
9             System.out.println("blok çalışıyor");  
10            throw new IllegalStateException("hata!");  
11        } catch (IllegalStateException e) {  
12            System.out.println("yakalandı: " + e.getMessage());  
13        } finally { System.out.println("finally"); }  
14    }  
15 }
```

açık kaynaklar (son açılan üstte)



Kapatma **ters sırayla** başlar: önce en son açılan `b` 'nin `close()` metodu çağrılır.

Adım 5 / 8

DEĞİŞKENLER

`this.ad = "B"`

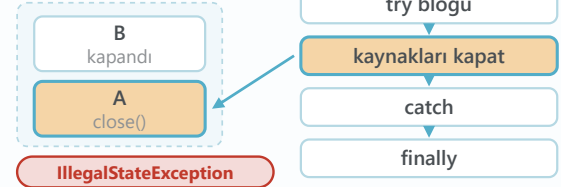
ÇIKTI

```
A açıldı  
B açıldı  
blok çalışıyor  
B kapandı
```

try-with-resources: Kapanış Sırası

```
1 class Kaynak implements AutoCloseable {  
2     String ad;  
3     Kaynak(String ad) { this.ad = ad; System.out.println(ad  
    + " açıldı"); }  
4     ▶ public void close() { System.out.println(ad + "  
    kapandı"); }  
5 }  
6 public class KapanisSirasi {  
7     public static void main(String[] args) {  
8         try (Kaynak a = new Kaynak("A"); Kaynak b = new  
            Kaynak("B")) {  
9             System.out.println("blok çalışıyor");  
10            throw new IllegalStateException("hata!");  
11        } catch (IllegalStateException e) {  
12            System.out.println("yakalandı: " + e.getMessage());  
13        } finally { System.out.println("finally"); }  
14    }  
15 }
```

açık kaynaklar (son açılan üstte)



Ardından `a.close()` çağrılır. İstisna hâlâ yolda; henüz yakalanmadı.

Adım 6 / 8

DEĞİŞKENLER

`this.ad = "A"`

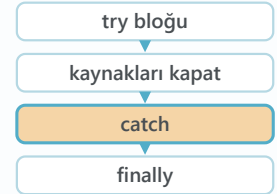
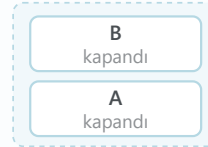
ÇIKTI

```
A açıldı  
B açıldı  
blok çalışıyor  
B kapandı  
A kapandı
```

try-with-resources: Kapanış Sırası

```
1 class Kaynak implements AutoCloseable {  
2     String ad;  
3     Kaynak(String ad) { this.ad = ad; System.out.println(ad  
4         + " açıldı"); }  
5     public void close() { System.out.println(ad + "  
6         kapandı"); }  
7 }  
8 public class KapanisSirasi {  
9     public static void main(String[] args) {  
10         try (Kaynak a = new Kaynak("A"); Kaynak b = new  
11             Kaynak("B")) {  
12             System.out.println("blok çalışıyor");  
13             throw new IllegalStateException("hata!");  
14         } catch (IllegalStateException e) {  
15             System.out.println("yakalandı: " + e.getMessage());  
16         } finally { System.out.println("finally"); }  
17     }  
18 }
```

açık kaynaklar (son açılan üstte)



Kaynaklar kapandıktan **sonra** `catch` istisnayı yakalar. Bu noktada `a` ve `b` kapsam dışıdır.

Adım 7 / 8

DEĞİŞKENLER

```
e = IllegalStateException("hata!")
```

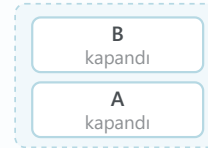
ÇIKTI

```
A açıldı  
B açıldı  
blok çalışıyor  
B kapandı  
A kapandı  
yakalandı: hata!
```

try-with-resources: Kapanış Sırası

```
1 class Kaynak implements AutoCloseable {
2     String ad;
3     Kaynak(String ad) { this.ad = ad; System.out.println(ad
4         + " açıldı"); }
5     public void close() { System.out.println(ad + "
6         kapandı"); }
7 }
8 public class KapanisSirasi {
9     public static void main(String[] args) {
10         try (Kaynak a = new Kaynak("A"); Kaynak b = new
11             Kaynak("B")) {
12             System.out.println("blok çalışıyor");
13             throw new IllegalStateException("hata!");
14         } catch (IllegalStateException e) {
15             System.out.println("yakalandı: " + e.getMessage());
16         } finally { System.out.println("finally"); }
17     }
18 }
```

açık kaynaklar (son açılan üstte)



finally en son çalışır. Sıra: aç A, aç B, gövde, kapat B, kapat A, **catch**, **finally**.

Adım 8 / 8

DEĞİŞKENLER

—

ÇIKTI

```
A açıldı
B açıldı
blok çalışıyor
B kapandı
A kapandı
yakalandı: hata!
finally
```

Java I/O Kütüphanesi

- `java.io` paketi giriş/çıkış işlemleri için güçlü ve esnek bir araç seti sunar; dosya işlemlerinden ağ programlamaya kadar kullanılır.

Giriş / çıkış

Dosya, ağ, bellek gibi kaynaklardan veri almak ve göndermek.

İkili / metin

Hem bayt tabanlı (ikili) hem karakter tabanlı (metin) işlemleri destekler.

Sıralı / rastgele

Dosyadaki verilere hem sırayla hem istenen konuma atlayarak erişilebilir.

Öğe tipi	Örnekler	Görevi
Sınıflar	<code>File</code>	dosya ve klasör (yol) işlemleri
	<code>InputStream</code> , <code>OutputStream</code>	temel bayt giriş/çıkışı
	<code>Reader</code> , <code>Writer</code>	metin (karakter) giriş/çıkışı
Arayüzler	<code>Closeable</code> , <code>AutoCloseable</code>	kaynağı serbest bırakma: <code>close()</code>
	<code>Flushable</code>	tamponu boşaltma: <code>flush()</code>
İstisnalar	<code>IOException</code>	giriş/çıkış hatalarının üst sınıfı
	<code>FileNotFoundException</code> , <code>EOFException</code>	dosya yok / beklenmedik dosya sonu

Bayt ve Karakter Odaklı Giriş Çıkış

Bayt akışları

- + `InputStream` ve `OutputStream` sınıfları
- + Veri **bayt** (8 bit) dizileri olarak işlenir
- + Resim, ses, video, `.class` gibi **ikili** verilerle çalışır
- + `read()` 0–255 arası bir bayt döndürür

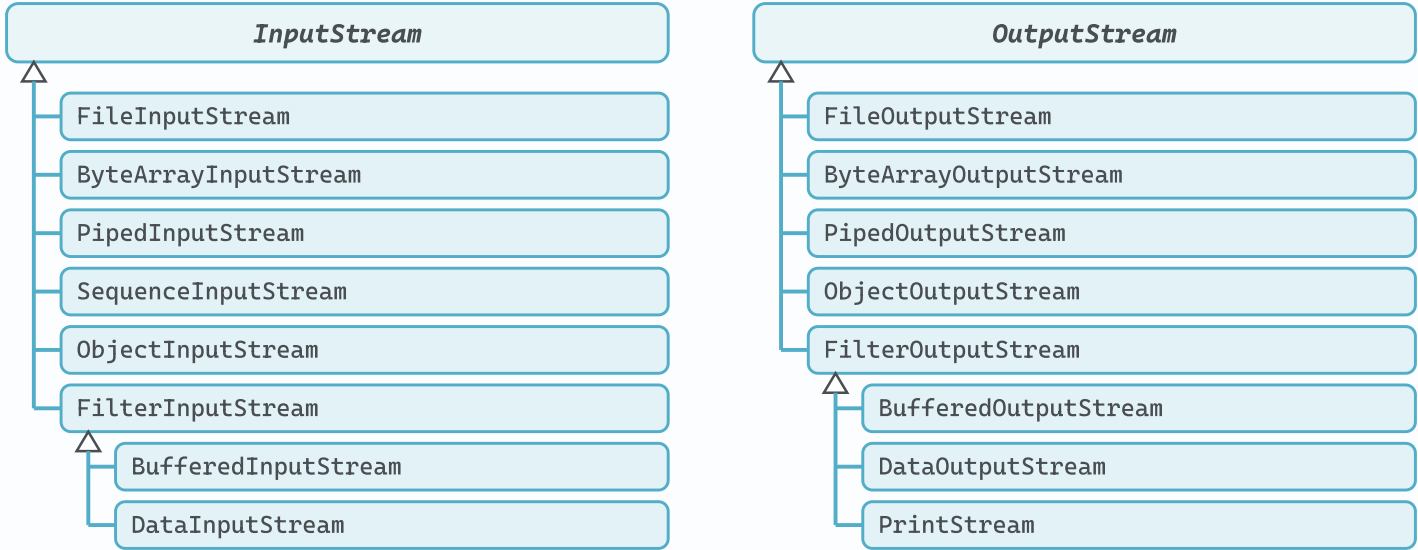
Karakter akışları

- + `Reader` ve `Writer` sınıfları
- + Veri **char** (16 bit Unicode) olarak işlenir
- + **Metin** tabanlı verilerle daha verimli çalışır
- + Kodlamayı (UTF-8...) çözer: farklı dil ve alfabeleri destekler

	Bayt akışı	Karakter akışı
Okuma birimi	<code>byte</code>	<code>char</code>
Dosya sınıfları	<code>FileInputStream</code> , <code>FileOutputStream</code>	<code>FileReader</code> , <code>FileWriter</code>
Tamponlu	<code>BufferedInputStream</code> , <code>BufferedOutputStream</code>	<code>BufferedReader</code> , <code>BufferedWriter</code>
Kullanım	her tür dosya, kopyalama	yalnızca metin dosyaları

Köprü sınıflar: `InputStreamReader` bir bayt akışını karakter akışına, `OutputStreamWriter` karakterleri bayt akışına çevirir (kodlama burada uygulanır).

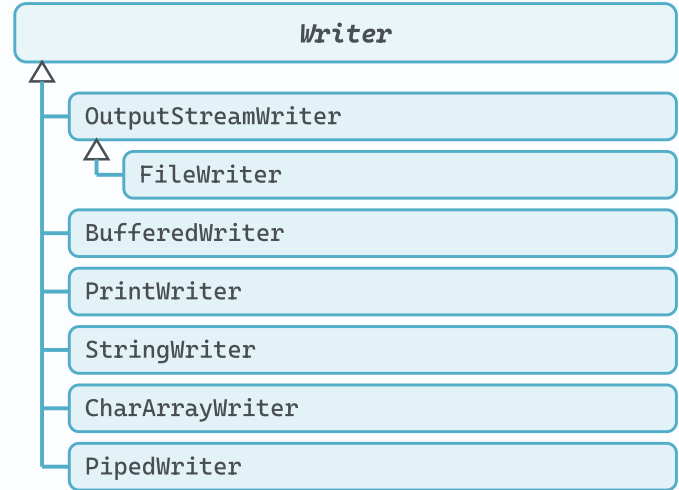
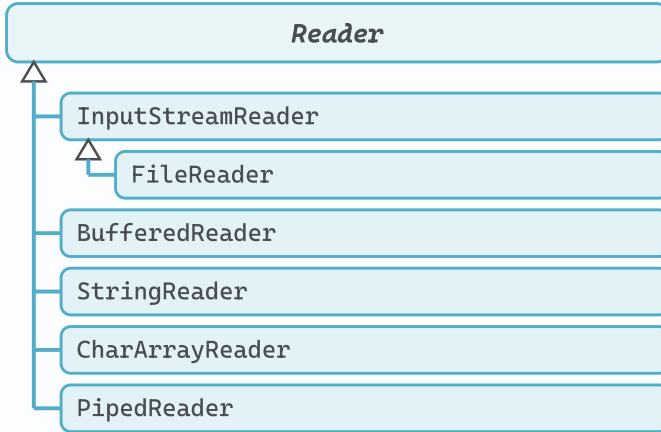
Bayt Akışı Sınıfları



İtalik kutular **soyut** (abstract) sınıflardır; içi boş üçgenli oklar kalıtımı gösterir.

`System.out` bir `PrintStream` 'dir. `Filter...` alt sınıfları başka bir akışı **sarmalayıp** yeni yetenek ekler (tampon, ilkel tip okuma).

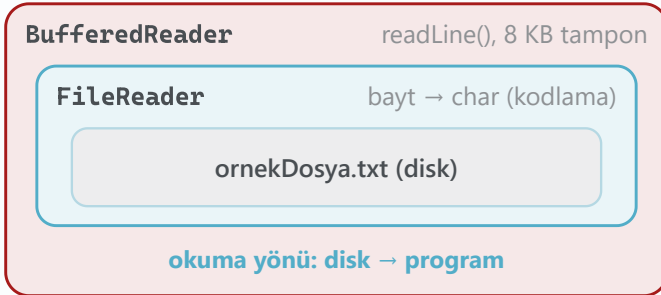
Karakter Akışı Sınıfları



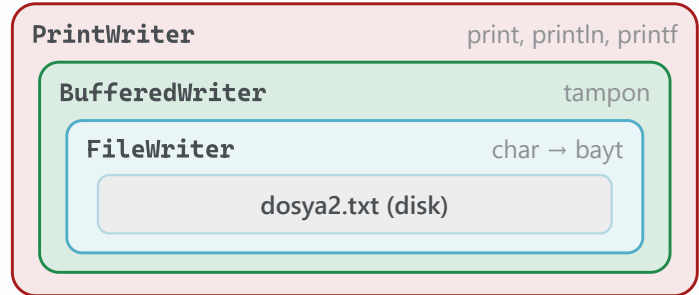
FileReader aslında bir **InputStreamReader** 'dir: **FileInputStream** 'den gelen baytları seçilen kodlamayla karakterlere çevirir. **FileWriter** da aynı şekilde **OutputStreamWriter** 'dan türer.

Sarmalayıcı (Decorator) Akışlar

Okuma zinciri



Yazma zinciri



```
BufferedReader okuyucu = new BufferedReader(new FileReader("ornekDosya.txt"));
PrintWriter yazici = new PrintWriter(new BufferedWriter(new FileWriter("dosya2.txt")));
```

- Her akış sınıfı **tek bir işi** iyi yapar; ihtiyaç duyulan yetenekler akışları iç içe **sarmalayarak** birleştirilir (Decorator tasarım deseni).
- Benzetme: hortumun ucuna önce filtre, sonra püskürtme başlığı takmak. Su (veri) hepsinden sırayla geçer.
- En dıştaki akışı kapatmak (`close()`) içtekilerin hepsini de kapatır.

BufferedReader ile Satır Satır Okuma: Adım Adım

```
1 import java.io.*;
2
3 public class SatirOku {
4     public static void main(String[] args) {
5         ▶ File dosya = new File("ornekDosya.txt");
6         try (BufferedReader okuyucu =
7             new BufferedReader(new FileReader(dosya))) {
8             String satir;
9             while ((satir = okuyucu.readLine()) != null) {
10                 System.out.println(satir);
11             }
12         } catch (IOException e) {
13             System.err.println("Okuma hatası: " + e.getMessage());
14             e.printStackTrace();
15         }
16     }
17 }
```

ornekDosya.txt



Java

Giriş Çıkış

Akışlar

(dosya sonu)

File yalnızca yolu tutar; dosya henüz açılmadı.

Adım 1 / 10

ÇIKTI

DEĞİŞKENLER

dosya = "ornekDosya.txt"

ÇAĞRI YİĞİNİ

main()

BufferedReader ile Satır Satır Okuma: Adım Adım

```
1 import java.io.*;
2
3 public class SatirOku {
4     public static void main(String[] args) {
5         File dosya = new File("ornekDosya.txt");
6         try (BufferedReader okuyucu =
7             new BufferedReader(new FileReader(dosya))) {
8             String satir;
9             while ((satir = okuyucu.readLine()) != null) {
10                 System.out.println(satir);
11             }
12         } catch (IOException e) {
13             System.err.println("Okuma hatası: " + e.getMessage());
14             e.printStackTrace();
15         }
16     }
17 }
```

ornekDosya.txt



Java

Giriş Çıkış

Akışlar

(dosya sonu)

FileReader dosyayı açar, **BufferedReader** onu sarmalar. Okuma konumu dosyanın başında.

Adım 2 / 10

ÇIKTI



DEĞİŞKENLER

okuyucu = BufferedReader

ÇAĞRI YİĞİNİ

main()

BufferedReader ile Satır Satır Okuma: Adım Adım

```
1 import java.io.*;
2
3 public class SatirOku {
4     public static void main(String[] args) {
5         File dosya = new File("ornekDosya.txt");
6         try (BufferedReader okuyucu =
7             new BufferedReader(new FileReader(dosya))) {
8             String satir;
9             ▶ while ((satir = okuyucu.readLine()) != null) {
10                 System.out.println(satir);
11             }
12         } catch (IOException e) {
13             System.err.println("Okuma hatası: " + e.getMessage());
14             e.printStackTrace();
15         }
16     }
17 }
```

ornekDosya.txt

Java

▶ Giriş Çıkış

Akışlar

(dosya sonu)

`readLine()` satırı okur, `␣`'i **atar**. Sonuç `null`
değil → döngüye gir.

Adım 3 / 10

ÇIKTI

DEĞİŞKENLER

`satir = "Java"`

ÇAĞRI YİĞİNİ

`okuyucu.readLine()``main()`

BufferedReader ile Satır Satır Okuma: Adım Adım

```
1 import java.io.*;
2
3 public class SatirOku {
4     public static void main(String[] args) {
5         File dosya = new File("ornekDosya.txt");
6         try (BufferedReader okuyucu =
7             new BufferedReader(new FileReader(dosya))) {
8             String satir;
9             while ((satir = okuyucu.readLine()) != null) {
10                 System.out.println(satir);
11             }
12         } catch (IOException e) {
13             System.err.println("Okuma hatası: " + e.getMessage());
14             e.printStackTrace();
15         }
16     }
17 }
```

ornekDosya.txt

Java

► Giriş Çıkış

Akışlar

(dosya sonu)

Satır ekrana yazılır.

Adım 4 / 10

ÇIKTI

Java

DEĞİŞKENLER

satir = "Java"

ÇAĞRI YİĞİNİ

main()

BufferedReader ile Satır Satır Okuma: Adım Adım

```
1 import java.io.*;
2
3 public class SatirOku {
4     public static void main(String[] args) {
5         File dosya = new File("ornekDosya.txt");
6         try (BufferedReader okuyucu =
7             new BufferedReader(new FileReader(dosya))) {
8             String satir;
9             ▶ while ((satir = okuyucu.readLine()) != null) {
10                 System.out.println(satir);
11             }
12         } catch (IOException e) {
13             System.err.println("Okuma hatası: " + e.getMessage());
14             e.printStackTrace();
15         }
16     }
17 }
```

ornekDosya.txt

Java

Giriş Çıkış

▶ Akışlar

(dosya sonu)

Sıradaki satır okunur: `satir` = "Giriş Çıkış".

Adım 5 / 10

ÇIKTI

Java

DEĞİŞKENLER

`satir = "Giriş Çıkış"`

ÇAĞRI YİĞİNİ

`okuyucu.readLine()``main()`

BufferedReader ile Satır Satır Okuma: Adım Adım

```
1 import java.io.*;
2
3 public class SatirOku {
4     public static void main(String[] args) {
5         File dosya = new File("ornekDosya.txt");
6         try (BufferedReader okuyucu =
7             new BufferedReader(new FileReader(dosya))) {
8             String satir;
9             while ((satir = okuyucu.readLine()) != null) {
10                 System.out.println(satir);
11             }
12         } catch (IOException e) {
13             System.err.println("Okuma hatası: " + e.getMessage());
14             e.printStackTrace();
15         }
16     }
17 }
```

ornekDosya.txt

Java

Giriş Çıkış

▶ Akışlar

(dosya sonu)

Satır ekrana yazılır.

Adım 6 / 10

ÇIKTI

Java

Giriş Çıkış

DEĞİŞKENLER

satir = "Giriş Çıkış"

ÇAĞRI YİĞİNİ

main()

BufferedReader ile Satır Satır Okuma: Adım Adım

```
1 import java.io.*;
2
3 public class SatirOku {
4     public static void main(String[] args) {
5         File dosya = new File("ornekDosya.txt");
6         try (BufferedReader okuyucu =
7             new BufferedReader(new FileReader(dosya))) {
8             String satir;
9             ▶ while ((satir = okuyucu.readLine()) != null) {
10                 System.out.println(satir);
11             }
12         } catch (IOException e) {
13             System.err.println("Okuma hatası: " + e.getMessage());
14             e.printStackTrace();
15         }
16     }
17 }
```

ornekDosya.txt

Java

Giriş Çıkış

Akışlar



(dosya sonu)

Sıradaki satır okunur: `satir` = `"Akışlar"`.

Adım 7 / 10

ÇIKTI

Java

Giriş Çıkış

DEĞİŞKENLER

`satir = "Akışlar"`

ÇAĞRI YİĞİNİ

`okuyucu.readLine()``main()`

BufferedReader ile Satır Satır Okuma: Adım Adım

```
1 import java.io.*;
2
3 public class SatirOku {
4     public static void main(String[] args) {
5         File dosya = new File("ornekDosya.txt");
6         try (BufferedReader okuyucu =
7             new BufferedReader(new FileReader(dosya))) {
8             String satir;
9             while ((satir = okuyucu.readLine()) != null) {
10                 System.out.println(satir);
11             }
12         } catch (IOException e) {
13             System.err.println("Okuma hatası: " + e.getMessage());
14             e.printStackTrace();
15         }
16     }
17 }
```

ornekDosya.txt

Java

Giriş Çıkış

Akışlar

▶ (dosya sonu)

Satır ekrana yazılır.

Adım 8 / 10

ÇIKTI

```
Java
Giriş Çıkış
Akışlar
```

DEĞİŞKENLER

satir = "Akışlar"

ÇAĞRI YİĞİNİ

main()

BufferedReader ile Satır Satır Okuma: Adım Adım

```
1 import java.io.*;
2
3 public class SatirOku {
4     public static void main(String[] args) {
5         File dosya = new File("ornekDosya.txt");
6         try (BufferedReader okuyucu =
7             new BufferedReader(new FileReader(dosya))) {
8             String satir;
9             while ((satir = okuyucu.readLine()) != null) {
10                 System.out.println(satir);
11             }
12         } catch (IOException e) {
13             System.err.println("Okuma hatası: " + e.getMessage());
14             e.printStackTrace();
15         }
16     }
17 }
```

ornekDosya.txt

Java

Giriş Çıkış

Akışlar

(dosya sonu)

Dosya sonu: `readLine()` `null` döndürür →
döngü biter.

Adım 9 / 10

ÇIKTI

```
Java
Giriş Çıkış
Akışlar
```

DEĞİŞKENLER

`satir = null`

ÇAĞRI YİĞİNİ

`okuyucu.readLine()``main()`

BufferedReader ile Satır Satır Okuma: Adım Adım

```
1 import java.io.*;
2
3 public class SatirOku {
4     public static void main(String[] args) {
5         File dosya = new File("ornekDosya.txt");
6         try (BufferedReader okuyucu =
7             new BufferedReader(new FileReader(dosya))) {
8             String satir;
9             while ((satir = okuyucu.readLine()) != null) {
10                 System.out.println(satir);
11             }
12         } catch (IOException e) {
13             System.err.println("Okuma hatası: " + e.getMessage());
14             e.printStackTrace();
15         }
16     }
17 }
```

ornekDosya.txt

Java

Giriş Çıkış

Akışlar

(dosya sonu)

try biter: okuyucu otomatik kapatılır;
catch çalışmaz.

Adım 10 / 10

ÇIKTI

```
Java
Giriş Çıkış
Akışlar
```

DEĞİŞKENLER

—

ÇAĞRI YİĞİNİ

main()

java.io.PrintWriter

```
public class PrintWriterOrnek {  
    public static void main(String[] args) {  
        PrintWriter yazici = null;  
        try {  
            yazici = new PrintWriter("yeniDosya.txt");  
            yazici.println("Merhaba, dünya!");  
            yazici.println("Bu bir örnek dosyadır.");  
            yazici.printf("Pi yaklaşık %.2f%n", Math.PI);  
            yazici.println(42);  
        } catch (IOException e) {  
            System.err.println("Dosya hatası: " + e.getMessage());  
            e.printStackTrace();  
        } finally {  
            if (yazici != null)  
                yazici.close();  
        }  
    }  
}
```

- Metin dosyalarına **kolayca** yazar: `print`, `println`, `printf`.
- Diğer tipleri **metne çevirerek** yazar.
- Dosya yoksa oluşturur, varsa **içini siler**.
- `close` unutulmamalı: veri tamponda kalabilir.

YENİDOSYA.TXT

```
Merhaba, dünya!  
Bu bir örnek dosyadır.  
Pi yaklaşık 3.14  
42
```

java.io.FileWriter

- Dosyalara **karakter tabanlı** veri yazmak için kullanılır; karakter akışına (`Writer`) veri yazma işlevi sağlar.
- Yalnızca `write(String)` , `write(char[])` , `write(int c)` vardır: **metin biçimlendirme yetenekleri sınırlıdır**, satır sonunu (`\n`) kendiniz yazarsınız.
- Daha **düşük seviyeli** ve esnek bir yaklaşımdır; çoğunlukla `BufferedWriter` veya `PrintWriter` ile sarmalanır.
- `new FileWriter(ad, true)` : dosyanın **sonuna ekler** (append); `false` veya parametresiz hâli dosyayı sıfırlar.

	<code>FileWriter</code>	<code>BufferedWriter</code>	<code>PrintWriter</code>
Yazma metotları	<code>write</code>	<code>write</code> , <code>newline()</code>	<code>print</code> , <code>println</code> , <code>printf</code>
Sayı yazmak	önce <code>String</code> 'e çevir	önce <code>String</code> 'e çevir	doğrudan: <code>println(42)</code>
Tampon	küçük iç tampon	8 KB tampon	sardığı akışa bağlı
Hata bildirimi	<code>IOException</code>	<code>IOException</code>	<code>checkError()</code>

FileWriter Örneği: Yazma ve Ekleme

```
public class FileWriterOrnek {  
    public static void main(String[] args) {  
        try (FileWriter yazici = new FileWriter("yeniDosya.txt")) {  
            yazici.write("Merhaba, dünya!\n");  
            yazici.write("Bu bir FileWriter örneğidir.\n");  
            System.out.println("Dosya oluşturuldu ve veri yazıldı.");  
        } catch (IOException e) {  
            System.err.println("Dosya hatası: " + e.getMessage());  
            e.printStackTrace();  
        }  
        // true: dosyanın sonuna ekle (append)  
        try (FileWriter ekle = new FileWriter("yeniDosya.txt", true)) {  
            ekle.write("Sona eklenen satır.\n");  
        } catch (IOException e) {  
            System.err.println("Dosya hatası: " + e.getMessage());  
        }  
    }  
}
```

ÇIKTI

Dosya oluşturuldu ve veri yazıldı.

YENİDOSYA.TXT

Merhaba, dünya!
Bu bir FileWriter örneğidir.
Sona eklenen satır.

Karakter Kodlamasını Belirtmek

```
public class KodlamaOku {  
    public static void main(String[] args) throws IOException {  
        File dosya = new File("turkce.txt");  
        try (BufferedWriter yazici = new BufferedWriter(  
            new FileWriter(dosya, StandardCharsets.UTF_8))) {    // Java 11+  
            yazici.write("Işık, şimşek, çiğ, öğrenci, Ünye");  
        }  
        try (BufferedReader dogru = new BufferedReader(  
            new FileReader(dosya, StandardCharsets.UTF_8))) {  
            System.out.println(dogru.readLine());  
        }  
        try (BufferedReader yanlis = new BufferedReader(  
            new FileReader(dosya, StandardCharsets.ISO_8859_1))) {  
            System.out.println(yanlis.readLine());  
        }  
    }  
}
```

ÇIKTI

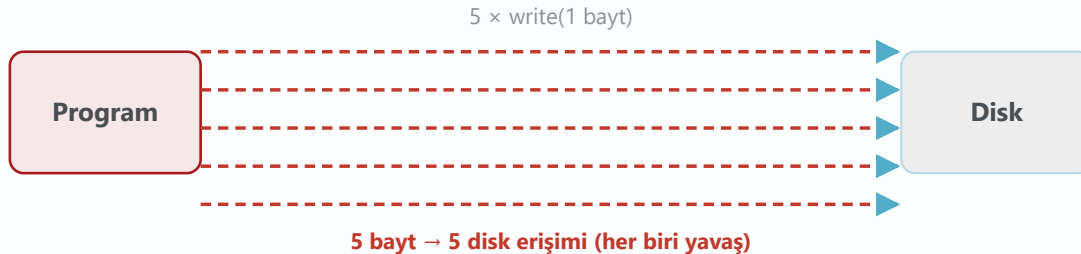
```
Işık, şimşek, çiğ, öğrenci, Ünye  
IÄÄ±k, ÅimÅek, ÅşiÄ, ÅğÄrenci, Ånye
```

- Kodlama verilmezse **varsayılan** kullanılır: Java 18+ UTF-8, eskiden işletim sisteminin ayarı (Windows-1254). Taşınabilir kod için kodlamayı **açıkça** yazın; Java 10 ve öncesinde `InputStreamReader` köprüsüyle.

Tamponlama (Buffering)

- Temel metotlar: `read()` bir bayt veya bayt dizisi **okur**, `write()` bir bayt veya bayt dizisi **yazar**.
- Her bayt için ayrı disk erişimi uygulamayı **ciddi şekilde yavaşlatır**: disk, bellekten binlerce kat yavaştır.
- Çözüm: veriyi bellekte bir **tamponda** (buffer) biriktirip diskten okuma / diske yazma işlemlerini **toplu** yapmak. Fiziksel disk erişimi sayısı azalır.

Benzetme: marketten 10 ürünü tek tek, her biri için ayrı yolculukla taşımak yerine bir sepete doldurup **tek seferde** taşımak.

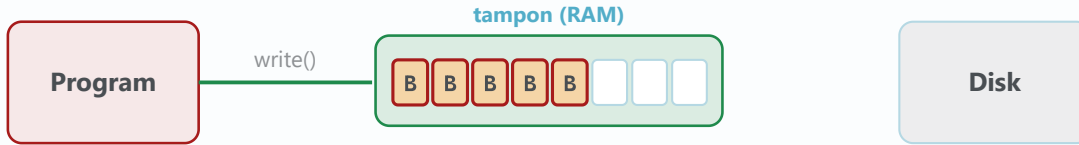


Tamponsuz yazma: her `write()` çağrısı doğrudan diske gider.

Tamponlama (Buffering)

- Temel metotlar: `read()` bir bayt veya bayt dizisi **okur**, `write()` bir bayt veya bayt dizisi **yazar**.
- Her bayt için ayrı disk erişimi uygulamayı **ciddi şekilde yavaşlatır**: disk, bellekten binlerce kat yavaştır.
- Çözüm: veriyi bellekte bir **tamponda** (buffer) biriktirip diskten okuma / diske yazma işlemlerini **toplu** yapmak. Fiziksel disk erişimi sayısı azalır.

Benzetme: marketten 10 ürünü tek tek, her biri için ayrı yolculukla taşımak yerine bir sepete doldurup **tek seferde** taşımak.



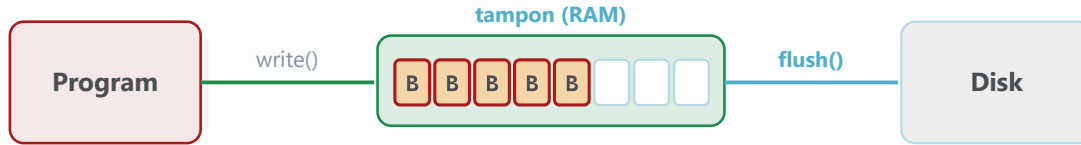
5 write() → bellekte birikti, disk erişimi yok

Tamponlu yazma: `write()` çağrılarını baytları önce bellekteki tampona koyar.

Tamponlama (Buffering)

- Temel metotlar: `read()` bir bayt veya bayt dizisi **okur**, `write()` bir bayt veya bayt dizisi **yazar**.
- Her bayt için ayrı disk erişimi uygulamayı **ciddi şekilde yavaşlatır**: disk, bellekten binlerce kat yavaştır.
- Çözüm: veriyi bellekte bir **tamponda** (buffer) biriktirip diskten okuma / diske yazma işlemlerini **toplu** yapmak. Fiziksel disk erişimi sayısı azalır.

Benzetme: marketten 10 ürünü tek tek, her biri için ayrı yolculukla taşımak yerine bir sepete doldurup **tek seferde** taşımak.



flush / close / tampon doldu → 1 disk erişimi

Tampon dolunca, `flush()` veya `close()` çağrılınca tamponun tamamı tek seferde diske yazılır.

Tamponlu Akış Sınıfları

Sınıf	Sardığı akış	Ne yapar?
BufferedInputStream	InputStream	okumayı hızlandırır: <code>read()</code> çağrıldığında diskten tek seferde 8192 baytı bellekteki önbelleğe alır
BufferedReader	Reader	aynısı, karakterler için; ayrıca <code>readLine()</code>
BufferedOutputStream	OutputStream	yazmayı hızlandırır: <code>write()</code> verisini önbellekte biriktirir
BufferedWriter	Writer	aynısı, karakterler için; ayrıca <code>newLine()</code>

```
public class FlushOrnek {  
    public static void main(String[] args) throws IOException {  
        File log = new File("log.txt");  
        BufferedWriter bw = new BufferedWriter(new FileWriter(log));  
        bw.write("Kayıt 1");  
        System.out.println("önce: " + log.length());  
        bw.flush();  
        System.out.println("sonra: " + log.length());  
        bw.close();  
    }  
}
```

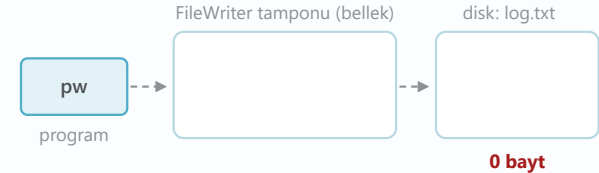
ÇIKTI

önce: 0
sonra: 8

`flush()` tamponu hemen diske boşaltır; `close()` önce `flush()` yapar. Kapatılmayan tamponlu yazıcıda son veriler **kaybolur**.

Tampon ve flush: Adım Adım

```
► File f = new File("log.txt");
► PrintWriter pw = new PrintWriter(new FileWriter(f));
3 System.out.println("0: " + f.length());
4 pw.print("kayit1;");
5 System.out.println("1: " + f.length());
6 pw.print("kayit2;");
7 System.out.println("2: " + f.length());
8 pw.flush();
9 System.out.println("3: " + f.length());
10 pw.print("kayit3;");
11 pw.close();
12 System.out.println("4: " + f.length());
13 pw.print("kayit4;"); // kapalı: sessizce yok sayılır
14 System.out.println("hata? " + pw.checkError());
```



FileWriter dosyayı oluşturur (boş). Yazılanlar önce bellekteki **tampona** gider.

Adım 1 / 11

DEĞİŞKENLER

f = log.txt

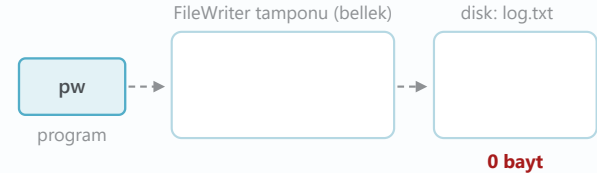
pw = PrintWriter

ÇIKTI



Tampon ve flush: Adım Adım

```
1 File f = new File("log.txt");
2 PrintWriter pw = new PrintWriter(new FileWriter(f));
3 ▶ System.out.println("0: " + f.length());
4 pw.print("kayit1;");
5 System.out.println("1: " + f.length());
6 pw.print("kayit2;");
7 System.out.println("2: " + f.length());
8 pw.flush();
9 System.out.println("3: " + f.length());
10 pw.print("kayit3;");
11 pw.close();
12 System.out.println("4: " + f.length());
13 pw.print("kayit4;"); // kapalı: sessizce yok sayılır
14 System.out.println("hata? " + pw.checkError());
```



Henüz hiçbir şey yazılmadı: 0 bayt.

Adım 2 / 11

DEĞİŞKENLER

`f.length()` = 0

ÇIKTI

0: 0

Tampon ve flush: Adım Adım

```
1 File f = new File("log.txt");
2 PrintWriter pw = new PrintWriter(new FileWriter(f));
3 System.out.println("0: " + f.length());
▶ pw.print("kayit1;");
5 System.out.println("1: " + f.length());
6 pw.print("kayit2;");
7 System.out.println("2: " + f.length());
8 pw.flush();
9 System.out.println("3: " + f.length());
10 pw.print("kayit3;");
11 pw.close();
12 System.out.println("4: " + f.length());
13 pw.print("kayit4;"); // kapalı: sessizce yok sayılır
14 System.out.println("hata? " + pw.checkError());
```



print 7 karakteri **tampona** koyar; diske dokunmaz.

Adım 3 / 11

DEĞİŞKENLER

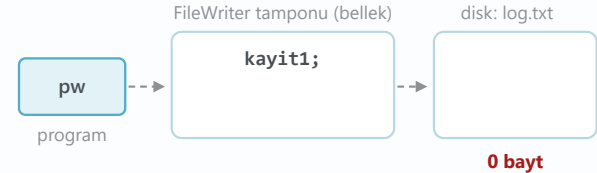
tampon = "kayit1"

ÇIKTI

0: 0

Tampon ve flush: Adım Adım

```
1 File f = new File("log.txt");
2 PrintWriter pw = new PrintWriter(new FileWriter(f));
3 System.out.println("0: " + f.length());
4 pw.print("kayit1;");
5 ► System.out.println("1: " + f.length());
6 pw.print("kayit2;");
7 System.out.println("2: " + f.length());
8 pw.flush();
9 System.out.println("3: " + f.length());
10 pw.print("kayit3;");
11 pw.close();
12 System.out.println("4: " + f.length());
13 pw.print("kayit4;"); // kapalı: sessizce yok sayılır
14 System.out.println("hata? " + pw.checkError());
```



Dosya hâlâ 0 bayt: veri bellekte bekliyor.

Adım 4 / 11

DEĞİŞKENLER

f.length() = 0

ÇIKTI

```
0: 0
1: 0
```

Tampon ve flush: Adım Adım

```
1 File f = new File("log.txt");
2 PrintWriter pw = new PrintWriter(new FileWriter(f));
3 System.out.println("0: " + f.length());
4 pw.print("kayit1;");
5 System.out.println("1: " + f.length());
▶ pw.print("kayit2;");
▶ System.out.println("2: " + f.length());
8 pw.flush();
9 System.out.println("3: " + f.length());
10 pw.print("kayit3;");
11 pw.close();
12 System.out.println("4: " + f.length());
13 pw.print("kayit4;"); // kapalı: sessizce yok sayılır
14 System.out.println("hata? " + pw.checkError());
```



İkinci kayıt da tampona eklenir; dosya yine 0 bayt.

Adım 5 / 11

DEĞİŞKENLER

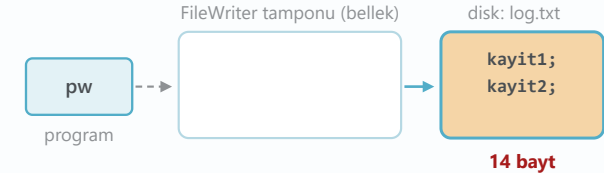
tampon = "kayit1"

ÇIKTI

```
0: 0
1: 0
2: 0
```

Tampon ve flush: Adım Adım

```
1 File f = new File("log.txt");
2 PrintWriter pw = new PrintWriter(new FileWriter(f));
3 System.out.println("0: " + f.length());
4 pw.print("kayit1;");
5 System.out.println("1: " + f.length());
6 pw.print("kayit2;");
7 System.out.println("2: " + f.length());
8 ► pw.flush();
9 System.out.println("3: " + f.length());
10 pw.print("kayit3;");
11 pw.close();
12 System.out.println("4: " + f.length());
13 pw.print("kayit4;"); // kapalı: sessizce yok sayılır
14 System.out.println("hata? " + pw.checkError());
```



flush() tamponu diske boşaltır: 14 bayt yazılır.

Adım 6 / 11

DEĞİŞKENLER

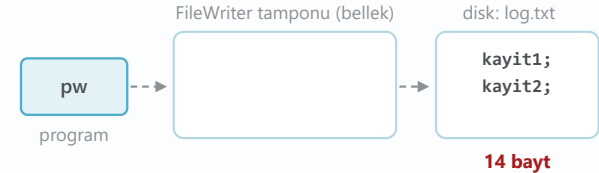
tampon = ""

ÇIKTI

```
0: 0
1: 0
2: 0
```

Tampon ve flush: Adım Adım

```
1 File f = new File("log.txt");
2 PrintWriter pw = new PrintWriter(new FileWriter(f));
3 System.out.println("0: " + f.length());
4 pw.print("kayit1;");
5 System.out.println("1: " + f.length());
6 pw.print("kayit2;");
7 System.out.println("2: " + f.length());
8 pw.flush();
9 ► System.out.println("3: " + f.length());
10 pw.print("kayit3;");
11 pw.close();
12 System.out.println("4: " + f.length());
13 pw.print("kayit4;"); // kapalı: sessizce yok sayılır
14 System.out.println("hata? " + pw.checkError());
```



Artık uzunluk 14.

Adım 7 / 11

DEĞİŞKENLER

`f.length()` = 14

ÇIKTI

```
0: 0
1: 0
2: 0
3: 14
```

Tampon ve flush: Adım Adım

```
1 File f = new File("log.txt");
2 PrintWriter pw = new PrintWriter(new FileWriter(f));
3 System.out.println("0: " + f.length());
4 pw.print("kayit1;");
5 System.out.println("1: " + f.length());
6 pw.print("kayit2;");
7 System.out.println("2: " + f.length());
8 pw.flush();
9 System.out.println("3: " + f.length());
10 pw.print("kayit3;");
11 pw.close();
12 System.out.println("4: " + f.length());
13 pw.print("kayit4;"); // kapalı: sessizce yok sayılır
14 System.out.println("hata? " + pw.checkError());
```



Üçüncü kayıt tampona girer.

Adım 8 / 11

DEĞİŞKENLER

tampon = "kayit3"

ÇIKTI

```
0: 0
1: 0
2: 0
3: 14
```

Tampon ve flush: Adım Adım

```
1 File f = new File("log.txt");
2 PrintWriter pw = new PrintWriter(new FileWriter(f));
3 System.out.println("0: " + f.length());
4 pw.print("kayit1;");
5 System.out.println("1: " + f.length());
6 pw.print("kayit2;");
7 System.out.println("2: " + f.length());
8 pw.flush();
9 System.out.println("3: " + f.length());
10 pw.print("kayit3;");
11 ► pw.close();
12 System.out.println("4: " + f.length());
13 pw.print("kayit4;"); // kapalı: sessizce yok sayılır
14 System.out.println("hata? " + pw.checkError());
```



`close()` önce `flush()` yapar, sonra dosyayı kapatır.

Adım 9 / 11

DEĞİŞKENLER

`pw = kapalı`

ÇIKTI

```
0: 0
1: 0
2: 0
3: 14
```

Tampon ve flush: Adım Adım

```
1 File f = new File("log.txt");
2 PrintWriter pw = new PrintWriter(new FileWriter(f));
3 System.out.println("0: " + f.length());
4 pw.print("kayit1;");
5 System.out.println("1: " + f.length());
6 pw.print("kayit2;");
7 System.out.println("2: " + f.length());
8 pw.flush();
9 System.out.println("3: " + f.length());
10 pw.print("kayit3;");
11 pw.close();
12 System.out.println("4: " + f.length());
13 pw.print("kayit4;"); // kapalı: sessizce yok sayılır
14 System.out.println("hata? " + pw.checkError());
```



Dosya 21 bayt. `close` unutulsaydı son kayıt kaybolabilirdi.

Adım 10 / 11

DEĞİŞKENLER

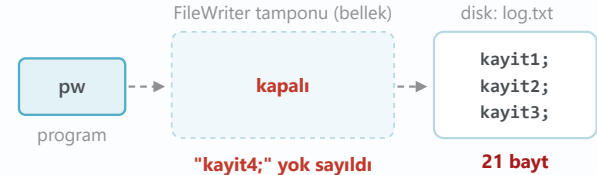
`f.length() = 21`

ÇIKTI

```
0: 0
1: 0
2: 0
3: 14
4: 21
```

Tampon ve flush: Adım Adım

```
1 File f = new File("log.txt");
2 PrintWriter pw = new PrintWriter(new FileWriter(f));
3 System.out.println("0: " + f.length());
4 pw.print("kayit1;");
5 System.out.println("1: " + f.length());
6 pw.print("kayit2;");
7 System.out.println("2: " + f.length());
8 pw.flush();
9 System.out.println("3: " + f.length());
10 pw.print("kayit3;");
11 pw.close();
12 System.out.println("4: " + f.length());
▶ pw.print("kayit4;"); // kapalı: sessizce yok sayılır
▶ System.out.println("hata? " + pw.checkError());
```



PrintWriter istisna fırlatmaz; kapalı akışa yazma **sessizce** başarısız olur. Hata yalnızca **checkError()** ile görülür.

Adım 11 / 11

DEĞİŞKENLER

```
pw.checkError() = true
```

ÇIKTI

```
0: 0
1: 0
2: 0
3: 14
4: 21
hata? true
```

BufferedInputStream ve BufferedReader

```
try {
    FileInputStream fis = new FileInputStream("dosya.txt");
    BufferedInputStream bis = new BufferedInputStream(fis);
    int veri;
    while ((veri = bis.read()) != -1) {
        System.out.print((char) veri);
    }
    bis.close();    // içteki fis'i de kapatır
} catch (IOException e) {
    System.err.println("Dosya hatası: " + e.getMessage());
}
```

ÇIKTI: BAYT BAYT

Java I/O
Tampon

```
try {
    FileReader fr = new FileReader("dosya.txt");
    BufferedReader br = new BufferedReader(fr);
    String satir;
    while ((satir = br.readLine()) != null) {
        System.out.println(satir);
    }
    br.close();    // içteki fr'yi de kapatır
} catch (IOException e) {
    System.err.println("Dosya hatası: " + e.getMessage());
}
```

ÇIKTI: SATIR SATIR

Java I/O
Tampon

Tamponlama Ne Kadar Hızlandırır?

```
public class TamponOlcum {
    static long oku(InputStream in) throws IOException {
        long bas = System.nanoTime();
        while (in.read() != -1) { }           // her seferinde 1 byte
        in.close();
        return (System.nanoTime() - bas) / 1_000_000;
    }
    public static void main(String[] args) throws IOException {
        File f = new File("buyuk.dat");
        try (FileOutputStream fos = new FileOutputStream(f)) {
            fos.write(new byte[2 * 1024 * 1024]); // 2 MB
        }
        long t1 = oku(new FileInputStream(f));
        long t2 = oku(new BufferedInputStream(new FileInputStream(f)));
        System.out.println("Tamponsuz: " + t1 + " ms");
        System.out.println("Tamponlu : " + t2 + " ms");
        f.delete();
    }
}
```

ÇIKTI (ÖRNEK ÇALIŞTIRMA)

Tamponsuz: 3151 ms
Tamponlu : 94 ms

- Aynı 2 MB dosya, iki kez bayt bayt okunuyor.
- Tamponsuz: 2 097 152
`read()` = o kadar **sistem çağırısı**.
- Tamponlu: sistem çağırısı 8 KB'ta bir, 256 kez. Fark **onlarca kat**.
- Süreler makineye göre değişir; oran benzer kalır.

DİĞER BAYT AKIŞLARI

java.io.FileInputStream

- `InputStream` 'dan türer; dosyadan doğrudan **bayt** okur. Düşük seviyeli: resim, ses gibi metin dışı veriler için.
- `read()` : tek bayt (0–255), dosya sonunda **-1**. `read(byte[] b)` : en fazla `b.length` bayt okur ve **okuduğu sayıyı** döndürür.
- Metin için zordur (ü gibi çok baytlı karakterler bölünebilir); `FileReader` gibi karakter tabanlı sınıflar daha uygundur.

```
public class ParcaOku {  
    public static void main(String[] args) throws IOException {  
        try (FileInputStream fis = new FileInputStream("dosya.txt")) {  
            byte[] dizi = new byte[4];  
            int adet;  
            while ((adet = fis.read(dizi)) != -1) {  
                System.out.println(adet + ": " + new String(dizi, 0, adet));  
            }  
        }  
    }  
}
```

ÇIKTI

```
4: Merh  
4: aba  
4: Java
```

dosya.txt: "Merhaba Java", 12 bayt, 4'erli parçalar. `new String(dizi, 0, adet)` yalnızca okunan kısmı çevirir.

DiĞER BAYT AKIŞLARI

java.io.ByteArrayInputStream

- `InputStream` 'dan türer; bellekteki bir **bayt dizisinden** okur. Fiziksel kaynağa erişmez, veri bellekte olduğu için **hızlıdır**.
- Yalnızca **okunur**; belleğe yazmak için eşi `ByteArrayOutputStream` var. Büyük veride bellek tüketimi sorun olabilir.

```
public class ByteDiziOku {  
    public static void main(String[] args) throws IOException {  
        byte[] dizi = {74, 97, 118, 97};           // "Java"  
        ByteArrayInputStream bais = new ByteArrayInputStream(dizi);  
        int b;  
        while ((b = bais.read()) != -1) {  
            System.out.print(b + "=" + (char) b + " ");  
        }  
        System.out.println();  
        // Belleğe yazma: ByteArrayOutputStream  
        ByteArrayOutputStream baos = new ByteArrayOutputStream();  
        baos.write(dizi);  
        baos.write('!');  
        byte[] sonuc = baos.toByteArray();  
        System.out.println(sonuc.length + " byte: " + baos);  
    }  
}
```

ÇIKTI

```
74=J 97=a 118=v 97=a  
5 byte: Java!
```

StringBufferInputStream: Kullanmayın

- `java.io.StringBufferInputStream` Java 1.1'den beri **kullanımdan kaldırılmıştır** (deprecated); kullanımı tavsiye edilmez.
- Karakterleri bayta **doğru çevirmez** (yalnızca alt 8 bit). Yerine `ByteArrayInputStream` + `getBytes(UTF_8)` veya `StringReader` .

```
public class EskiAkis {
    @SuppressWarnings("deprecation")
    public static void main(String[] args) throws IOException {
        InputStream eski = new StringBufferInputStream("ş€");
        int b;
        while ((b = eski.read()) != -1) {
            System.out.print(b + " ");
        }
        System.out.println();
        byte[] dogru = "ş€".getBytes("UTF-8");
        InputStream yeni = new ByteArrayInputStream(dogru);
        while ((b = yeni.read()) != -1) {
            System.out.print(b + " ");
        }
        System.out.println();
    }
}
```

ÇIKTI

```
95 172
197 159 226 130 172
```

ş (351) → 95 = `_`, € →
172. Doğrusu UTF-8: 197 159
226 130 172.

PipedInputStream ve PipedOutputStream

- `InputStream` / `OutputStream` sınıflarından türetilmiştir; iki **iş parçacığı** (thread) arasında güvenli veri iletişimi sağlar.
- Bir iş parçacığı `PipedOutputStream` ile veri **yazar**, diğeri `PipedInputStream` ile bu veriyi **okur**. İki uç `new PipedOutputStream(pis)` ile bağlanır.
- Okuma ve yazma senkronizedir: boru boşsa okuyucu **bekler**, doluysa yazıcı bekler.
- Yalnızca **tek yönlü** veri iletimini destekler; çift yön için iki boru gerekir.
- İki ucu aynı iş parçacığında kullanmak kilitlenmeye (deadlock) yol açabilir. İş parçacıkları Bölüm 9'da ayrıntılı işlenecek.



DİĞER BAYT AKIŞLARI

Boru Örneği

```
PipedInputStream pis = new PipedInputStream();
PipedOutputStream pos = new PipedOutputStream(pis);
```

```
// Okuyucu iş parçası
Thread okuyucu = new Thread(() -> {
    try {
        int b;
        while ((b = pis.read()) != -1)
            System.out.print((char) b);
        System.out.println();
        pis.close();
    } catch (IOException e) {
        e.printStackTrace();
    }
});
```

```
// Yazıcı iş parçası
Thread yazici = new Thread(() -> {
    try {
        pos.write("Merhaba, boru!".getBytes());
        pos.close();
    } catch (IOException e) {
        e.printStackTrace();
    }
});
okuyucu.start();
yazici.start();
yazici.join();
okuyucu.join();
```

ÇIKTI

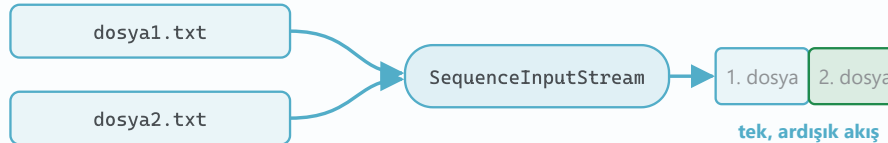
Merhaba, boru!

- `start()` iş parçasını başlatır; `join()` bitmesini bekler.
- Yazıcı `pos.close()` deyince okuyucunun `read()` 'i -1 döner, döngü biter.

DiĞER BAYT AKIŞLARI

java.io.SequenceInputStream

- Birden fazla `InputStream` 'i birleştirerek **ardışık** tek bir giriş akışı oluşturur: birinci akış bitince (-1) ikinciye geçer.
- Farklı kaynaklardan gelen veriyi tek akışta birleştirir. Bir akışta hata olursa `IOException` fırlatılır, diğerine geçilmez.



```
try {
    FileInputStream is1 = new FileInputStream("dosya1.txt");
    FileInputStream is2 = new FileInputStream("dosya2.txt");
    SequenceInputStream sis = new SequenceInputStream(is1, is2);
    int okunanByte;
    while ((okunanByte = sis.read()) != -1)
        System.out.print((char) okunanByte);
    sis.close(); // is1 ve is2'yi de kapatır
} catch (IOException e) {
    System.err.println("Okuma hatası: " + e.getMessage());
}
```

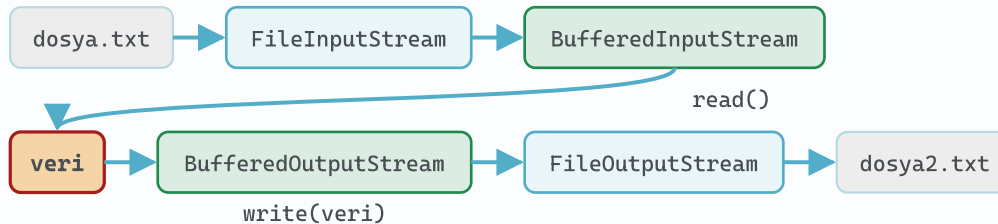
ÇIKTI

Birinci dosya.
İkinci dosya.

`sis.close()` içteki
akışları da kapatır.

Bayt Bayt Dosya Kopyalama

- Kaynak dosyadan okunan her baytı hedef dosyaya yazar: **her tür** dosyayı (metin, resim, `.exe`) bozmadan kopyalar.
- `BufferedInputStream` ve `BufferedOutputStream` giriş ve çıkışı hızlandırır: önbellek kullanarak okuma ve yazmayı toplu yapar.
- `bis.read()` : bir bayt okur, dosyanın sonunda `-1` döner. `bos.write(veri)` : okunan baytı çıkış dosyasına yazar.
- Okunan değer `int` bir değişkende tutulmalıdır: `-1` ile 255 bayt değerini ayırt etmek için.



Kopyalama döngüsü tek satırdır: `while ((veri = bis.read()) != -1) bos.write(veri);`

Bayt Bayt Kopyalama: Adım Adım

```
1 import java.io.*;
2
3 public class BaytKopya {
4     public static void main(String[] args) throws IOException {
5         ▶ File dosya = new File("dosya.txt");
6         ▶ File dosya2 = new File("dosya2.txt");
7         FileInputStream fis = new FileInputStream(dosya);
8         FileOutputStream fos = new FileOutputStream(dosya2);
9         BufferedInputStream bis = new BufferedInputStream(fis);
10        BufferedOutputStream bos = new BufferedOutputStream(fos);
11        int veri;
12        // Bir byte oku. Dosyanın sonunda -1 dönecektir.
13        while ((veri = bis.read()) != -1)
14            bos.write(veri);    // Okunan byte'ı çıkışa yaz
15        bos.close();
16        bis.close();
17        System.out.println(dosya2.length() + " byte kopyalandı");
18    }
19 }
```

dosya.txt (disk)

J a v a

bis tamponu (bellek)

□ □ □ □ □ ...

veri = ?

bos tamponu (bellek)

□ □ □ □ □ ...

dosya2.txt (disk)

□ □ □ □

İki **File** nesnesi yalnızca yolları tutar.

Adım 1 / 15

ÇIKTI

DEĞİŞKENLER

dosya = "dosya.txt"

dosya2 = "dosya2.txt"

```

1 import java.io.*;
2
3 public class BaytKopya {
4     public static void main(String[] args) throws IOException {
5         File dosya = new File("dosya.txt");
6         File dosya2 = new File("dosya2.txt");
7         FileInputStream fis = new FileInputStream(dosya);
8         FileOutputStream fos = new FileOutputStream(dosya2);
9         BufferedInputStream bis = new BufferedInputStream(fis);
10        BufferedOutputStream bos = new BufferedOutputStream(fos);
11        int veri;
12        // Bir byte oku. Dosyanın sonunda -1 dönecektir.
13        while ((veri = bis.read()) != -1)
14            bos.write(veri);    // Okunan byte'ı çıkışa yaz
15        bos.close();
16        bis.close();
17        System.out.println(dosya2.length() + " byte kopyalandı");
18    }
19 }

```

J a v a

...

bos tamponu (bellek)

						...
--	--	--	--	--	--	-----

0 bayt

`FileOutputStream` hedef dosyayı **oluşturur** (varsa içeriğini siler): 0 bayt.

Adım 2 / 15

ÇIKTI

DEĞİŞKENLER

```
( dosya = "dosya.txt"
```

```
dosya2 = "dosya2.txt"
```

Bayt Bayt Kopyalama: Adım Adım

```
1 import java.io.*;
2
3 public class BaytKopya {
4     public static void main(String[] args) throws IOException {
5         File dosya = new File("dosya.txt");
6         File dosya2 = new File("dosya2.txt");
7         FileInputStream fis = new FileInputStream(dosya);
8         FileOutputStream fos = new FileOutputStream(dosya2);
9         ▶ BufferedInputStream bis = new BufferedInputStream(fis);
10        ▶ BufferedOutputStream bos = new BufferedOutputStream(fos);
11        int veri;
12        // Bir byte oku. Dosyanın sonunda -1 dönecektir.
13        while ((veri = bis.read()) != -1)
14            bos.write(veri);    // Okunan byte'ı çıkışa yaz
15        bos.close();
16        bis.close();
17        System.out.println(dosya2.length() + " byte kopyalandı");
18    }
19 }
```

dosya.txt (disk)

J	a	v	a
---	---	---	---

bis tamponu (bellek)

						...
--	--	--	--	--	--	-----

veri = ?

bos tamponu (bellek)

						...
--	--	--	--	--	--	-----

dosya2.txt (disk)

--	--	--	--

 0 bayt

İki akış tamponlu akışlarla sarmalanır;
tamponlar (8192 bayt) boş.

Adım 3 / 15

ÇIKTI

DEĞİŞKENLER

veri = ?

Bayt Bayt Kopyalama: Adım Adım

```
1 import java.io.*;
2
3 public class BaytKopya {
4     public static void main(String[] args) throws IOException {
5         File dosya = new File("dosya.txt");
6         File dosya2 = new File("dosya2.txt");
7         FileInputStream fis = new FileInputStream(dosya);
8         FileOutputStream fos = new FileOutputStream(dosya2);
9         BufferedInputStream bis = new BufferedInputStream(fis);
10        BufferedOutputStream bos = new BufferedOutputStream(fos);
11        int veri;
12        // Bir byte oku. Dosyanın sonunda -1 dönecektir.
13        ► while ((veri = bis.read()) != -1)
14            bos.write(veri);    // Okunan byte'ı çıkışa yaz
15        bos.close();
16        bis.close();
17        System.out.println(dosya2.length() + " byte kopyalandı");
18    }
19 }
```

dosya.txt (disk)

J a v a

bis tamponu (bellek)

J a v a ...

veri = 74 'J'

bos tamponu (bellek)

... ..

dosya2.txt (disk)

... 0 bayt

İlk `read()` : tampon boş olduğu için diskten **tek seferde** 4 baytın tamamı tampona alınır; ilk bayt 74 ('J') döner.

Adım 4 / 15

ÇIKTI

DEĞİŞKENLER

veri = 74

Bayt Bayt Kopyalama: Adım Adım

```
1 import java.io.*;
2
3 public class BaytKopya {
4     public static void main(String[] args) throws IOException {
5         File dosya = new File("dosya.txt");
6         File dosya2 = new File("dosya2.txt");
7         FileInputStream fis = new FileInputStream(dosya);
8         FileOutputStream fos = new FileOutputStream(dosya2);
9         BufferedInputStream bis = new BufferedInputStream(fis);
10        BufferedOutputStream bos = new BufferedOutputStream(fos);
11        int veri;
12        // Bir byte oku. Dosyanın sonunda -1 dönecektir.
13        while ((veri = bis.read()) != -1)
14            ▶ bos.write(veri); // Okunan byte'ı çıkışa yaz
15        bos.close();
16        bis.close();
17        System.out.println(dosya2.length() + " byte kopyalandı");
18    }
19 }
```

dosya.txt (disk)



bis tamponu (bellek)



veri = 74 'J'

bos tamponu (bellek)



dosya2.txt (disk)



write baytı çıkış **tamponuna** koyar;
dosya2.txt hâlâ 0 bayt.

Adım 5 / 15

ÇIKTI



DEĞİŞKENLER

veri = 74

Bayt Bayt Kopyalama: Adım Adım

```
1 import java.io.*;
2
3 public class BaytKopya {
4     public static void main(String[] args) throws IOException {
5         File dosya = new File("dosya.txt");
6         File dosya2 = new File("dosya2.txt");
7         FileInputStream fis = new FileInputStream(dosya);
8         FileOutputStream fos = new FileOutputStream(dosya2);
9         BufferedInputStream bis = new BufferedInputStream(fis);
10        BufferedOutputStream bos = new BufferedOutputStream(fos);
11        int veri;
12        // Bir byte oku. Dosyanın sonunda -1 dönecektir.
13        ► while ((veri = bis.read()) != -1)
14            bos.write(veri); // Okunan byte'ı çıkışa yaz
15        bos.close();
16        bis.close();
17        System.out.println(dosya2.length() + " byte kopyalandı");
18    }
19 }
```

dosya.txt (disk)



bis tamponu (bellek)



veri = 97 'a'

bos tamponu (bellek)



dosya2.txt (disk)



Bu kez diske gidilmez: bayt tampondan gelir (97 = 'a').

Adım 6 / 15

ÇIKTI



DEĞİŞKENLER

veri = 97

Bayt Bayt Kopyalama: Adım Adım

```
1 import java.io.*;
2
3 public class BaytKopya {
4     public static void main(String[] args) throws IOException {
5         File dosya = new File("dosya.txt");
6         File dosya2 = new File("dosya2.txt");
7         FileInputStream fis = new FileInputStream(dosya);
8         FileOutputStream fos = new FileOutputStream(dosya2);
9         BufferedInputStream bis = new BufferedInputStream(fis);
10        BufferedOutputStream bos = new BufferedOutputStream(fos);
11        int veri;
12        // Bir byte oku. Dosyanın sonunda -1 dönecektir.
13        while ((veri = bis.read()) != -1)
14            bos.write(veri); // Okunan byte'ı çıkışa yaz
15        bos.close();
16        bis.close();
17        System.out.println(dosya2.length() + " byte kopyalandı");
18    }
19 }
```

dosya.txt (disk)



bis tamponu (bellek)



veri = 97 'a'

bos tamponu (bellek)



dosya2.txt (disk)



Çıkış tamponunda artık 2 bayt var.

Adım 7 / 15

ÇIKTI



DEĞİŞKENLER

veri = 97

Bayt Bayt Kopyalama: Adım Adım

```
1 import java.io.*;
2
3 public class BaytKopya {
4     public static void main(String[] args) throws IOException {
5         File dosya = new File("dosya.txt");
6         File dosya2 = new File("dosya2.txt");
7         FileInputStream fis = new FileInputStream(dosya);
8         FileOutputStream fos = new FileOutputStream(dosya2);
9         BufferedInputStream bis = new BufferedInputStream(fis);
10        BufferedOutputStream bos = new BufferedOutputStream(fos);
11        int veri;
12        // Bir byte oku. Dosyanın sonunda -1 dönecektir.
13        ► while ((veri = bis.read()) != -1)
14            bos.write(veri); // Okunan byte'ı çıkışa yaz
15        bos.close();
16        bis.close();
17        System.out.println(dosya2.length() + " byte kopyalandı");
18    }
19 }
```

dosya.txt (disk)



bis tamponu (bellek)



veri = 118 'v'

bos tamponu (bellek)



dosya2.txt (disk)



Bu kez diske gidilmez: bayt tampondan gelir (118 = 'v').

Adım 8 / 15

ÇIKTI



DEĞİŞKENLER

veri = 118

Bayt Bayt Kopyalama: Adım Adım

```
1 import java.io.*;
2
3 public class BaytKopya {
4     public static void main(String[] args) throws IOException {
5         File dosya = new File("dosya.txt");
6         File dosya2 = new File("dosya2.txt");
7         FileInputStream fis = new FileInputStream(dosya);
8         FileOutputStream fos = new FileOutputStream(dosya2);
9         BufferedInputStream bis = new BufferedInputStream(fis);
10        BufferedOutputStream bos = new BufferedOutputStream(fos);
11        int veri;
12        // Bir byte oku. Dosyanın sonunda -1 dönecektir.
13        while ((veri = bis.read()) != -1)
14            bos.write(veri); // Okunan byte'ı çıkışa yaz
15        bos.close();
16        bis.close();
17        System.out.println(dosya2.length() + " byte kopyalandı");
18    }
19 }
```

dosya.txt (disk)



bis tamponu (bellek)



veri = 118 'v'

bos tamponu (bellek)



dosya2.txt (disk)



Çıkış tamponunda artık 3 bayt var.

Adım 9 / 15

ÇIKTI



DEĞİŞKENLER

veri = 118

Bayt Bayt Kopyalama: Adım Adım

```
1 import java.io.*;
2
3 public class BaytKopya {
4     public static void main(String[] args) throws IOException {
5         File dosya = new File("dosya.txt");
6         File dosya2 = new File("dosya2.txt");
7         FileInputStream fis = new FileInputStream(dosya);
8         FileOutputStream fos = new FileOutputStream(dosya2);
9         BufferedInputStream bis = new BufferedInputStream(fis);
10        BufferedOutputStream bos = new BufferedOutputStream(fos);
11        int veri;
12        // Bir byte oku. Dosyanın sonunda -1 dönecektir.
13        ► while ((veri = bis.read()) != -1)
14            bos.write(veri);    // Okunan byte'ı çıkışa yaz
15        bos.close();
16        bis.close();
17        System.out.println(dosya2.length() + " byte kopyalandı");
18    }
19 }
```

dosya.txt (disk)



bis tamponu (bellek)



veri = 97 'a'

bos tamponu (bellek)



dosya2.txt (disk)



Bu kez diske gidilmez: bayt tampondan gelir (97 = 'a').

Adım 10 / 15

ÇIKTI



DEĞİŞKENLER

veri = 97

Bayt Bayt Kopyalama: Adım Adım

```
1 import java.io.*;
2
3 public class BaytKopya {
4     public static void main(String[] args) throws IOException {
5         File dosya = new File("dosya.txt");
6         File dosya2 = new File("dosya2.txt");
7         FileInputStream fis = new FileInputStream(dosya);
8         FileOutputStream fos = new FileOutputStream(dosya2);
9         BufferedInputStream bis = new BufferedInputStream(fis);
10        BufferedOutputStream bos = new BufferedOutputStream(fos);
11        int veri;
12        // Bir byte oku. Dosyanın sonunda -1 dönecektir.
13        while ((veri = bis.read()) != -1)
14            bos.write(veri); // Okunan byte'ı çıkışa yaz
15        bos.close();
16        bis.close();
17        System.out.println(dosya2.length() + " byte kopyalandı");
18    }
19 }
```

dosya.txt (disk)



bis tamponu (bellek)



veri = 97 'a'

bos tamponu (bellek)



dosya2.txt (disk)



Çıkış tamponunda artık 4 bayt var.

Adım 11 / 15

ÇIKTI



DEĞİŞKENLER

veri = 97

Bayt Bayt Kopyalama: Adım Adım

```
1 import java.io.*;
2
3 public class BaytKopya {
4     public static void main(String[] args) throws IOException {
5         File dosya = new File("dosya.txt");
6         File dosya2 = new File("dosya2.txt");
7         FileInputStream fis = new FileInputStream(dosya);
8         FileOutputStream fos = new FileOutputStream(dosya2);
9         BufferedInputStream bis = new BufferedInputStream(fis);
10        BufferedOutputStream bos = new BufferedOutputStream(fos);
11        int veri;
12        // Bir byte oku. Dosyanın sonunda -1 dönecektir.
13        ► while ((veri = bis.read()) != -1)
14            bos.write(veri); // Okunan byte'ı çıkışa yaz
15        bos.close();
16        bis.close();
17        System.out.println(dosya2.length() + " byte kopyalandı");
18    }
19 }
```

dosya.txt (disk)



bis tamponu (bellek)



veri = -1

bos tamponu (bellek)



dosya2.txt (disk)



Okunacak byte kalmadı: **read()** **-1**
döndürür, döngü biter.

Adım 12 / 15

ÇIKTI



DEĞİŞKENLER

veri = -1

Bayt Bayt Kopyalama: Adım Adım

```
1 import java.io.*;
2
3 public class BaytKopya {
4     public static void main(String[] args) throws IOException {
5         File dosya = new File("dosya.txt");
6         File dosya2 = new File("dosya2.txt");
7         FileInputStream fis = new FileInputStream(dosya);
8         FileOutputStream fos = new FileOutputStream(dosya2);
9         BufferedInputStream bis = new BufferedInputStream(fis);
10        BufferedOutputStream bos = new BufferedOutputStream(fos);
11        int veri;
12        // Bir byte oku. Dosyanın sonunda -1 dönecektir.
13        while ((veri = bis.read()) != -1)
14            bos.write(veri);    // Okunan byte'ı çıkışa yaz
15        ▶ bos.close();
16        bis.close();
17        System.out.println(dosya2.length() + " byte kopyalandı");
18    }
19 }
```

dosya.txt (disk)

J a v a

bis tamponu (bellek)

J a v a

veri = -1

bos tamponu (bellek)

dosya2.txt (disk)

J a v a

close() önce **flush()** yapar: 4 bayt **tek seferde** diske yazılır, sonra **fos** kapanır.

Adım 13 / 15

ÇIKTI

DEĞİŞKENLER

veri = -1

Bayt Bayt Kopyalama: Adım Adım

```
1 import java.io.*;
2
3 public class BaytKopya {
4     public static void main(String[] args) throws IOException {
5         File dosya = new File("dosya.txt");
6         File dosya2 = new File("dosya2.txt");
7         FileInputStream fis = new FileInputStream(dosya);
8         FileOutputStream fos = new FileOutputStream(dosya2);
9         BufferedInputStream bis = new BufferedInputStream(fis);
10        BufferedOutputStream bos = new BufferedOutputStream(fos);
11        int veri;
12        // Bir byte oku. Dosyanın sonunda -1 dönecektir.
13        while ((veri = bis.read()) != -1)
14            bos.write(veri);    // Okunan byte'ı çıkışa yaz
15        bos.close();
16        ▶ bis.close();
17        System.out.println(dosya2.length() + " byte kopyalandı");
18    }
19 }
```

dosya.txt (disk)

J a v a

bis tamponu (bellek)

J a v a

veri = -1

bos tamponu (bellek)

dosya2.txt (disk)

J a v a

Giriş akışları da kapatılır.

Adım 14 / 15

ÇIKTI

DEĞİŞKENLER

veri = -1

Bayt Bayt Kopyalama: Adım Adım

```
1 import java.io.*;
2
3 public class BaytKopya {
4     public static void main(String[] args) throws IOException {
5         File dosya = new File("dosya.txt");
6         File dosya2 = new File("dosya2.txt");
7         FileInputStream fis = new FileInputStream(dosya);
8         FileOutputStream fos = new FileOutputStream(dosya2);
9         BufferedInputStream bis = new BufferedInputStream(fis);
10        BufferedOutputStream bos = new BufferedOutputStream(fos);
11        int veri;
12        // Bir byte oku. Dosyanın sonunda -1 dönecektir.
13        while ((veri = bis.read()) != -1)
14            bos.write(veri);    // Okunan byte'ı çıkışa yaz
15        bos.close();
16        bis.close();
17        ▶ System.out.println(dosya2.length() + " byte kopyalandı");
18    }
19 }
```

dosya.txt (disk)



bis tamponu (bellek)



veri = -1

bos tamponu (bellek)



dosya2.txt (disk)



Diskte 1 okuma ve 1 yazma oldu; tamponsuz olsaydı 5 okuma ve 4 yazma olurdu.

Adım 15 / 15

ÇIKTI

4 byte kopyalandı

DEĞİŞKENLER

veri = -1

Blok Blok (Parça Parça) Kopyalama

```
public class ParcaKopya {  
    public static void main(String[] args) throws IOException {  
        // Veri, 16K'lık parçalar hâlinde okunacak  
        byte[] parca = new byte[1024 * 16];  
        File dosya = new File("dosya.txt");  
        File dosya2 = new File("dosya2.txt");  
        FileInputStream fis = new FileInputStream(dosya);  
        FileOutputStream fos = new FileOutputStream(dosya2);  
        BufferedInputStream bis = new BufferedInputStream(fis);  
        BufferedOutputStream bos = new BufferedOutputStream(fos);  
        int boyut;  
        // 16K'ya kadar oku. Dosyanın sonunda -1 dönecektir.  
        while ((boyut = bis.read(parca)) > -1)  
            bos.write(parca, 0, boyut); // Okunan parçayı çıkışa yaz  
        bos.close();  
        bis.close();  
        System.out.println(dosya2.length() + " byte kopyalandı");  
    }  
}
```

ÇIKTI (40 000 BAYT)

40000 byte kopyalandı

- Toplu okuma performansı **artırır**.
- `read(parca)` : okunan bayt **sayısını** döndürür.
- `write(parca, 0, boyut)` : yalnızca ilk `boyut` baytı yazar.

`new byte[1024 * 16]` : veri 16 KB'lık parçalar hâlinde işlenir; dosya ne kadar büyük olursa olsun bellek kullanımı sabit kalır.

Blok Blok Kopyalama: Adım Adım

```
1 public class ParcaIzle {
2     public static void main(String[] args) throws IOException {
3         ▶ byte[] parca = new byte[16];
4         try (InputStream in = new FileInputStream("dosya.txt");
5             OutputStream out = new FileOutputStream("dosya2.txt")) {
6             int boyut;
7             while ((boyut = in.read(parca)) > -1) {
8                 out.write(parca, 0, boyut);
9                 System.out.println("boyut = " + boyut);
10            }
11        }
12        long n = new File("dosya2.txt").length();
13        System.out.println("Bitti: " + n + " byte");
14    }
15 }
```

dosya.txt: 39 bayt, 16'lık bloklar

0–15

16–31

32–38

parca dizisi (16 öge)



boyut = ?

16 baytlık bir dizi (gerçek programlarda 8–16 KB).

Adım 1 / 13

ÇIKTI



DEĞİŞKENLER

parca = byte[16]

Blok Blok Kopyalama: Adım Adım

```
1 public class ParcaIzle {
2     public static void main(String[] args) throws IOException {
3         byte[] parca = new byte[16];
4         ▶ try (InputStream in = new FileInputStream("dosya.txt");
5             ▶     OutputStream out = new FileOutputStream("dosya2.txt")) {
6             int boyut;
7             while ((boyut = in.read(parca)) > -1) {
8                 out.write(parca, 0, boyut);
9                 System.out.println("boyut = " + boyut);
10            }
11        }
12        long n = new File("dosya2.txt").length();
13        System.out.println("Bitti: " + n + " byte");
14    }
15 }
```

dosya.txt: 39 bayt, 16'lık bloklar

0–15

16–31

32–38

parca dizisi (16 öge)

--	--	--	--	--	--	--	--	--	--	--	--	--	--	--

boyut = ?

İki kaynak birlikte açılır; **try**
bitince ikisi de kapanacak.

Adım 2 / 13

ÇIKTI

DEĞİŞKENLER

parca = byte[16]

Blok Blok Kopyalama: Adım Adım

```
1 public class ParcaIzle {
2     public static void main(String[] args) throws IOException {
3         byte[] parca = new byte[16];
4         try (InputStream in = new FileInputStream("dosya.txt");
5             OutputStream out = new FileOutputStream("dosya2.txt")) {
6             int boyut;
7             while ((boyut = in.read(parca)) > -1) {
8                 out.write(parca, 0, boyut);
9                 System.out.println("boyut = " + boyut);
10            }
11        }
12        long n = new File("dosya2.txt").length();
13        System.out.println("Bitti: " + n + " byte");
14    }
15 }
```

dosya.txt: 39 bayt, 16'lık bloklar



parca dizisi (16 öge)



boyut = 16

read(parca) diziyi doldurur, 16 döner.

Adım 3 / 13

ÇIKTI



DEĞİŞKENLER

boyut = 16

Blok Blok Kopyalama: Adım Adım

```
1 public class ParcaIzle {
2     public static void main(String[] args) throws IOException {
3         byte[] parca = new byte[16];
4         try (InputStream in = new FileInputStream("dosya.txt");
5             OutputStream out = new FileOutputStream("dosya2.txt")) {
6             int boyut;
7             while ((boyut = in.read(parca)) > -1) {
8                 ► out.write(parca, 0, boyut);
9                 System.out.println("boyut = " + boyut);
10            }
11        }
12        long n = new File("dosya2.txt").length();
13        System.out.println("Bitti: " + n + " byte");
14    }
15 }
```

dosya.txt: 39 bayt, 16'lık bloklar



parca dizisi (16 öge)



boyut = 16

`write(parca, 0, 16)` : yalnızca okunan 16 bayt yazılır.

Adım 4 / 13

ÇIKTI



DEĞİŞKENLER

boyut = 16

Blok Blok Kopyalama: Adım Adım

```
1 public class ParcaIzle {
2     public static void main(String[] args) throws IOException {
3         byte[] parca = new byte[16];
4         try (InputStream in = new FileInputStream("dosya.txt");
5             OutputStream out = new FileOutputStream("dosya2.txt")) {
6             int boyut;
7             while ((boyut = in.read(parca)) > -1) {
8                 out.write(parca, 0, boyut);
9                 System.out.println("boyut = " + boyut);
10            }
11        }
12        long n = new File("dosya2.txt").length();
13        System.out.println("Bitti: " + n + " byte");
14    }
15 }
```

dosya.txt: 39 bayt, 16'lık bloklar



parca dizisi (16 öge)



boyut = 16

Okunan miktar yazdırılır.

Adım 5 / 13

ÇIKTI

boyut = 16

DEĞİŞKENLER

boyut = 16

Blok Blok Kopyalama: Adım Adım

```
1 public class ParcaIzle {
2     public static void main(String[] args) throws IOException {
3         byte[] parca = new byte[16];
4         try (InputStream in = new FileInputStream("dosya.txt");
5             OutputStream out = new FileOutputStream("dosya2.txt")) {
6             int boyut;
7             while ((boyut = in.read(parca)) > -1) {
8                 out.write(parca, 0, boyut);
9                 System.out.println("boyut = " + boyut);
10            }
11        }
12        long n = new File("dosya2.txt").length();
13        System.out.println("Bitti: " + n + " byte");
14    }
15 }
```

dosya.txt: 39 bayt, 16'lık bloklar

0-15

16-31

32-38

parca dizisi (16 öge)

k i s l a r - J a v a D e

boyut = 16

Sonraki 16 bayt aynı dizinin **üzerine** yazılır.

Adım 6 / 13

ÇIKTI

boyut = 16

DEĞİŞKENLER

boyut = 16

Blok Blok Kopyalama: Adım Adım

```
1 public class ParcaIzle {
2     public static void main(String[] args) throws IOException {
3         byte[] parca = new byte[16];
4         try (InputStream in = new FileInputStream("dosya.txt");
5             OutputStream out = new FileOutputStream("dosya2.txt")) {
6             int boyut;
7             while ((boyut = in.read(parca)) > -1) {
8                 ► out.write(parca, 0, boyut);
9                 System.out.println("boyut = " + boyut);
10            }
11        }
12        long n = new File("dosya2.txt").length();
13        System.out.println("Bitti: " + n + " byte");
14    }
15 }
```

dosya.txt: 39 bayt, 16'lık bloklar

0-15

16-31

32-38

parca dizisi (16 öge)

k i s l a r - J a v a D e

boyut = 16

`write(parca, 0, 16)` : yalnızca okunan 16 bayt yazılır.

Adım 7 / 13

ÇIKTI

boyut = 16

DEĞİŞKENLER

boyut = 16

Blok Blok Kopyalama: Adım Adım

```
1 public class ParcaIzle {
2     public static void main(String[] args) throws IOException {
3         byte[] parca = new byte[16];
4         try (InputStream in = new FileInputStream("dosya.txt");
5             OutputStream out = new FileOutputStream("dosya2.txt")) {
6             int boyut;
7             while ((boyut = in.read(parca)) > -1) {
8                 out.write(parca, 0, boyut);
9                 System.out.println("boyut = " + boyut);
10            }
11        }
12        long n = new File("dosya2.txt").length();
13        System.out.println("Bitti: " + n + " byte");
14    }
15 }
```

dosya.txt: 39 bayt, 16'lık bloklar

0-15

16-31

32-38

parca dizisi (16 öge)

k i s l a r - J a v a D e

boyut = 16

Okunan miktar yazdırılır.

Adım 8 / 13

ÇIKTI

boyut = 16

boyut = 16

DEĞİŞKENLER

boyut = 16

Blok Blok Kopyalama: Adım Adım

```
1 public class ParcaIzle {
2     public static void main(String[] args) throws IOException {
3         byte[] parca = new byte[16];
4         try (InputStream in = new FileInputStream("dosya.txt");
5             OutputStream out = new FileOutputStream("dosya2.txt")) {
6             int boyut;
7             while ((boyut = in.read(parca)) > -1) {
8                 out.write(parca, 0, boyut);
9                 System.out.println("boyut = " + boyut);
10            }
11        }
12        long n = new File("dosya2.txt").length();
13        System.out.println("Bitti: " + n + " byte");
14    }
15 }
```

dosya.txt: 39 bayt, 16'lık bloklar



parca dizisi (16 öge)



boyut = 7

Yalnızca 7 bayt kaldı: dizinin ilk 7 ögesi yenilenir, kalan 9'u **önceki bloktan** kalma!

Adım 9 / 13

ÇIKTI

```
boyut = 16
boyut = 16
```

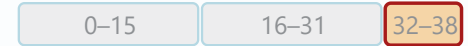
DEĞİŞKENLER

boyut = 7

Blok Blok Kopyalama: Adım Adım

```
1 public class ParcaIzle {
2     public static void main(String[] args) throws IOException {
3         byte[] parca = new byte[16];
4         try (InputStream in = new FileInputStream("dosya.txt");
5             OutputStream out = new FileOutputStream("dosya2.txt")) {
6             int boyut;
7             while ((boyut = in.read(parca)) > -1) {
8                 ► out.write(parca, 0, boyut);
9                 System.out.println("boyut = " + boyut);
10            }
11        }
12        long n = new File("dosya2.txt").length();
13        System.out.println("Bitti: " + n + " byte");
14    }
15 }
```

dosya.txt: 39 bayt, 16'lık bloklar



parca dizisi (16 öge)



boyut = 7

`write(parca, 0, 7)` : yalnızca okunan 7 bayt yazılır.

`write(parca)` deseydik eski 9 bayt da yazılır, kopya 48 bayt olurdu.

Adım 10 / 13

ÇIKTI

```
boyut = 16
boyut = 16
```

DEĞİŞKENLER

```
boyut = 7
```

Blok Blok Kopyalama: Adım Adım

```
1 public class ParcaIzle {
2     public static void main(String[] args) throws IOException {
3         byte[] parca = new byte[16];
4         try (InputStream in = new FileInputStream("dosya.txt");
5             OutputStream out = new FileOutputStream("dosya2.txt")) {
6             int boyut;
7             while ((boyut = in.read(parca)) > -1) {
8                 out.write(parca, 0, boyut);
9                 System.out.println("boyut = " + boyut);
10            }
11        }
12        long n = new File("dosya2.txt").length();
13        System.out.println("Bitti: " + n + " byte");
14    }
15 }
```

dosya.txt: 39 bayt, 16'lık bloklar



parca dizisi (16 öge)



boyut = 7

Okunan miktar yazdırılır.

Adım 11 / 13

ÇIKTI

```
boyut = 16
boyut = 16
boyut = 7
```

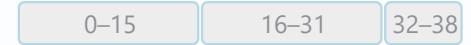
DEĞİŞKENLER

boyut = 7

Blok Blok Kopyalama: Adım Adım

```
1 public class ParcaIzle {
2     public static void main(String[] args) throws IOException {
3         byte[] parca = new byte[16];
4         try (InputStream in = new FileInputStream("dosya.txt");
5             OutputStream out = new FileOutputStream("dosya2.txt")) {
6             int boyut;
7             while ((boyut = in.read(parca)) > -1) {
8                 out.write(parca, 0, boyut);
9                 System.out.println("boyut = " + boyut);
10            }
11        }
12        long n = new File("dosya2.txt").length();
13        System.out.println("Bitti: " + n + " byte");
14    }
15 }
```

dosya.txt: 39 bayt, 16'lık bloklar



parca dizisi (16 öge)



boyut = -1

Dosya sonu: -1 döner, > -1

yanlış, döngü biter.

Adım 12 / 13

ÇIKTI

```
boyut = 16
boyut = 16
boyut = 7
```

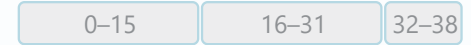
DEĞİŞKENLER

boyut = -1

Blok Blok Kopyalama: Adım Adım

```
1 public class ParcaIzle {
2     public static void main(String[] args) throws IOException {
3         byte[] parca = new byte[16];
4         try (InputStream in = new FileInputStream("dosya.txt");
5             OutputStream out = new FileOutputStream("dosya2.txt")) {
6             int boyut;
7             while ((boyut = in.read(parca)) > -1) {
8                 out.write(parca, 0, boyut);
9                 System.out.println("boyut = " + boyut);
10            }
11        }
12        long n = new File("dosya2.txt").length();
13        System.out.println("Bitti: " + n + " byte");
14    }
15 }
```

dosya.txt: 39 bayt, 16'lık bloklar



parca dizisi (16 öğe)



boyut = -1

Kaynaklar kapandı; kopya kaynakla aynı boyutta: 16 + 16 + 7 = 39 bayt.

Adım 13 / 13

ÇIKTI

```
boyut = 16
boyut = 16
boyut = 7
Bitti: 39 byte
```

DEĞİŞKENLER

boyut = -1

n = 39

Satır Satır Dosya Kopyalama

```
public class SatirKopya {  
    public static void main(String[] args) throws IOException {  
        File dosya = new File("dosya.txt");  
        File dosya2 = new File("dosya2.txt");  
        FileReader fr = new FileReader(dosya);  
        BufferedReader br = new BufferedReader(fr);  
        FileWriter fw = new FileWriter(dosya2);  
        BufferedWriter bw = new BufferedWriter(fw);  
        PrintWriter pw = new PrintWriter(bw);  
        String satir;  
        // dosya sonuna kadar bir satır oku  
        while ((satir = br.readLine()) != null)  
            pw.println(satir);    // satırı yaz  
        pw.close();  
        br.close();  
        System.out.println("Kopyalandı: " + dosya2.getName());  
    }  
}
```

ÇIKTI

Kopyalandı: dosya2.txt

- Karakter akışlarıyla çalışır: yalnızca **metin** dosyaları için.
- `readLine()` satır sonunu atar; `println` sistemin satır sonunu ekler (Windows: `\r\n`, Linux: `\n`). Kopya bayt bayt aynı olmayabilir.
- `pw.close()` zincirin tamamını (`bw` , `fw`) kapatır.

Web Sayfası İndirme

```
public class WebIndir {  
    public static void main(String[] args) throws IOException {  
        URL url = URI.create("https://sercankulcu.github.io/").toURL();  
        BufferedInputStream bis = new BufferedInputStream(url.openStream());  
        FileOutputStream fos = new FileOutputStream(new File("kopya.html"));  
        BufferedOutputStream bos = new BufferedOutputStream(fos);  
        for (int c = bis.read(); c != -1; c = bis.read()) {  
            bos.write(c);  
        }  
        bis.close();  
        bos.close();  
        System.out.println("kopya.html: " + new File("kopya.html").length() + " byte");  
    }  
}
```

ÇIKTI (BOYUT SAYFAYA GÖRE DEĞİŞİR)

- Kaynak bir dosya değil, **ağ**: `url.openStream()` bir `InputStream` döndürür; kopyalama döngüsü aynıdır.
- `new URL(String)` Java 20'den beri kullanımdan kaldırıldı; `URI.create(...).toURL()` önerilir.
- `for` döngüsü: oku, -1 değilse yaz, tekrar oku.

DOSYA KOPYALAMA

Kısa Yollar: `transferTo` ve `Files.copy`

```
public class KopyaKisa {  
    public static void main(String[] args) throws IOException {  
        // Java 9+: transferTo  
        try (InputStream in = new FileInputStream("dosya.txt");  
             OutputStream out = new FileOutputStream("dosya2.txt")) {  
            long adet = in.transferTo(out);  
            System.out.println(adet + " byte aktarıldı");  
        }  
        // Java 7+: Files.copy  
        Files.copy(Path.of("dosya.txt"), Path.of("dosya3.txt"),  
                   StandardCopyOption.REPLACE_EXISTING);  
        System.out.println(Files.size(Path.of("dosya3.txt")) + " byte kopyalandı");  
    }  
}
```

ÇIKTI

```
4 byte aktarıldı  
4 byte kopyalandı
```

Kütüphane metotları içeride blok kopyalama yapar. `Files.copy` hedef varsa `REPLACE_EXISTING` seçeneği olmadan `FileAlreadyExistsException` fırlatır.

Kopyalama Yöntemleri

Yöntem	Birim	Uygun olduğu durum
Bayt bayt (<code>read()</code> / <code>write(int)</code>)	1 bayt	öğrenmek için; tamponsuz çok yavaş
Blok blok (<code>read(byte[])</code>)	8–16 KB	her tür dosya, hızlı
Satır satır (<code>readLine</code> / <code>println</code>)	1 satır	yalnızca metin; satırları işlerken
<code>in.transferTo(out)</code> (Java 9+)	blok	iki akış arasında tek satırda kopya
<code>Files.copy(kaynak, hedef)</code> (Java 7+)	blok	dosyadan dosyaya en kısa yol



Hangi yöntem seçilirse seçilsin: **tamponlu** çalışın, akışları **kapatın** (tercihen try-with-resources) ve `IOException` 'ı ele alın.

Dosya Erişimi: Sıralı ve Rastgele

Sıralı erişim (sequential)

- Her okuma/yazma bir **sonraki** konumdan devam eder; geri dönülemez.
- Dosya içeriğinin yapısı bilinmediğinde veya kopya oluşturulurken kullanılır.
- Akışların (`FileInputStream`, `BufferedReader` ...) hepsi sıralıdır.

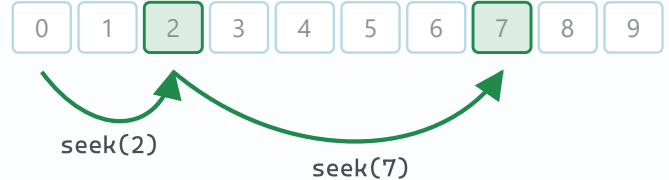
Sıralı erişim: kaset gibi, baştan sona



Rastgele erişim (random access)

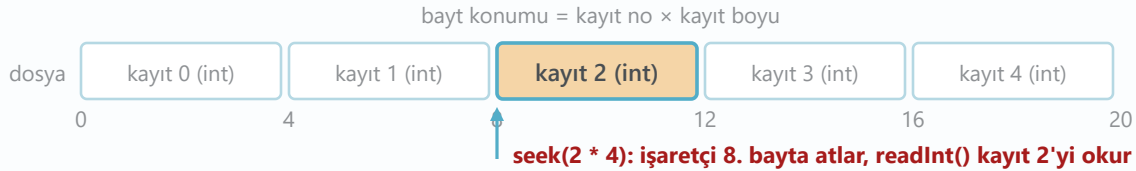
- Dosya içinde **istenen konuma** hareket edilip oradan okunur/yazılır.
- Kayıtların **boyutu ve konumu** biliniyorsa kullanılabilir: $k. \text{ kayıt} = k \times \text{ kayıt boyutu}$.
- Belirli bir kaydı okumak veya **yerinde değiştirmek** için kullanışlıdır.

Rastgele erişim: plak gibi, istenen yere atla



java.io.RandomAccessFile

Üye	Görevi
<code>RandomAccessFile(String ad, String mod)</code>	mod <code>"r"</code> : okuma, <code>"rw"</code> : okuma ve yazma
<code>seek(long konum)</code>	dosya işaretçisini belirtilen konuma taşır
<code>getFilePointer()</code>	işaretçinin dosya başına uzaklığı (bayt)
<code>length()</code>	dosyanın uzunluğu (bayt)
<code>read()</code> / <code>write(int b)</code>	bir bayt okur (int olarak döndürür) / yazar
<code>readInt()</code> , <code>writeInt(int)</code> , <code>readDouble()</code> ...	ilkel tipleri ikili okur/yazar (int: 4 bayt)
<code>close()</code>	sistem kaynaklarını serbest bırakır



Akışlardan farklı olarak tek bir nesne hem okur hem yazar; `InputStream` / `OutputStream` hiyerarşisinde değildir.

Sıralı ve Rastgele Erişim: Kod

```
public class RasgeleErisim {
    public static void main(String[] args) throws IOException {
        // Sıralı Erişim
        FileInputStream sirali = new FileInputStream("dosya");
        int bayt = sirali.read();
        sirali.close();
        System.out.println("İlk byte: " + bayt);
        // Rasgele Erişim
        RandomAccessFile rast = new RandomAccessFile("dosya", "rw");
        // Dosyanın başından 100 byte ilerideki konuma git
        rast.seek(100);
        // 100. konumdan itibaren 4 byte'lık bir int oku
        int veri = rast.readInt();
        System.out.println("100. konumdaki int: " + veri);
        rast.close();
    }
}
```

ÇIKTI

İlk byte: 1
100. konumdaki int:
256

`seek(100)` işaretçiyi 100 konumuna, yani **101.** bayta taşır (konumlar 0'dan başlar). Oradaki 4 bayt `00 00 01 00` olduğundan `readInt()` = 256 döner. `dosya` burada 124 baytlık bir veri dosyasıdır.

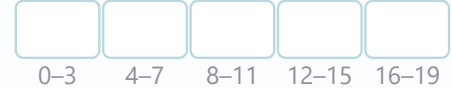
Kayıtlara Rastgele Erişim: Adım Adım

```

1 import java.io.*;
2
3 public class Kayitlar {
4     public static void main(String[] args) throws IOException {
5         RandomAccessFile f = new RandomAccessFile("n.dat", "rw");
6         int[] notlar = {70, 85, 90, 45, 60};
7         for (int n : notlar)
8             f.writeInt(n);           // her kayıt 4 byte
9         System.out.println("Uzunluk: " + f.length());
10        f.seek(2 * 4);               // 3. kayda git
11        System.out.println("3. not: " + f.readInt());
12        System.out.println("Konum: " + f.getFilePointer());
13        f.seek(3 * 4);               // 4. kayıt
14        f.writeInt(50);              // 45 yerine 50
15        f.seek(0);
16        int toplam = 0;
17        while (f.getFilePointer() < f.length())
18            toplam += f.readInt();
19        System.out.println("Toplam: " + toplam);
20        f.close();
21    }
22 }

```

n.dat: her kayıt 4 bayt (int)



İşaretçi = 0

"rw" : dosya yoksa oluşturulur. İşaretçi 0'da.

Adım 1 / 13

ÇIKTI

DEĞİŞKENLER

ÇAĞRI YİĞİNİ

main()

Kayıtlara Rastgele Erişim: Adım Adım

```

1 import java.io.*;
2
3 public class Kayitlar {
4     public static void main(String[] args) throws IOException {
5         RandomAccessFile f = new RandomAccessFile("n.dat", "rw");
6         int[] notlar = {70, 85, 90, 45, 60};
7         for (int n : notlar)
8             f.writeInt(n);           // her kayıt 4 byte
9         System.out.println("Uzunluk: " + f.length());
10        f.seek(2 * 4);               // 3. kayda git
11        System.out.println("3. not: " + f.readInt());
12        System.out.println("Konum: " + f.getFilePointer());
13        f.seek(3 * 4);               // 4. kayıt
14        f.writeInt(50);              // 45 yerine 50
15        f.seek(0);
16        int toplam = 0;
17        while (f.getFilePointer() < f.length())
18            toplam += f.readInt();
19        System.out.println("Toplam: " + toplam);
20        f.close();
21    }
22 }

```

n.dat: her kayıt 4 bayt (int)



▲ işaretçi = 0

Yazılacak beş not.

Adım 2 / 13

ÇIKTI

DEĞİŞKENLER

notlar = [70, 85, 90, 45, 60]

ÇAĞRI YİĞİNİ

main()

Kayıtlara Rastgele Erişim: Adım Adım

```

1 import java.io.*;
2
3 public class Kayitlar {
4     public static void main(String[] args) throws IOException {
5         RandomAccessFile f = new RandomAccessFile("n.dat", "rw");
6         int[] notlar = {70, 85, 90, 45, 60};
7         for (int n : notlar)
8             f.writeInt(n); // her kayıt 4 byte
9         System.out.println("Uzunluk: " + f.length());
10        f.seek(2 * 4); // 3. kayda git
11        System.out.println("3. not: " + f.readInt());
12        System.out.println("Konum: " + f.getFilePointer());
13        f.seek(3 * 4); // 4. kayıt
14        f.writeInt(50); // 45 yerine 50
15        f.seek(0);
16        int toplam = 0;
17        while (f.getFilePointer() < f.length())
18            toplam += f.readInt();
19        System.out.println("Toplam: " + toplam);
20        f.close();
21    }
22 }

```

n.dat: her kayıt 4 bayt (int)



işaretçi = 4

Her `writeInt` 4 bayt yazar ve işaretçiyi 4 ilerletir.

Adım 3 / 13

ÇIKTI

DEĞİŞKENLER

n = 70

ÇAĞRI YİĞİNİ

main()

Kayıtlara Rastgele Erişim: Adım Adım

```

1 import java.io.*;
2
3 public class Kayitlar {
4     public static void main(String[] args) throws IOException {
5         RandomAccessFile f = new RandomAccessFile("n.dat", "rw");
6         int[] notlar = {70, 85, 90, 45, 60};
7         for (int n : notlar)
8             f.writeInt(n); // her kayıt 4 byte
9         System.out.println("Uzunluk: " + f.length());
10        f.seek(2 * 4); // 3. kayda git
11        System.out.println("3. not: " + f.readInt());
12        System.out.println("Konum: " + f.getFilePointer());
13        f.seek(3 * 4); // 4. kayıt
14        f.writeInt(50); // 45 yerine 50
15        f.seek(0);
16        int toplam = 0;
17        while (f.getFilePointer() < f.length())
18            toplam += f.readInt();
19        System.out.println("Toplam: " + toplam);
20        f.close();
21    }
22 }

```

n.dat: her kayıt 4 bayt (int)



İşaretçi = 20 ▲

Döngü bitti: $5 \times 4 = 20$ bayt, işaretçi dosya sonunda (20).

Adım 4 / 13

ÇIKTI

DEĞİŞKENLER

n = 60

ÇAĞRI YİĞİNİ

main()

Kayıtlara Rastgele Erişim: Adım Adım

```

1 import java.io.*;
2
3 public class Kayitlar {
4     public static void main(String[] args) throws IOException {
5         RandomAccessFile f = new RandomAccessFile("n.dat", "rw");
6         int[] notlar = {70, 85, 90, 45, 60};
7         for (int n : notlar)
8             f.writeInt(n);           // her kayıt 4 byte
9         System.out.println("Uzunluk: " + f.length());
10        f.seek(2 * 4);               // 3. kayda git
11        System.out.println("3. not: " + f.readInt());
12        System.out.println("Konum: " + f.getFilePointer());
13        f.seek(3 * 4);               // 4. kayıt
14        f.writeInt(50);               // 45 yerine 50
15        f.seek(0);
16        int toplam = 0;
17        while (f.getFilePointer() < f.length())
18            toplam += f.readInt();
19        System.out.println("Toplam: " + toplam);
20        f.close();
21    }
22 }

```

n.dat: her kayıt 4 bayt (int)

70	85	90	45	60
0-3	4-7	8-11	12-15	16-19

işaretçi = 20 ▲

length() = 20.

Adım 5 / 13

ÇIKTI

Uzunluk: 20

DEĞİŞKENLER

ÇAĞRI YİĞİNİ

main()

Kayıtlara Rastgele Erişim: Adım Adım

```

1 import java.io.*;
2
3 public class Kayitlar {
4     public static void main(String[] args) throws IOException {
5         RandomAccessFile f = new RandomAccessFile("n.dat", "rw");
6         int[] notlar = {70, 85, 90, 45, 60};
7         for (int n : notlar)
8             f.writeInt(n);           // her kayıt 4 byte
9         System.out.println("Uzunluk: " + f.length());
10        ► f.seek(2 * 4);             // 3. kayda git
11        System.out.println("3. not: " + f.readInt());
12        System.out.println("Konum: " + f.getFilePointer());
13        f.seek(3 * 4);               // 4. kayıt
14        f.writeInt(50);              // 45 yerine 50
15        f.seek(0);
16        int toplam = 0;
17        while (f.getFilePointer() < f.length())
18            toplam += f.readInt();
19        System.out.println("Toplam: " + toplam);
20        f.close();
21    }
22 }

```

n.dat: her kayıt 4 bayt (int)

70	85	90	45	60
0-3	4-7	8-11	12-15	16-19

▲
işaretçi = 8

3. kayıt (indis 2) 8. baytta başlar: doğrudan oraya **atlanır**, öncekiler okunmaz.

Adım 6 / 13

ÇIKTI

Uzunluk: 20

DEĞİŞKENLER

ÇAĞRI YİĞİNİ

main()

Kayıtlara Rastgele Erişim: Adım Adım

```

1 import java.io.*;
2
3 public class Kayitlar {
4     public static void main(String[] args) throws IOException {
5         RandomAccessFile f = new RandomAccessFile("n.dat", "rw");
6         int[] notlar = {70, 85, 90, 45, 60};
7         for (int n : notlar)
8             f.writeInt(n);           // her kayıt 4 byte
9         System.out.println("Uzunluk: " + f.length());
10        f.seek(2 * 4);               // 3. kayda git
11        System.out.println("3. not: " + f.readInt());
12        System.out.println("Konum: " + f.getFilePointer());
13        f.seek(3 * 4);               // 4. kayıt
14        f.writeInt(50);               // 45 yerine 50
15        f.seek(0);
16        int toplam = 0;
17        while (f.getFilePointer() < f.length())
18            toplam += f.readInt();
19        System.out.println("Toplam: " + toplam);
20        f.close();
21    }
22 }

```

n.dat: her kayıt 4 bayt (int)



▲
işaretçi = 12

readInt() 8-11. baytları okur: 90.
İşaretçi 12'ye ilerler.

Adım 7 / 13

ÇIKTI

```

Uzunluk: 20
3. not: 90

```

DEĞİŞKENLER

—

ÇAĞRI YİĞİNİ

f.readInt()

main()

Kayıtlara Rastgele Erişim: Adım Adım

```

1 import java.io.*;
2
3 public class Kayitlar {
4     public static void main(String[] args) throws IOException {
5         RandomAccessFile f = new RandomAccessFile("n.dat", "rw");
6         int[] notlar = {70, 85, 90, 45, 60};
7         for (int n : notlar)
8             f.writeInt(n);           // her kayıt 4 byte
9         System.out.println("Uzunluk: " + f.length());
10        f.seek(2 * 4);               // 3. kayda git
11        System.out.println("3. not: " + f.readInt());
12        ▶ System.out.println("Konum: " + f.getFilePointer());
13        f.seek(3 * 4);               // 4. kayıt
14        f.writeInt(50);              // 45 yerine 50
15        f.seek(0);
16        int toplam = 0;
17        while (f.getFilePointer() < f.length())
18            toplam += f.readInt();
19        System.out.println("Toplam: " + toplam);
20        f.close();
21    }
22 }

```

n.dat: her kayıt 4 bayt (int)

70	85	90	45	60
0-3	4-7	8-11	12-15	16-19

▲
işaretçi = 12

getFilePointer() = 12.

Adım 8 / 13

ÇIKTI

```

Uzunluk: 20
3. not: 90
Konum: 12

```

DEĞİŞKENLER

—

ÇAĞRI YİĞİNİ

main()

Kayıtlara Rastgele Erişim: Adım Adım

```

1 import java.io.*;
2
3 public class Kayitlar {
4     public static void main(String[] args) throws IOException {
5         RandomAccessFile f = new RandomAccessFile("n.dat", "rw");
6         int[] notlar = {70, 85, 90, 45, 60};
7         for (int n : notlar)
8             f.writeInt(n);                // her kayıt 4 byte
9         System.out.println("Uzunluk: " + f.length());
10        f.seek(2 * 4);                    // 3. kayda git
11        System.out.println("3. not: " + f.readInt());
12        System.out.println("Konum: " + f.getFilePointer());
13        ► f.seek(3 * 4);                  // 4. kayıt
14        f.writeInt(50);                   // 45 yerine 50
15        f.seek(0);
16        int toplam = 0;
17        while (f.getFilePointer() < f.length())
18            toplam += f.readInt();
19        System.out.println("Toplam: " + toplam);
20        f.close();
21    }
22 }

```

n.dat: her kayıt 4 bayt (int)



▲
işaretçi = 12

4. kayıt da 12'de başlıyor; işaretçi zaten orada.

Adım 9 / 13

ÇIKTI

```

Uzunluk: 20
3. not: 90
Konum: 12

```

DEĞİŞKENLER

ÇAĞRI YİĞİNİ

main()

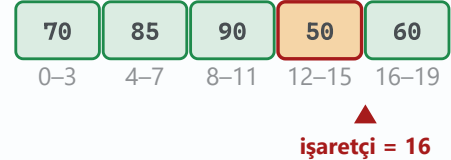
Kayıtlara Rastgele Erişim: Adım Adım

```

1 import java.io.*;
2
3 public class Kayitlar {
4     public static void main(String[] args) throws IOException {
5         RandomAccessFile f = new RandomAccessFile("n.dat", "rw");
6         int[] notlar = {70, 85, 90, 45, 60};
7         for (int n : notlar)
8             f.writeInt(n); // her kayıt 4 byte
9         System.out.println("Uzunluk: " + f.length());
10        f.seek(2 * 4); // 3. kayda git
11        System.out.println("3. not: " + f.readInt());
12        System.out.println("Konum: " + f.getFilePointer());
13        f.seek(3 * 4); // 4. kayıt
14        f.writeInt(50); // 45 yerine 50
15        f.seek(0);
16        int toplam = 0;
17        while (f.getFilePointer() < f.length())
18            toplam += f.readInt();
19        System.out.println("Toplam: " + toplam);
20        f.close();
21    }
22 }

```

n.dat: her kayıt 4 bayt (int)



45 yerine 50 **yerinde** yazılır; dosyanın geri kalanı değişmez. Sıralı akışla bu yapılamaz.

Adım 10 / 13

ÇIKTI

```

Uzunluk: 20
3. not: 90
Konum: 12

```

DEĞİŞKENLER

ÇAĞRI YİĞİNİ

main()

Kayıtlara Rastgele Erişim: Adım Adım

```

1 import java.io.*;
2
3 public class Kayitlar {
4     public static void main(String[] args) throws IOException {
5         RandomAccessFile f = new RandomAccessFile("n.dat", "rw");
6         int[] notlar = {70, 85, 90, 45, 60};
7         for (int n : notlar)
8             f.writeInt(n);           // her kayıt 4 byte
9         System.out.println("Uzunluk: " + f.length());
10        f.seek(2 * 4);               // 3. kayda git
11        System.out.println("3. not: " + f.readInt());
12        System.out.println("Konum: " + f.getFilePointer());
13        f.seek(3 * 4);               // 4. kayıt
14        f.writeInt(50);               // 45 yerine 50
15        ▶ f.seek(0);
16        int toplam = 0;
17        while (f.getFilePointer() < f.length())
18            toplam += f.readInt();
19        System.out.println("Toplam: " + toplam);
20        f.close();
21    }
22 }

```

n.dat: her kayıt 4 bayt (int)

70	85	90	50	60
0-3	4-7	8-11	12-15	16-19

▲ işaretçi = 0

Başa dönülür.

Adım 11 / 13

ÇIKTI

```

Uzunluk: 20
3. not: 90
Konum: 12

```

DEĞİŞKENLER

toplam = 0

ÇAĞRI YİĞİNİ

main()

Kayıtlara Rastgele Erişim: Adım Adım

```

1 import java.io.*;
2
3 public class Kayitlar {
4     public static void main(String[] args) throws IOException {
5         RandomAccessFile f = new RandomAccessFile("n.dat", "rw");
6         int[] notlar = {70, 85, 90, 45, 60};
7         for (int n : notlar)
8             f.writeInt(n);           // her kayıt 4 byte
9         System.out.println("Uzunluk: " + f.length());
10        f.seek(2 * 4);               // 3. kayda git
11        System.out.println("3. not: " + f.readInt());
12        System.out.println("Konum: " + f.getFilePointer());
13        f.seek(3 * 4);               // 4. kayıt
14        f.writeInt(50);               // 45 yerine 50
15        f.seek(0);
16        int toplam = 0;
17        while (f.getFilePointer() < f.length())
18            toplam += f.readInt();
19        System.out.println("Toplam: " + toplam);
20        f.close();
21    }
22 }

```

n.dat: her kayıt 4 bayt (int)

70	85	90	50	60
0-3	4-7	8-11	12-15	16-19

İşaretçi = 20 ▲

İşaretçi uzunluğa ulaşana kadar okunur: 70
+ 85 + 90 + 50 + 60 = 355.

Adım 12 / 13

ÇIKTI

```

Uzunluk: 20
3. not: 90
Konum: 12

```

DEĞİŞKENLER

toplam = 355

ÇAĞRI YİĞİNİ

main()

Kayıtlara Rastgele Erişim: Adım Adım

```

1 import java.io.*;
2
3 public class Kayitlar {
4     public static void main(String[] args) throws IOException {
5         RandomAccessFile f = new RandomAccessFile("n.dat", "rw");
6         int[] notlar = {70, 85, 90, 45, 60};
7         for (int n : notlar)
8             f.writeInt(n);           // her kayıt 4 byte
9         System.out.println("Uzunluk: " + f.length());
10        f.seek(2 * 4);               // 3. kayda git
11        System.out.println("3. not: " + f.readInt());
12        System.out.println("Konum: " + f.getFilePointer());
13        f.seek(3 * 4);               // 4. kayıt
14        f.writeInt(50);               // 45 yerine 50
15        f.seek(0);
16        int toplam = 0;
17        while (f.getFilePointer() < f.length())
18            toplam += f.readInt();
19        ▶ System.out.println("Toplam: " + toplam);
20        f.close();
21    }
22 }

```

n.dat: her kayıt 4 bayt (int)

70	85	90	50	60
0-3	4-7	8-11	12-15	16-19

işaretçi = 20 ▲

Sonuç yazdırılır; ardından dosya kapatılır.

Adım 13 / 13

ÇIKTI

```

Uzunluk: 20
3. not: 90
Konum: 12
Toplam: 355

```

DEĞİŞKENLER

toplam = 355

ÇAĞRI YİĞİNİ

main()

Serileştirme (Serialization)

- **Serileştirme**: bir nesnenin durumunu (alan değerlerini) **bayt dizisine** çevirmek; bu sayede nesne dosyaya yazılabilir, ağ üzerinden başka bir ortama taşınabilir. Ters **serileştirmeyi geri alma** (deserialization).
- Bir sınıfın nesneleri serileştirilebilmesi için sınıf `java.io.Serializable` arayüzünü **gerçekleştirmelidir** (metodu olmayan bir işaret arayüzü).
- Sınıfın sürüm kimliği `serialVersionUID` ile belirtilmesi **önerilir**; yazılmazsa derleyici sınıf yapısından üretir ve sınıf değişince eski dosyalar okunamaz.
- `static` alanlar (sınıfa aittir) ve `transient` (geçici) olarak işaretlenen alanlar **serileştirilmez**; şifre gibi alanları dışarıda bırakmak için kullanılır.

Kullanım alanları: oyun kayıtları, önbellekler, nesnelerin ağ üzerinden başka bir JVM'e gönderilmesi. Uzun süreli ve diller arası veri saklamak için JSON gibi metin biçimleri daha çok tercih edilir.

Serileştirme Akışı

bellek (heap)

Ogrenci nesnesi

ad = "Ahmet"

yas = 20

Bellekte bir **Ogrenci** nesnesi var.

Adım 1 / 3

Serileştirme Akışı



`ObjectOutputStream.writeObject` nesneyi baytlara çevirir; dosya `AC ED` sihirli sayısıyla başlar.

Serileştirme Akışı



`ObjectInputStream.readObject` baytlardan **yeni** bir nesne kurar: aynı değerler, farklı nesne.

SERİLEŞTİRME

Serileştirilebilir Sınıf

```
import java.io.Serializable;

class Ogrenci implements Serializable {
    private static final long serialVersionUID = 1L;
    private String ad;
    private int yas;

    public Ogrenci(String ad, int yas) {
        this.ad = ad;
        this.yas = yas;
    }

    public String toString() {
        return "Ogrenci [ad=" + ad + ", yas=" + yas + "]";
    }
}
```

- `implements Serializable` : metodu olmayan bir **işaret** arayüzü.
- `serialVersionUID` : sınıfın sürüm numarası. Yazan ve okuyan sürüm farklıysa `InvalidClassException` .
- Alanlar da serileştirilebilir olmalı; değilse `NotSerializableException` .
- `readObject` `Object` döndürür: `(Ogrenci)` dönüşümü gerekir.

SERİLEŞTİRME

Nesneyi Yazma ve Geri Okuma

```
// Serileştirme işlemi
try (ObjectOutputStream cikti = new ObjectOutputStream(
    new FileOutputStream("ogrenci.ser"))) {
    Ogresci ogrenci = new Ogresci("Ahmet", 20);
    cikti.writeObject(ogrenci);
    System.out.println("Serileştirme başarılı.");
} catch (IOException e) {
    e.printStackTrace();
}

System.out.println("ogrenci.ser: " + new File("ogrenci.ser").length() + " byte");

// Seriyi okuma işlemi
try (ObjectInputStream girdi = new ObjectInputStream(
    new FileInputStream("ogrenci.ser"))) {
    Ogresci ogrenci = (Ogresci) girdi.readObject();
    System.out.println("Ogresci: " + ogrenci);
} catch (IOException | ClassNotFoundException e) {
    e.printStackTrace();
}
```

ÇIKTI

```
Serileştirme başarılı.
ogrenci.ser: 72 byte
Ogresci: Ogresci [ad=Ahmet, yas=20]
```

transient ve static Alanlar

```
class Kullanici implements Serializable {  
    private static final long serialVersionUID = 1L;  
    static int sayac;  
    private String ad;  
    private transient String sifre;    // yazılmaz  
  
    Kullanici(String ad, String sifre) {  
        this.ad = ad;  
        this.sifre = sifre;  
    }  
  
    public String toString() {  
        return "Kullanici [ad=" + ad + ", sifre=" + sifre + "];"  
    }  
}
```

- **transient** : bu alan serileştirmede **atlanır**; geri okunduğunda varsayılan değeri alır (**null** , 0, **false**).
- Şifre, geçici önbellek, açık dosya/bağlantı gibi saklanmaması gereken veya saklanamayan alanlar için.
- **static** alanlar nesneye değil sınıfa aittir: dosyaya hiç yazılmaz.

SERİLEŞTİRME

transient ve static: Deneme

```
Kullanici k = new Kullanici("ayse", "gizli123");
Kullanici.sayac = 5;
ObjectOutputStream c = new ObjectOutputStream(
    new FileOutputStream("k.ser"));
c.writeObject(k);
c.close();
Kullanici.sayac = 99;           // static: sınıfa ait
ObjectInputStream g = new ObjectInputStream(
    new FileInputStream("k.ser"));
Kullanici okunan = (Kullanici) g.readObject();
g.close();
System.out.println(okunan);
System.out.println("sayac = " + Kullanici.sayac);
System.out.println("aynı nesne mi? " + (okunan == k));
```

ÇIKTI

```
Kullanici [ad=ayse, sifre=null]
sayac = 99
aynı nesne mi? false
```

- `sifre` → `null` : yazılmadı.
- `sayac` → 99: bellekteki son değer.
- `==` yanlış: okunan, **yeni** bir nesne.

Serileştirme: Yazma ve Geri Okuma Adım Adım

```
► Kullanici k = new Kullanici("ayse", "gizli123");
2 Kullanici.sayac = 5;
3 ObjectOutputStream c = new ObjectOutputStream(
4     new FileOutputStream("k.ser"));
5 c.writeObject(k);
6 c.close();
7 Kullanici.sayac = 99;           // static: sınıfa ait
8 ObjectInputStream g = new ObjectInputStream(
9     new FileInputStream("k.ser"));
10 Kullanici okunan = (Kullanici) g.readObject();
11 g.close();
12 System.out.println(okunan);
13 System.out.println("sayac = " + Kullanici.sayac);
14 System.out.println("aynı nesne mi? " + (okunan == k));
```

sınıf alanı

Kullanici.sayac

0

heap

k

ad="ayse" sifre="gizli123"

k.ser

dosya yok

Heap'te yeni bir **Kullanici** nesnesi: **ad** ve **sifre** dolu. **static** **sayac** sınıf alanında, 0.

Adım 1 / 11

DEĞİŞKENLER

k = Kullanici@1

ÇIKTI



Serileştirme: Yazma ve Geri Okuma Adım Adım

```
1 Kullanici k = new Kullanici("ayse", "gizli123");
2
3 ► Kullanici.sayac = 5;
4
5 ObjectOutputStream c = new ObjectOutputStream(
6     new FileOutputStream("k.ser"));
7 c.writeObject(k);
8 c.close();
9
10 Kullanici.sayac = 99;           // static: sınıfa ait
11
12 ObjectInputStream g = new ObjectInputStream(
13     new FileInputStream("k.ser"));
14
15 Kullanici okunan = (Kullanici) g.readObject();
16 g.close();
17
18 System.out.println(okunan);
19
20 System.out.println("sayac = " + Kullanici.sayac);
21
22 System.out.println("aynı nesne mi? " + (okunan == k));
```

sınıf alanı

Kullanici.sayac

5

heap

k

ad="ayse" sifre="gizli123"

k.ser

dosya yok

sayac nesneye değil **sınıfa** aittir; tek kopyası vardır.

Adım 2 / 11

DEĞİŞKENLER

k = Kullanici@1**Kullanici.sayac = 5**

ÇIKTI

Serileştirme: Yazma ve Geri Okuma Adım Adım

```
1 Kullanici k = new Kullanici("ayse", "gizli123");
2 Kullanici.sayac = 5;
▶ ObjectOutputStream c = new ObjectOutputStream(
▶   new FileOutputStream("k.ser"));
5 c.writeObject(k);
6 c.close();
7 Kullanici.sayac = 99;           // static: sınıfa ait
8 ObjectInputStream g = new ObjectInputStream(
9   new FileInputStream("k.ser"));
10 Kullanici okunan = (Kullanici) g.readObject();
11 g.close();
12 System.out.println(okunan);
13 System.out.println("sayac = " + Kullanici.sayac);
14 System.out.println("aynı nesne mi? " + (okunan == k));
```

sınıf alanı

Kullanici.sayac

5

heap

k

ad="ayse" sifre="gizli123"

k.ser

4 bayt

akış başlığı

Akış açılır ve dosyaya hemen 4 baytlık **akış başlığını** yazar.

Adım 3 / 11

DEĞİŞKENLER

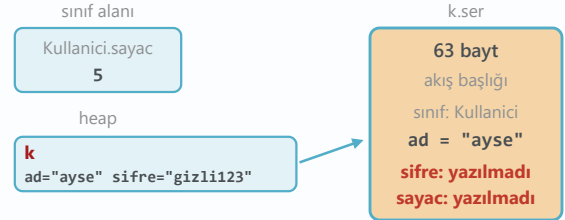
c = ObjectOutputStream

ÇIKTI



Serileştirme: Yazma ve Geri Okuma Adım Adım

```
1 Kullanici k = new Kullanici("ayse", "gizli123");
2 Kullanici.sayac = 5;
3 ObjectOutputStream c = new ObjectOutputStream(
4     new FileOutputStream("k.ser"));
5 c.writeObject(k);
6 c.close();
7 Kullanici.sayac = 99;           // static: sınıfa ait
8 ObjectInputStream g = new ObjectInputStream(
9     new FileInputStream("k.ser"));
10 Kullanici okunan = (Kullanici) g.readObject();
11 g.close();
12 System.out.println(okunan);
13 System.out.println("sayac = " + Kullanici.sayac);
14 System.out.println("aynı nesne mi? " + (okunan == k));
```



`writeObject` sınıf tanımını ve alanları yazar: dosya 63 bayt. `transient sifre` ve `static sayac` yazılmaz.

Adım 4 / 11

DEĞİŞKENLER

c = ObjectOutputStream

ÇIKTI



Serileştirme: Yazma ve Geri Okuma Adım Adım

```
1 Kullanici k = new Kullanici("ayse", "gizli123");
2 Kullanici.sayac = 5;
3 ObjectOutputStream c = new ObjectOutputStream(
4     new FileOutputStream("k.ser"));
5 c.writeObject(k);
6 c.close();
7 Kullanici.sayac = 99;           // static: sınıfa ait
8 ObjectInputStream g = new ObjectInputStream(
9     new FileInputStream("k.ser"));
10 Kullanici okunan = (Kullanici) g.readObject();
11 g.close();
12 System.out.println(okunan);
13 System.out.println("sayac = " + Kullanici.sayac);
14 System.out.println("aynı nesne mi? " + (okunan == k));
```

sınıf alanı

Kullanici.sayac

5

heap

k

ad="ayse" sifre="gizli123"

k.ser

63 bayt

akış başlığı

sınıf: Kullanici

ad = "ayse"

sifre: yazılmadı

sayac: yazılmadı

Akış kapatılır; dosya 63 bayt olarak kalır.

Adım 5 / 11

DEĞİŞKENLER

c = kapalı

ÇIKTI



Serileştirme: Yazma ve Geri Okuma Adım Adım

```
1 Kullanici k = new Kullanici("ayse", "gizli123");
2 Kullanici.sayac = 5;
3 ObjectOutputStream c = new ObjectOutputStream(
4     new FileOutputStream("k.ser"));
5 c.writeObject(k);
6 c.close();
7
8 ▶ Kullanici.sayac = 99;           // static: sınıfa ait
9
10 ObjectInputStream g = new ObjectInputStream(
11     new FileInputStream("k.ser"));
12 Kullanici okunan = (Kullanici) g.readObject();
13 g.close();
14 System.out.println(okunan);
15 System.out.println("sayac = " + Kullanici.sayac);
16 System.out.println("aynı nesne mi? " + (okunan == k));
```

sınıf alanı

Kullanici.sayac
99

heap

k
ad="ayse" sifre="gizli123"

k.ser

63 bayt

akış başlığı

sınıf: Kullanici

ad = "ayse"

sifre: yazılmadı

sayac: yazılmadı

Sınıftaki **sayac** 99 olur. Dosyada **sayac** olmadığı için okuma bunu değiştiremez.

Adım 6 / 11

DEĞİŞKENLER

Kullanici.sayac = 99

ÇIKTI



Serileştirme: Yazma ve Geri Okuma Adım Adım

```
1 Kullanici k = new Kullanici("ayse", "gizli123");
2 Kullanici.sayac = 5;
3 ObjectOutputStream c = new ObjectOutputStream(
4     new FileOutputStream("k.ser"));
5 c.writeObject(k);
6 c.close();
7 Kullanici.sayac = 99;           // static: sınıfa ait
8
9 ▶ ObjectInputStream g = new ObjectInputStream(
10    new FileInputStream("k.ser"));
11
12 Kullanici okunan = (Kullanici) g.readObject();
13 g.close();
14 System.out.println(okunan);
15 System.out.println("sayac = " + Kullanici.sayac);
16 System.out.println("aynı nesne mi? " + (okunan == k));
```

sınıf alanı

Kullanici.sayac
99

heap

k
ad="ayse" sifre="gizli123"

k.ser

63 bayt
akış başlığı
sınıf: Kullanici
ad = "ayse"
sifre: yazılmadı
sayac: yazılmadı

Okuma akışı açılır; önce akış başlığını denetler.

Adım 7 / 11

DEĞİŞKENLER

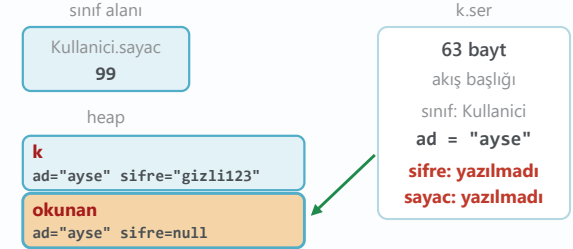
g = ObjectInputStream

ÇIKTI



Serileştirme: Yazma ve Geri Okuma Adım Adım

```
1 Kullanici k = new Kullanici("ayse", "gizli123");
2 Kullanici.sayac = 5;
3 ObjectOutputStream c = new ObjectOutputStream(
4     new FileOutputStream("k.ser"));
5 c.writeObject(k);
6 c.close();
7 Kullanici.sayac = 99;           // static: sınıfa ait
8 ObjectInputStream g = new ObjectInputStream(
9     new FileInputStream("k.ser"));
10 ► Kullanici okunan = (Kullanici) g.readObject();
11 g.close();
12 System.out.println(okunan);
13 System.out.println("sayac = " + Kullanici.sayac);
14 System.out.println("aynı nesne mi? " + (okunan == k));
```



Heap'te **yeni** bir nesne kurulur; yapıcı **çağrılmaz**. **ad** dosyadan gelir, **sifre** varsayılan değer **null** olur.

Adım 8 / 11

DEĞİŞKENLER

okunan = Kullanici@2

ÇIKTI



Serileştirme: Yazma ve Geri Okuma Adım Adım

```
1 Kullanici k = new Kullanici("ayse", "gizli123");
2 Kullanici.sayac = 5;
3 ObjectOutputStream c = new ObjectOutputStream(
4     new FileOutputStream("k.ser"));
5 c.writeObject(k);
6 c.close();
7 Kullanici.sayac = 99;           // static: sınıfa ait
8 ObjectInputStream g = new ObjectInputStream(
9     new FileInputStream("k.ser"));
10 Kullanici okunan = (Kullanici) g.readObject();
11 g.close();
12 ► System.out.println(okunan);
13 System.out.println("sayac = " + Kullanici.sayac);
14 System.out.println("aynı nesne mi? " + (okunan == k));
```

sınıf alanı

Kullanici.sayac
99

heap

k
ad="ayse" sifre="gizli123"

okunan
ad="ayse" sifre=null

toString() okunan nesnenin alanlarını
yazar.

k.ser

63 bayt
akış başlığı
sınıf: Kullanici
ad = "ayse"
sifre: yazılmadı
sayac: yazılmadı

Adım 9 / 11

DEĞİŞKENLER

okunan = Kullanici@2

ÇIKTI

Kullanici [ad=ayse, sifre=null]

Serileştirme: Yazma ve Geri Okuma Adım Adım

```
1 Kullanici k = new Kullanici("ayse", "gizli123");
2 Kullanici.sayac = 5;
3 ObjectOutputStream c = new ObjectOutputStream(
4     new FileOutputStream("k.ser"));
5 c.writeObject(k);
6 c.close();
7 Kullanici.sayac = 99;           // static: sınıfa ait
8 ObjectInputStream g = new ObjectInputStream(
9     new FileInputStream("k.ser"));
10 Kullanici okunan = (Kullanici) g.readObject();
11 g.close();
12 System.out.println(okunan);
13 ► System.out.println("sayac = " + Kullanici.sayac);
14 System.out.println("aynı nesne mi? " + (okunan == k));
```

sınıf alanı

Kullanici.sayac
99

heap

k
ad="ayse" sifre="gizli123"**okunan**
ad="ayse" sifre=null

k.ser

63 bayt
akış başlığı
sınıf: Kullanici
ad = "ayse"
sifre: yazılmadı
sayac: yazılmadı**sayac** bellekteki son değerdir: 99.

Adım 10 / 11

DEĞİŞKENLER

Kullanici.sayac = 99

ÇIKTI

**Kullanici [ad=ayse, sifre=null]
sayac = 99**

Serileştirme: Yazma ve Geri Okuma Adım Adım

```
1 Kullanici k = new Kullanici("ayse", "gizli123");
2 Kullanici.sayac = 5;
3 ObjectOutputStream c = new ObjectOutputStream(
4     new FileOutputStream("k.ser"));
5 c.writeObject(k);
6 c.close();
7 Kullanici.sayac = 99;           // static: sınıfa ait
8 ObjectInputStream g = new ObjectInputStream(
9     new FileInputStream("k.ser"));
10 Kullanici okunan = (Kullanici) g.readObject();
11 g.close();
12 System.out.println(okunan);
13 System.out.println("sayac = " + Kullanici.sayac);
▶ System.out.println("aynı nesne mi? " + (okunan == k));
```

sınıf alanı

Kullanici.sayac
99

heap

k
ad="ayse" sifre="gizli123"

okunan
ad="ayse" sifre=null

k.ser

63 bayt
akış başlığı
sınıf: Kullanici
ad = "ayse"
sifre: yazılmadı
sayac: yazılmadı

İki ayrı nesne: içerikleri benzese de referanslar farklıdır.

Adım 11 / 11

DEĞİŞKENLER

okunan == k sonucu=false

ÇIKTI

```
Kullanici [ad=ayse, sifre=null]
sayac = 99
aynı nesne mi? false
```

NIO

NIO (New I/O)

- Java NIO (`java.nio` , Java 1.4+), gelişmiş G/Ç işlemleri için güçlü bir API sunar: tampon (buffer), kanal (channel), seçici (selector) ve dosya sistemi entegrasyonu içerir.
- Klasik `java.io` akışlarına göre **performans** ve **esneklik** sağlar.

Buffer

Veriyi geçici olarak depolayan **bellek alanları**.

Channel

Giriş/çıkış işlemlerini yöneten, **çift yönlü** bağlantı nesneleri.

Selector

Birden çok kanalı **tek iş parçacığında** yönetir; girdi bekleyen kanalları izler.

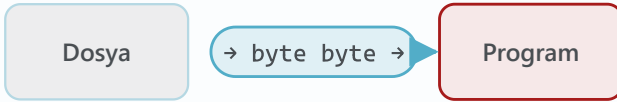
Path

Dosya ve dizin **yollarını** temsil eder (Java 7+, NIO.2).

NIO.2 (`java.nio.file` , Java 7+) dosya işlemleri için `Path` ve `Files` sınıflarını ekledi.

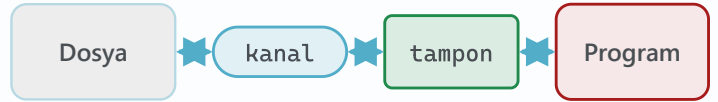
Akış mı, Kanal mı?

Klasik G/Ç: akış



tek yön, sıralı, beklemeli

NIO: kanal + tampon



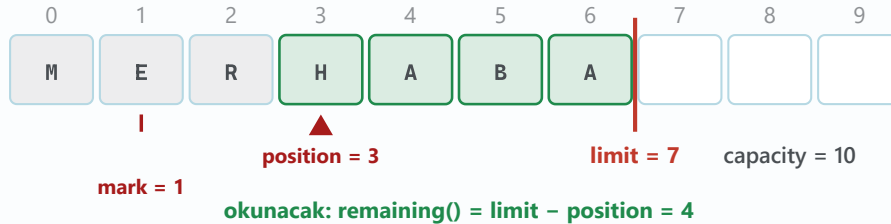
çift yön, bloklar, beklemesiz olabilir

	Klasik G/Ç (java.io)	NIO (java.nio)
Temel kavram	akış (stream)	kanal (channel) + tampon (buffer)
Yön	tek yön: girdi <i>ya da</i> çıktı	kanal çift yönlü olabilir
Birim	bayt / karakter	tampon dolusu blok
Bekleme	her zaman bloklayan	bloklamayan (non-blocking) mod da var
Çok bağlantı	bağlantı başına bir iş parçacığı	tek iş parçacığı + seçici
Ne zaman?	basit dosya/konsol işleri	çok sayıda bağlantılı sunucular, büyük dosyalar

NIO Tamponları

- **Tampon:** verileri geçici olarak saklayan, sabit boyutlu bir bellek alanıdır (bir dizi + konum bilgileri).
- Her ilkel tip için bir sınıf: `ByteBuffer`, `CharBuffer`, `ShortBuffer`, `IntBuffer`, `LongBuffer`, `FloatBuffer`, `DoubleBuffer`.
- Oluşturma: `ByteBuffer.allocate(1024)` veya var olan diziyi sarmak: `ByteBuffer.wrap(dizi)`.

Özellik	Anlamı
kapasite (capacity)	tutulabilecek en fazla öge; hiç değişmez
konum (position)	bir sonraki okuma/yazmanın yapılacağı indis
sınır (limit)	okunabilecek/yazılabilecek ilk olmayan indis
işaret (mark)	<code>mark()</code> ile saklanan konum; <code>reset()</code> oraya döner



Her zaman: $0 \leq \text{mark} \leq \text{position} \leq \text{limit} \leq \text{capacity}$.

ByteBuffer Örneği

```
public class TamponOrnek {  
    public static void main(String[] args) {  
        // 1 KB boyutunda bir ByteBuffer oluştur  
        ByteBuffer buffer = ByteBuffer.allocate(1024);  
        // Veri yazma  
        buffer.put("Merhaba, Java!".getBytes());  
        System.out.println("put sonrası : " + buffer);  
        // Buffer'ı okuma moduna geçirme  
        buffer.flip();  
        System.out.println("flip sonrası : " + buffer);  
        // Veriyi okuma  
        while (buffer.hasRemaining()) {  
            System.out.print((char) buffer.get());  
        }  
        System.out.println();  
    }  
}
```

ÇIKTI

```
put sonrası :  
java.nio.HeapByteBuffer[pos=14  
lim=1024 cap=1024]  
flip sonrası :  
java.nio.HeapByteBuffer[pos=0 lim=14  
cap=1024]  
Merhaba, Java!
```

- `put` yazar, konum 14'e ilerler.
- `flip()` : sınır = 14, konum = 0: **okuma** modu.
- `hasRemaining()` : konum < sınır mı?

Tampon ve Türkçe Karakterler

```
public class TamponUmlaut {  
    public static void main(String[] args) {  
        ByteBuffer b = ByteBuffer.allocate(64);  
        b.put("Dünya".getBytes(StandardCharsets.UTF_8));  
        b.flip();  
        while (b.hasRemaining())  
            System.out.print((char) b.get());    // yanlış  
        System.out.println();  
        b.rewind();  
        System.out.println(StandardCharsets.UTF_8.decode(b)); // doğru  
    }  
}
```

ÇIKTI

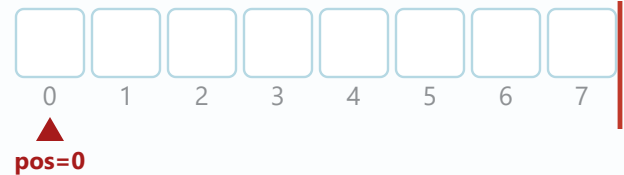
```
D||Enya  
Dünya
```

`(char) buffer.get()` yalnızca ASCII metinde doğrudur. "Dünya" UTF-8'de 6 bayttır (ü = `C3 BC`); baytları tek tek `char` 'a çevirmek ü'yü bozar. Metni çözmek için `Charset.decode` kullanılmalıdır.

put, flip, get, compact, rewind, clear: Adım Adım

```
1 import java.nio.ByteBuffer;
2
3 public class TamponIzle {
4     public static void main(String[] args) {
5         ▶ ByteBuffer b = ByteBuffer.allocate(8);
6         b.put((byte) 'J');
7         b.put((byte) 'A');
8         b.put((byte) 'V');
9         b.put((byte) 'A');
10        b.flip();
11        char c1 = (char) b.get();
12        char c2 = (char) b.get();
13        b.compact();
14        b.put((byte) '!');
15        b.flip();
16        b.get();
17        b.rewind();
18        b.clear();
19        System.out.println(c1 + "" + c2 + " " +
20        b);
21    }
22 }
```

ByteBuffer (capacity = 8) · yazma modu



Boş tampon: konum 0, sınır = kapasite = 8.

Adım 1 / 15

ÇIKTI



DEĞİŞKENLER

pos = 0

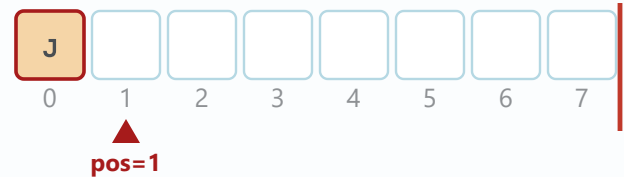
lim = 8

cap = 8

put, flip, get, compact, rewind, clear: Adım Adım

```
1 import java.nio.ByteBuffer;
2
3 public class TamponIzle {
4     public static void main(String[] args) {
5         ByteBuffer b = ByteBuffer.allocate(8);
6         ▶ b.put((byte) 'J');
7         b.put((byte) 'A');
8         b.put((byte) 'V');
9         b.put((byte) 'A');
10        b.flip();
11        char c1 = (char) b.get();
12        char c2 = (char) b.get();
13        b.compact();
14        b.put((byte) '!');
15        b.flip();
16        b.get();
17        b.rewind();
18        b.clear();
19        System.out.println(c1 + "" + c2 + " " +
20        b);
21    }
```

ByteBuffer (capacity = 8) · yazma modu



put konuma yazar ve konumu 1 ilerletir.

Adım 2 / 15

ÇIKTI

DEĞİŞKENLER

pos = 1

lim = 8

cap = 8

put, flip, get, compact, rewind, clear: Adım Adım

```
1 import java.nio.ByteBuffer;
2
3 public class TamponIzle {
4     public static void main(String[] args) {
5         ByteBuffer b = ByteBuffer.allocate(8);
6         b.put((byte) 'J');
7         ▶ b.put((byte) 'A');
8         b.put((byte) 'V');
9         b.put((byte) 'A');
10        b.flip();
11        char c1 = (char) b.get();
12        char c2 = (char) b.get();
13        b.compact();
14        b.put((byte) '!');
15        b.flip();
16        b.get();
17        b.rewind();
18        b.clear();
19        System.out.println(c1 + "" + c2 + " " +
20        b);
21    }
22 }
```

ByteBuffer (capacity = 8) · yazma modu



Konum 2.

Adım 3 / 15

ÇIKTI

DEĞİŞKENLER

pos = 2

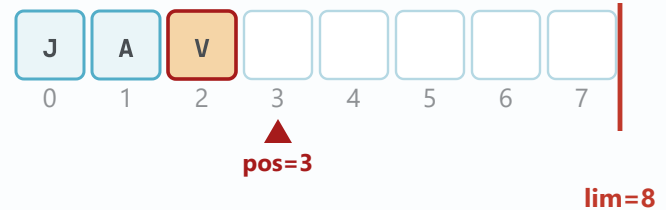
lim = 8

cap = 8

put, flip, get, compact, rewind, clear: Adım Adım

```
1 import java.nio.ByteBuffer;
2
3 public class TamponIzle {
4     public static void main(String[] args) {
5         ByteBuffer b = ByteBuffer.allocate(8);
6         b.put((byte) 'J');
7         b.put((byte) 'A');
8         ▶ b.put((byte) 'V');
9         b.put((byte) 'A');
10        b.flip();
11        char c1 = (char) b.get();
12        char c2 = (char) b.get();
13        b.compact();
14        b.put((byte) '!');
15        b.flip();
16        b.get();
17        b.rewind();
18        b.clear();
19        System.out.println(c1 + "" + c2 + " " +
20        b);
21    }
22 }
```

ByteBuffer (capacity = 8) · yazma modu



Konum 3.

Adım 4 / 15

ÇIKTI

DEĞİŞKENLER

pos = 3

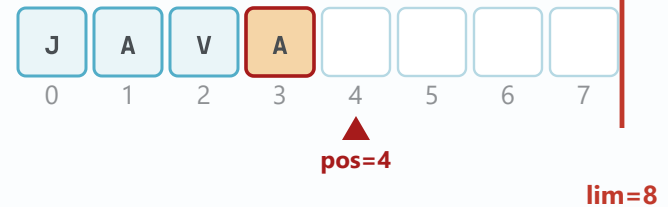
lim = 8

cap = 8

put, flip, get, compact, rewind, clear: Adım Adım

```
1 import java.nio.ByteBuffer;
2
3 public class TamponIzle {
4     public static void main(String[] args) {
5         ByteBuffer b = ByteBuffer.allocate(8);
6         b.put((byte) 'J');
7         b.put((byte) 'A');
8         b.put((byte) 'V');
9         ▶ b.put((byte) 'A');
10        b.flip();
11        char c1 = (char) b.get();
12        char c2 = (char) b.get();
13        b.compact();
14        b.put((byte) '!');
15        b.flip();
16        b.get();
17        b.rewind();
18        b.clear();
19        System.out.println(c1 + "" + c2 + " " +
20        b);
21    }
22 }
```

ByteBuffer (capacity = 8) · yazma modu



4 bayt yazıldı; konum 4.

Adım 5 / 15

ÇIKTI

DEĞİŞKENLER

pos = 4

lim = 8

cap = 8

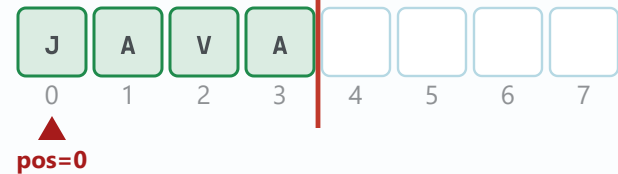
put, flip, get, compact, rewind, clear: Adım Adım

```

1 import java.nio.ByteBuffer;
2
3 public class TamponIzle {
4     public static void main(String[] args) {
5         ByteBuffer b = ByteBuffer.allocate(8);
6         b.put((byte) 'J');
7         b.put((byte) 'A');
8         b.put((byte) 'V');
9         b.put((byte) 'A');
10        ▶ b.flip();
11        char c1 = (char) b.get();
12        char c2 = (char) b.get();
13        b.compact();
14        b.put((byte) '!');
15        b.flip();
16        b.get();
17        b.rewind();
18        b.clear();
19        System.out.println(c1 + "" + c2 + " " +
20        b);
21    }
22 }

```

ByteBuffer (capacity = 8) · okuma modu



lim=4

flip() : sınır = eski konum (4), konum = 0. Yalnızca yazılan 4 bayt okunabilir.

Adım 6 / 15

ÇIKTI



DEĞİŞKENLER

pos = 0

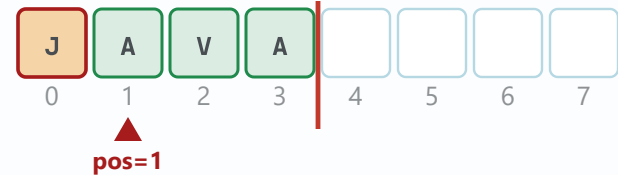
lim = 4

cap = 8

put, flip, get, compact, rewind, clear: Adım Adım

```
1 import java.nio.ByteBuffer;
2
3 public class TamponIzle {
4     public static void main(String[] args) {
5         ByteBuffer b = ByteBuffer.allocate(8);
6         b.put((byte) 'J');
7         b.put((byte) 'A');
8         b.put((byte) 'V');
9         b.put((byte) 'A');
10        b.flip();
11        ► char c1 = (char) b.get();
12        char c2 = (char) b.get();
13        b.compact();
14        b.put((byte) '!');
15        b.flip();
16        b.get();
17        b.rewind();
18        b.clear();
19        System.out.println(c1 + "" + c2 + " " +
20        b);
21    }
22 }
```

ByteBuffer (capacity = 8) · okuma modu



lim=4

get() konumdaki baytı döndürür, konum 1 olur.

Adım 7 / 15

ÇIKTI

DEĞİŞKENLER

pos = 1

lim = 4

cap = 8

c1 = 'J'

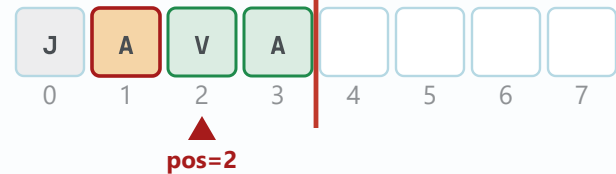
put, flip, get, compact, rewind, clear: Adım Adım

```

1 import java.nio.ByteBuffer;
2
3 public class TamponIzle {
4     public static void main(String[] args) {
5         ByteBuffer b = ByteBuffer.allocate(8);
6         b.put((byte) 'J');
7         b.put((byte) 'A');
8         b.put((byte) 'V');
9         b.put((byte) 'A');
10        b.flip();
11        char c1 = (char) b.get();
12        ► char c2 = (char) b.get();
13        b.compact();
14        b.put((byte) '!');
15        b.flip();
16        b.get();
17        b.rewind();
18        b.clear();
19        System.out.println(c1 + "" + c2 + " " +
20        b);
21    }
22 }

```

ByteBuffer (capacity = 8) · okuma modu



c2 = 'A'; okunmamış 2 bayt kaldı (V, A).

Adım 8 / 15

ÇIKTI



DEĞİŞKENLER

pos = 2

lim = 4

cap = 8

c1 = 'J'

c2 = 'A'

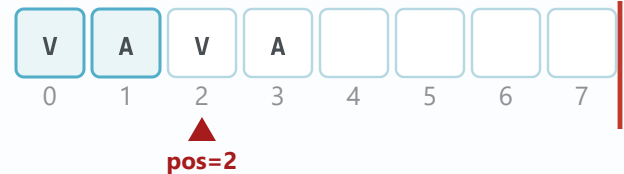
put, flip, get, compact, rewind, clear: Adım Adım

```

1 import java.nio.ByteBuffer;
2
3 public class TamponIzle {
4     public static void main(String[] args) {
5         ByteBuffer b = ByteBuffer.allocate(8);
6         b.put((byte) 'J');
7         b.put((byte) 'A');
8         b.put((byte) 'V');
9         b.put((byte) 'A');
10        b.flip();
11        char c1 = (char) b.get();
12        char c2 = (char) b.get();
13        ▶ b.compact();
14        b.put((byte) '!');
15        b.flip();
16        b.get();
17        b.rewind();
18        b.clear();
19        System.out.println(c1 + "" + c2 + " " +
20        b);
21    }
22 }

```

ByteBuffer (capacity = 8) · yazma modu



compact() : okunmamış V, A başa kopyalanır; konum 2, sınır 8. 2–3'teki eski baytlar silinmez, üzerine yazılacak.

Adım 9 / 15

ÇIKTI



DEĞİŞKENLER

pos = 2

lim = 8

cap = 8

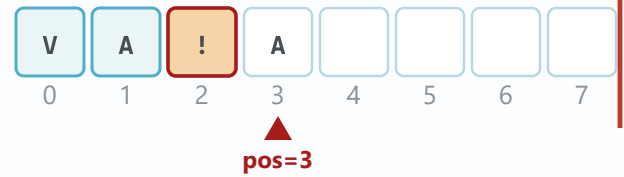
c1 = 'J'

c2 = 'A'

put, flip, get, compact, rewind, clear: Adım Adım

```
1 import java.nio.ByteBuffer;
2
3 public class TamponIzle {
4     public static void main(String[] args) {
5         ByteBuffer b = ByteBuffer.allocate(8);
6         b.put((byte) 'J');
7         b.put((byte) 'A');
8         b.put((byte) 'V');
9         b.put((byte) 'A');
10        b.flip();
11        char c1 = (char) b.get();
12        char c2 = (char) b.get();
13        b.compact();
14        ▶ b.put((byte) '!');
15        b.flip();
16        b.get();
17        b.rewind();
18        b.clear();
19        System.out.println(c1 + "" + c2 + " " +
20        b);
21    }
```

ByteBuffer (capacity = 8) · yazma modu



'!' konum 2'ye yazılır; eski 'V'nin üzerine.

Adım 10 / 15

ÇIKTI

DEĞİŞKENLER

pos = 3

lim = 8

cap = 8

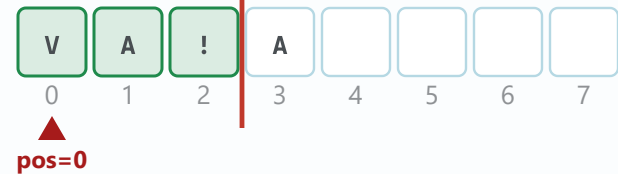
c1 = 'J'

c2 = 'A'

put, flip, get, compact, rewind, clear: Adım Adım

```
1 import java.nio.ByteBuffer;
2
3 public class TamponIzle {
4     public static void main(String[] args) {
5         ByteBuffer b = ByteBuffer.allocate(8);
6         b.put((byte) 'J');
7         b.put((byte) 'A');
8         b.put((byte) 'V');
9         b.put((byte) 'A');
10        b.flip();
11        char c1 = (char) b.get();
12        char c2 = (char) b.get();
13        b.compact();
14        b.put((byte) '!');
15        ▶ b.flip();
16        b.get();
17        b.rewind();
18        b.clear();
19        System.out.println(c1 + "" + c2 + " " +
20        b);
21    }
```

ByteBuffer (capacity = 8) · okuma modu



lim=3

Okumaya geç: sınır 3 (V, A, !).

Adım 11 / 15

ÇIKTI



DEĞİŞKENLER

pos = 0

lim = 3

cap = 8

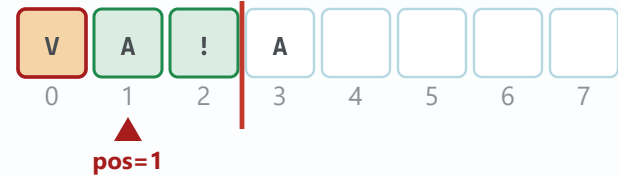
c1 = 'J'

c2 = 'A'

put, flip, get, compact, rewind, clear: Adım Adım

```
1 import java.nio.ByteBuffer;
2
3 public class TamponIzle {
4     public static void main(String[] args) {
5         ByteBuffer b = ByteBuffer.allocate(8);
6         b.put((byte) 'J');
7         b.put((byte) 'A');
8         b.put((byte) 'V');
9         b.put((byte) 'A');
10        b.flip();
11        char c1 = (char) b.get();
12        char c2 = (char) b.get();
13        b.compact();
14        b.put((byte) '!');
15        b.flip();
16        ▶ b.get();
17        b.rewind();
18        b.clear();
19        System.out.println(c1 + "" + c2 + " " +
20        b);
21    }
```

ByteBuffer (capacity = 8) · okuma modu



Bir bayt okunur ('V'), konum 1.

Adım 12 / 15

ÇIKTI



DEĞİŞKENLER

pos = 1

lim = 3

cap = 8

c1 = 'J'

c2 = 'A'

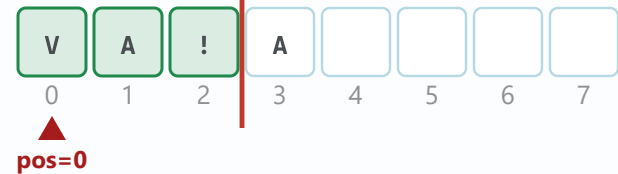
put, flip, get, compact, rewind, clear: Adım Adım

```

1 import java.nio.ByteBuffer;
2
3 public class TamponIzle {
4     public static void main(String[] args) {
5         ByteBuffer b = ByteBuffer.allocate(8);
6         b.put((byte) 'J');
7         b.put((byte) 'A');
8         b.put((byte) 'V');
9         b.put((byte) 'A');
10        b.flip();
11        char c1 = (char) b.get();
12        char c2 = (char) b.get();
13        b.compact();
14        b.put((byte) '!');
15        b.flip();
16        b.get();
17        ► b.rewind();
18        b.clear();
19        System.out.println(c1 + "" + c2 + " " +
20        b);
21    }
22 }

```

ByteBuffer (capacity = 8) · okuma modu



lim=3

`rewind()` : konum 0, sınır **değişmez**: aynı veri baştan yeniden okunabilir.

Adım 13 / 15

ÇIKTI



DEĞİŞKENLER

pos = 0

lim = 3

cap = 8

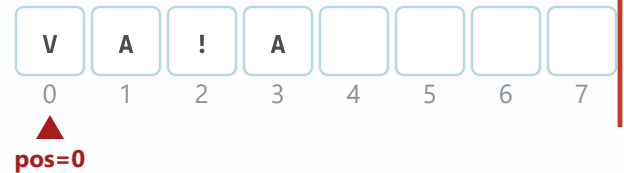
c1 = 'J'

c2 = 'A'

put, flip, get, compact, rewind, clear: Adım Adım

```
1 import java.nio.ByteBuffer;
2
3 public class TamponIzle {
4     public static void main(String[] args) {
5         ByteBuffer b = ByteBuffer.allocate(8);
6         b.put((byte) 'J');
7         b.put((byte) 'A');
8         b.put((byte) 'V');
9         b.put((byte) 'A');
10        b.flip();
11        char c1 = (char) b.get();
12        char c2 = (char) b.get();
13        b.compact();
14        b.put((byte) '!');
15        b.flip();
16        b.get();
17        b.rewind();
18        ▶ b.clear();
19        System.out.println(c1 + "" + c2 + " " +
20        b);
21    }
22 }
```

ByteBuffer (capacity = 8) · yazma modu



clear() : konum 0, sınır 8. Veri **silinmez**; sadece "boş" sayılır, üzerine yazılır.

Adım 14 / 15

ÇIKTI



DEĞİŞKENLER

pos = 0

lim = 8

cap = 8

c1 = 'J'

c2 = 'A'

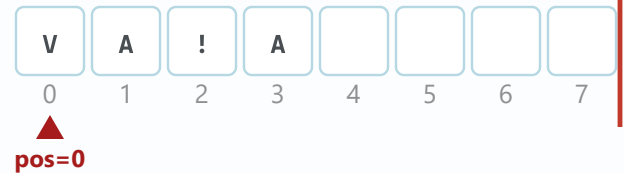
put, flip, get, compact, rewind, clear: Adım Adım

```

1 import java.nio.ByteBuffer;
2
3 public class TamponIzle {
4     public static void main(String[] args) {
5         ByteBuffer b = ByteBuffer.allocate(8);
6         b.put((byte) 'J');
7         b.put((byte) 'A');
8         b.put((byte) 'V');
9         b.put((byte) 'A');
10        b.flip();
11        char c1 = (char) b.get();
12        char c2 = (char) b.get();
13        b.compact();
14        b.put((byte) '!');
15        b.flip();
16        b.get();
17        b.rewind();
18        b.clear();
19        System.out.println(c1 + "" + c2 + " " +
20        b);
21    }
22 }

```

ByteBuffer (capacity = 8) · yazma modu



`toString()` tamponun konum, sınır ve kapasitesini gösterir.

Adım 15 / 15

ÇIKTI

```
JA java.nio.HeapByteBuffer[pos=0 lim=8
cap=8]
```

DEĞİŞKENLER

pos = 0 lim = 8 cap = 8 c1 = 'J'

c2 = 'A'

Tampon İşlemleri

Metot	Ne yapar?	position	limit
<code>put(x)</code>	konuma veri yazar	+1	—
<code>get()</code>	konumdaki veriyi okur	+1	—
<code>flip()</code>	yazmadan okumaya geçer	0	eski position
<code>rewind()</code>	aynı veriyi baştan yeniden okumak için	0	değişmez
<code>clear()</code>	yeniden yazmaya hazırlar; veriyi silmez	0	capacity
<code>compact()</code>	okunmamış veriyi başa taşır, yazmaya devam eder	kalan sayısı	capacity
<code>mark()</code> / <code>reset()</code>	konumu işaretler / işarete döner	— / mark	—

Avantajlar

- + Veri üzerinde esnek gezinme ve değiştirme
- + Verimli bellek yönetimi: tek alan tekrar tekrar kullanılır

Dezavantajlar

- Kullanımı karmaşıktır (`flip` unutmak sık hata)
- Boyutu sonradan değiştirilemez

Deneyin: ByteBuffer

bayt:

J

put

get

flip

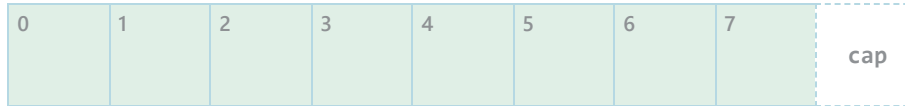
rewind

compact

clear

yeni tampon

```
ByteBuffer b = ByteBuffer.allocate(8);
```

**pos****lim****position=0 limit=8 capacity=8 remaining=8**

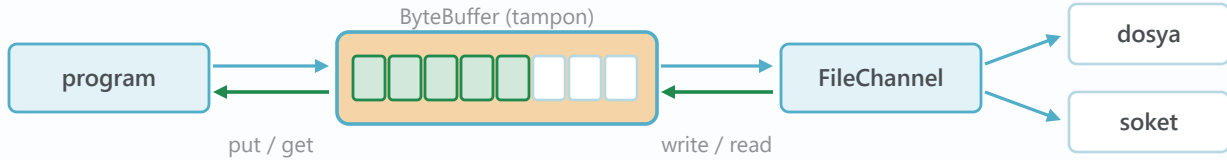
Boş tampon: yazma modunda, position = 0, limit = capacity = 8.

Yeşil hücreler **position ile limit arasındır**: yazma modunda boş yer, okuma modunda

okunacak veri. Deneyin: üç kez **put**, **flip**, iki kez **get**, **compact**. Ya da **flip** etmeden **get** deyin.

NIO Kanalları (Channel)

- Kanal, giriş ve çıkış verilerini yönetir: bir dosya, soket veya belirli bir protokol üzerinden veri alışverişi sağlar.
- Veri her zaman bir **tampon** üzerinden okunur (`read(tampon)`) veya yazılır (`write(tampon)`).



Kanal türü	Kullanımı
<code>FileChannel</code>	dosyada okuma ve yazma
<code>SocketChannel</code>	TCP ağ soketi üzerinden veri iletimi
<code>ServerSocketChannel</code>	gelen TCP bağlantılarını kabul etme
<code>DatagramChannel</code>	UDP soketi üzerinden veri iletimi

Kanal Oluşturma ve Kapatma

```
public class KanalAcKapa {  
    public static void main(String[] args) throws IOException {  
        // FileChannel oluşturma  
        FileChannel fileChannel = FileChannel.open(  
            Paths.get("dosya.txt"), StandardOpenOption.READ);  
        // SocketChannel oluşturma  
        SocketChannel socketChannel = SocketChannel.open();  
        socketChannel.connect(new InetSocketAddress("example.com", 80));  
        System.out.println("Bağlandı: " + socketChannel.isConnected());  
        // Kanal kapatma  
        fileChannel.close();  
        socketChannel.close();  
    }  
}
```

ÇIKTI

Bağlandı: true

`FileChannel.open(yol, seçenekler...)` dosya kanalı, `SocketChannel.open()` + `connect` ağ kanalı açar. Kanallar da `close()` ile (veya try-with-resources ile) kapatılmalıdır.

FileChannel ile Okuma

```
public class KanalOku {  
    public static void main(String[] args) throws IOException {  
        try (FileChannel channel = FileChannel.open(  
            Paths.get("dosya.txt"), StandardOpenOption.READ)) {  
            ByteBuffer buffer = ByteBuffer.allocate(1024);  
            // Dosyadan veriyi okuma  
            int bytesRead = channel.read(buffer);  
            System.out.println(bytesRead + " byte okundu");  
            // Buffer'ı okuma moduna geçirme  
            buffer.flip();  
            // Veriyi ekrana yazma  
            while (buffer.hasRemaining()) {  
                System.out.print((char) buffer.get());  
            }  
        }  
    }  
}
```

ÇIKTI

```
17 byte okundu  
NIO kanal ornegi
```

`channel.read(buffer)` tampona okur, okunan bayt sayısını (dosya sonunda -1) döndürür. Okumadan önce `flip()` unutulmamalı.

FileChannel ile Yazma

```
public class KanalYaz {  
    public static void main(String[] args) throws IOException {  
        try (FileChannel kanal = FileChannel.open(Paths.get("cikti.txt"),  
            StandardOpenOption.CREATE, StandardOpenOption.WRITE)) {  
            byte[] veri = "Kanal ile yazıldı\n".getBytes(StandardCharsets.UTF_8);  
            ByteBuffer tampon = ByteBuffer.wrap(veri);  
            int yazilan = kanal.write(tampon);  
            System.out.println(yazilan + " byte yazıldı");  
        }  
    }  
}
```

ÇIKTI

20 byte yazıldı

`ByteBuffer.wrap(dizi)` hazır bir diziyi tampon olarak kullanır (konum 0, sınır = dizi boyu); bu yüzden `flip()` gerekmez. `CREATE` : dosya yoksa oluştur.

SocketChannel Kullanımı

```
try (SocketChannel socketChannel = SocketChannel.open()) {
    socketChannel.connect(new InetSocketAddress("example.com", 80));
    ByteBuffer buffer = ByteBuffer.allocate(1024);
    buffer.put(("GET / HTTP/1.1\r\nHost: example.com\r\n"
        + "Connection: close\r\n\r\n").getBytes(StandardCharsets.US_ASCII));
    buffer.flip();
    // Veriyi sokete yazma
    socketChannel.write(buffer);
    // Cevabı okuma
    buffer.clear();
    socketChannel.read(buffer);
    buffer.flip();
    String cevap = StandardCharsets.US_ASCII.decode(buffer).toString();
    System.out.println(cevap.substring(0, cevap.indexOf("\r\n")));
} catch (IOException e) {
    e.printStackTrace();
}
```

ÇIKTI

```
HTTP/1.1 200 OK
```

main gövdesi. Ağ bağlantısı da bir kanaldır: aynı **put** / **flip** / **write** kalıbı. HTTP/1.1 isteğinde **Host** başlığı zorunludur.

NIO: Avantajlar ve Dezavantajlar

Avantajlar

- + **Performans:** bloklar hâlinde, klasik G/Ç'den verimli aktarım
- + **Esneklik:** tampon ve kanallarla esnek veri işleme; çift yönlü kanallar
- + Dosya ve ağ için aynı yöntem
- + **Çoklu bağlantı:** seçiciyle birden çok kanal tek iş parçacığında; yalnızca etkin kanallar işlenir

Dezavantajlar

- **Karmaşıklık:** klasik G/Ç'ye göre daha zor
- Tampon durumunu ve hata durumlarını (yarım okuma, kopan bağlantı) yönetmek gerekir
- Seçici için öğrenme eğrisi: çok kanalın durumunu doğru izlemek gerekir

Kural: tek bir dosyayı satır satır okuyacaksanız `BufferedReader` veya `Files` yeterlidir. Kanallar, çok büyük dosyalar ve çok sayıda ağ bağlantısı için öne çıkar.

Path ve Files (NIO.2)

```
public class DosyalarNio {  
    public static void main(String[] args) throws IOException {  
        Path yol = Path.of("veri", "notlar.txt");        // Java 11+  
        Files.createDirectories(yol.getParent());  
        Files.writeString(yol, "Ali 70\nAyşe 85\nCan 90\n");  
        System.out.println(yol.getFileName() + ": " + Files.exists(yol));  
        System.out.println("Boyut: " + Files.size(yol) + " byte");  
        List<String> satirlar = Files.readAllLines(yol);  
        for (String satir : satirlar) {  
            System.out.println(satir.toUpperCase());  
        }  
        Files.write(yol, List.of("Deniz 60"), StandardOpenOption.APPEND);  
        System.out.println(Files.readAllLines(yol).size() + " satır");  
    }  
}
```

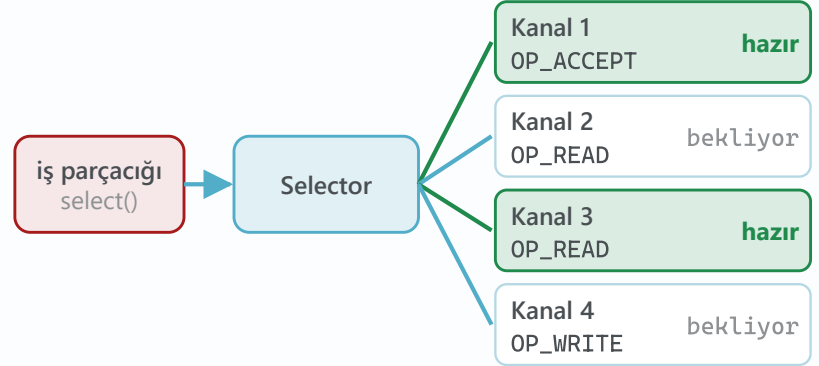
ÇIKTI

```
notlar.txt: true  
Boyut: 23 byte  
ALI 70  
AYŞE 85  
CAN 90  
4 satır
```

Metot	Görevi
<code>Path.of(...)</code> / <code>Paths.get(...)</code>	yol oluşturur (<code>File</code> 'in modern karşılığı)
<code>Files.readAllLines</code> , <code>Files.readString</code>	tüm dosyayı tek çağrıda okur (UTF-8)
<code>Files.write</code> , <code>Files.writeString</code>	tek çağrıda yazar
<code>Files.exists</code> , <code>size</code> , <code>copy</code> , <code>move</code> , <code>delete</code>	dosya sistemi işlemleri

NIO Seçici (Selector)

- Birden çok kanalı **tek bir iş parçacığı**nda izleme mekanizmasıdır.
- `Selector.open()` ile yeni bir seçici oluşturulur.
- Kanal bloklamayan moda alınır ve seçiciye **kaydedilir**:
`channel.register(selector, ops)`.
- Kanalın hangi olayları beklediği `SelectionKey` sabitleriyle belirlenir.
`select()` en az bir kanal hazır olana kadar bekler.



SelectionKey	Anlamı
OP_READ	kanaldan okuma yapılabilir
OP_WRITE	kanala yazma yapılabilir
OP_CONNECT	bağlantı kurma tamamlandı
OP_ACCEPT	bağlantı kabul edilebilir

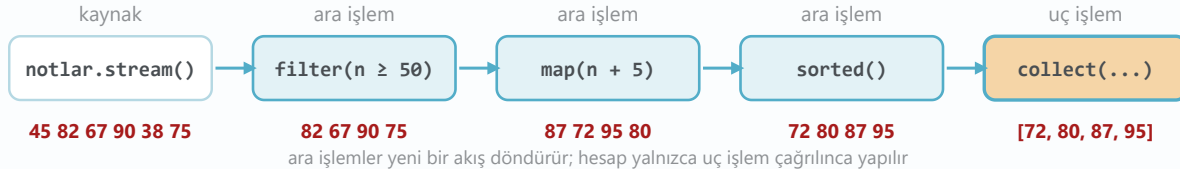
Seçici ile Sunucu

```
public class SeciciGoster {  
    public static void main(String[] args) throws IOException {  
        ServerSocketChannel kanal = ServerSocketChannel.open();  
        Selector secici = Selector.open();  
        kanal.bind(new InetSocketAddress(8080));  
        kanal.configureBlocking(false);  
        kanal.register(secici, SelectionKey.OP_ACCEPT);  
        while (true) {  
            int hazirKanallar = secici.select();  
            if (hazirKanallar > 0) {  
                Set<SelectionKey> secilenler = secici.selectedKeys();  
                for (SelectionKey secilen : secilenler) {  
                    if (secilen.isAcceptable()) {  
                        // Bağlantı kabul  
                    } else if (secilen.isReadable()) {  
                        // Veri okuma  
                    }  
                }  
                secilenler.clear(); // İşlenen anahtarları temizle  
            }  
        }  
    }  
}
```

- Sunucu kanalı 8080 portuna bağlanır, bloklamayan moda alınır, `OP_ACCEPT` için kaydedilir.
- `select()` hazır kanal sayısını, `selectedKeys()` hazır anahtarları verir.
- İşlenen anahtarlar `clear()` ile temizlenmezse sonraki turda yine gelir.
- Gerçek sunucuda kabul edilen istemci `OP_READ` ile kaydedilir ve veri okunur.

Akış API'si (`java.util.stream`)

- Java 8 ile gelen `java.util.stream` , bir veri dizisini **işlem hattı** (pipeline) olarak işler: “ne yapılacağını” söyleriz, döngüyü kütüphane kurar.
- Buradaki akış, `InputStream` gibi G/Ç akışlarından **farklıdır**: veri saklamaz, bir kaynağın (liste, dizi, dosya satırları) üzerinden geçer ve kaynağı değiştirmez.



```
List<Integer> notlar = Arrays.asList(45, 82, 67, 90, 38, 75);
List<Integer> gecenler = notlar.stream()
    .filter(n -> n >= 50)
    .map(n -> n + 5)
    .sorted()
    .collect(Collectors.toList());
System.out.println(gecenler);
```

ÇIKTI

```
[72, 80, 87, 95]
```

Java 16+ sürümünde son satır kısaca `.toList()` yazılabilir.

Gereken importlar:

```
java.util.* ,
```

```
java.util.stream.Collectors
```

Ara ve Uç İşlemler

Ara işlem	Görevi
<code>filter(p)</code>	koşulu sağlayanları geçirir
<code>map(f)</code>	her öğeyi dönüştürür
<code>flatMap(f)</code>	iç içe akışları düzleştirir
<code>sorted()</code>	sıralar (tüm öğeleri bekler)
<code>distinct()</code>	tekrarları atar (<code>equals</code>)
<code>limit(n)</code> / <code>skip(n)</code>	ilk <i>n</i> öğeyi alır / atlar
<code>peek(f)</code>	öğeleri izler (hata ayıklama)

Uç işlem	Sonuç
<code>collect(...)</code>	liste, küme, eşlem...
<code>forEach(f)</code>	her öğe için bir eylem
<code>count()</code>	öge sayısı (<code>long</code>)
<code>reduce(...)</code>	öğeleri tek değere indirger
<code>findFirst()</code>	ilk öge (<code>Optional</code>)
<code>anyMatch</code> / <code>allMatch</code> / <code>noneMatch</code>	<code>boolean</code>
<code>min</code> / <code>max</code>	<code>Optional</code>

Ara işlem **yeni bir akış** döndürür ve zincir sürer; uç işlem akış dışında bir sonuç (veya hiçbir şey) döndürür ve hattı **çalıştırır**. Uç işlemi olmayan bir hat hiçbir şey yapmaz.

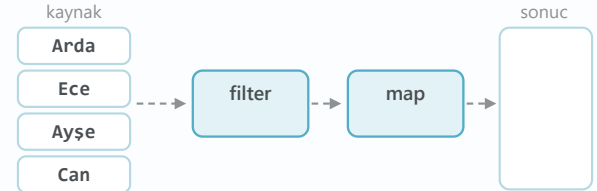
Tembel Değerlendirme ve Tek Kullanım

```
List<String> adlar =  
    Arrays.asList("Arda", "Ece", "Ayşe", "Can");  
Stream<String> hat = adlar.stream()  
    .filter(s -> s.startsWith("A"))  
    .map(String::toUpperCase);  
// Henüz hiçbir öge işlenmedi.  
List<String> sonuc =  
    hat.collect(Collectors.toList());  
System.out.println(sonuc); // [ARDA, AYŞE]  
hat.count(); // IllegalStateException
```

- **Tembel** (lazy): ara işlemler yalnızca bir **tarif** kaydeder; öğeler uç işlem çağrılınca işlenir.
- Öğeler hattan **tek tek** geçer: ilk öge **filter** ve **map** 'ten geçer, sonra ikincisi gelir.
- **findFirst**, **limit** sonucu bulunca durur; kalan öğeler hiç işlenmez.
- Akış **tek kullanımlıktır**: uç işlem den sonra yeniden kullanmak **IllegalStateException** fırlatır; gerekirse kaynaktan yeni akış alınır.

Tembel Akış Hattı: Adım Adım

```
▶ List<String> adlar = Arrays.asList("Arda", "Ece", "Ayşe",  
  "Can");  
▶ Stream<String> hat = adlar.stream()  
▶   .filter(s -> { System.out.println("filter " + s);  
▶     return s.startsWith("A"); })  
▶   .map(s -> { System.out.println("map " + s);  
▶     return s.toUpperCase(); });  
7 System.out.println("hat hazır");  
8 List<String> sonuc = hat.collect(Collectors.toList());  
9 System.out.println(sonuc);
```



Hat kuruldu: **filter** ve **map** yalnızca kaydedildi.
Hiçbir lambda çalışmadı, çıktı boş.

Adım 1 / 10

DEĞİŞKENLER

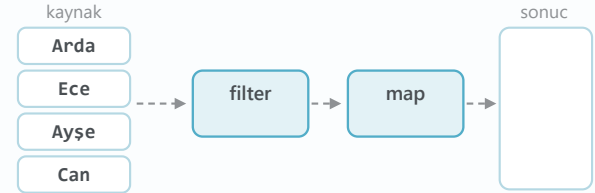
adlar = [Arda, Ece, Ayşe, Can]

hat = **Stream**

ÇIKTI

Tembel Akış Hattı: Adım Adım

```
1 List<String> adlar = Arrays.asList("Arda", "Ece", "Ayşe",  
  "Can");  
2 Stream<String> hat = adlar.stream()  
3   .filter(s -> { System.out.println("filter " + s);  
4     return s.startsWith("A"); })  
5   .map(s -> { System.out.println("map " + s);  
6     return s.toUpperCase(); });  
▶ System.out.println("hat hazır");  
8 List<String> sonuc = hat.collect(Collectors.toList());  
9 System.out.println(sonuc);
```



Bu satır, işlemlerden **önce** yazılır: ara işlemler tembeldir.

Adım 2 / 10

DEĞİŞKENLER

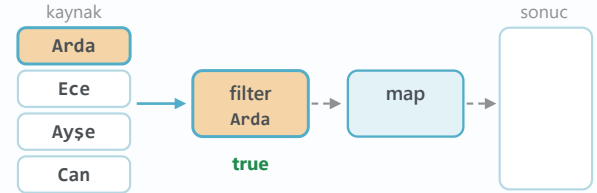
hat = **Stream**

ÇIKTI

hat hazır

Tembel Akış Hattı: Adım Adım

```
1 List<String> adlar = Arrays.asList("Arda", "Ece", "Ayşe",  
  "Can");  
2 Stream<String> hat = adlar.stream()  
▶ .filter(s -> { System.out.println("filter " + s);  
▶   return s.startsWith("A"); })  
5   .map(s -> { System.out.println("map " + s);  
6     return s.toUpperCase(); });  
7 System.out.println("hat hazır");  
▶ List<String> sonuc = hat.collect(Collectors.toList());  
9 System.out.println(sonuc);
```



`collect` hattı çalıştırır. İlk öge `Arda` `filter` 'a girer: "A" ile başlıyor → geçer.

Adım 3 / 10

DEĞİŞKENLER

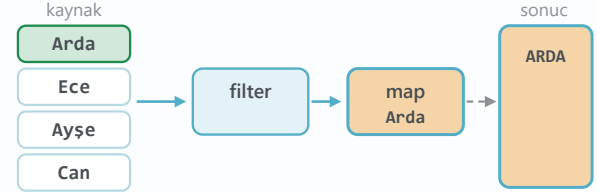
`s = "Arda"`

ÇIKTI

```
hat hazır  
filter Arda
```

Tembel Akış Hattı: Adım Adım

```
1 List<String> adlar = Arrays.asList("Arda", "Ece", "Ayşe",  
  "Can");  
2 Stream<String> hat = adlar.stream()  
3   .filter(s -> { System.out.println("filter " + s);  
4     return s.startsWith("A"); })  
5   .map(s -> { System.out.println("map " + s);  
6     return s.toUpperCase(); });  
7 System.out.println("hat hazır");  
8 List<String> sonuc = hat.collect(Collectors.toList());  
9 System.out.println(sonuc);
```



Aynı öğe hemen **map** 'e geçer ve **ARDA** sonuca eklenir. İkinci öğeye ancak şimdi sıra gelir.

Adım 4 / 10

DEĞİŞKENLER

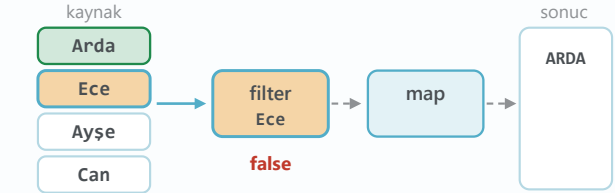
s = "Arda"

ÇIKTI

```
hat hazır  
filter Arda  
map Arda
```

Tembel Akış Hattı: Adım Adım

```
1 List<String> adlar = Arrays.asList("Arda", "Ece", "Ayşe",  
  "Can");  
2 Stream<String> hat = adlar.stream()  
▶ .filter(s -> { System.out.println("filter " + s);  
  ▶      return s.startsWith("A"); })  
5   .map(s -> { System.out.println("map " + s);  
6     return s.toUpperCase(); });  
7 System.out.println("hat hazır");  
▶ List<String> sonuc = hat.collect(Collectors.toList());  
9 System.out.println(sonuc);
```



Ece koşulu sağlamaz: elenir, map'e hiç ulaşmaz.

Adım 5 / 10

DEĞİŞKENLER

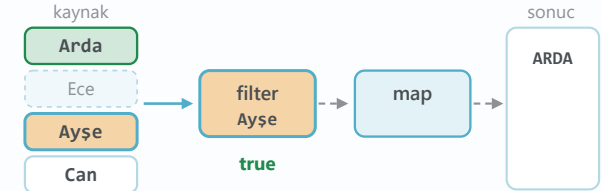
s = "Ece"

ÇIKTI

```
hat hazır
filter Arda
map Arda
filter Ece
```

Tembel Akış Hattı: Adım Adım

```
1 List<String> adlar = Arrays.asList("Arda", "Ece", "Ayşe",  
  "Can");  
2 Stream<String> hat = adlar.stream()  
▶ .filter(s -> { System.out.println("filter " + s);  
▶   return s.startsWith("A"); })  
5   .map(s -> { System.out.println("map " + s);  
6     return s.toUpperCase(); });  
7 System.out.println("hat hazır");  
▶ List<String> sonuc = hat.collect(Collectors.toList());  
9 System.out.println(sonuc);
```



Ayşe filtreyi geçer.

Adım 6 / 10

DEĞİŞKENLER

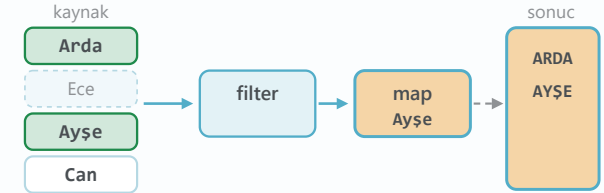
s = "Ayşe"

ÇIKTI

```
hat hazır  
filter Arda  
map Arda  
filter Ece  
filter Ayşe
```

Tembel Akış Hattı: Adım Adım

```
1 List<String> adlar = Arrays.asList("Arda", "Ece", "Ayşe",  
  "Can");  
2 Stream<String> hat = adlar.stream()  
3   .filter(s -> { System.out.println("filter " + s);  
4     return s.startsWith("A"); })  
5   .map(s -> { System.out.println("map " + s);  
6     return s.toUpperCase(); });  
7 System.out.println("hat hazır");  
8 List<String> sonuc = hat.collect(Collectors.toList());  
9 System.out.println(sonuc);
```



map büyük harfe çevirir: AYŞE sonuca eklenir.

Adım 7 / 10

DEĞİŞKENLER

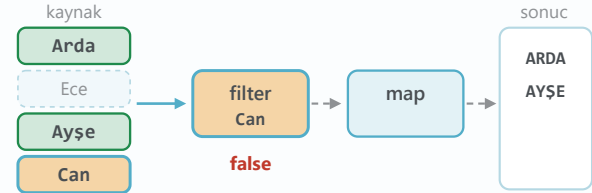
s = "Ayşe"

ÇIKTI

```
hat hazır  
filter Arda  
map Arda  
filter Ece  
filter Ayşe  
map Ayşe
```

Tembel Akış Hattı: Adım Adım

```
1 List<String> adlar = Arrays.asList("Arda", "Ece", "Ayşe",  
  "Can");  
2 Stream<String> hat = adlar.stream()  
▶ .filter(s -> { System.out.println("filter " + s);  
  ▶      return s.startsWith("A"); })  
5   .map(s -> { System.out.println("map " + s);  
6     return s.toUpperCase(); });  
7 System.out.println("hat hazır");  
▶ List<String> sonuc = hat.collect(Collectors.toList());  
9 System.out.println(sonuc);
```



Can elenir. Kaynakta öğe kalmadı.

Adım 8 / 10

DEĞİŞKENLER

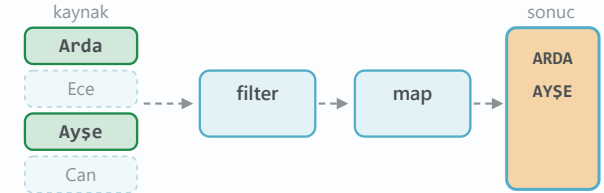
s = "Can"

ÇIKTI

```
hat hazır
filter Arda
map Arda
filter Ece
filter Ayşe
map Ayşe
filter Can
```

Tembel Akış Hattı: Adım Adım

```
1 List<String> adlar = Arrays.asList("Arda", "Ece", "Ayşe",  
  "Can");  
2 Stream<String> hat = adlar.stream()  
3   .filter(s -> { System.out.println("filter " + s);  
4     return s.startsWith("A"); })  
5   .map(s -> { System.out.println("map " + s);  
6     return s.toUpperCase(); });  
7 System.out.println("hat hazır");  
▶ List<String> sonuc = hat.collect(Collectors.toList());  
9 System.out.println(sonuc);
```



collect listeyi döndürür. **filter** 4 kez, **map** yalnızca 2 kez çalıştı.

Adım 9 / 10

DEĞİŞKENLER

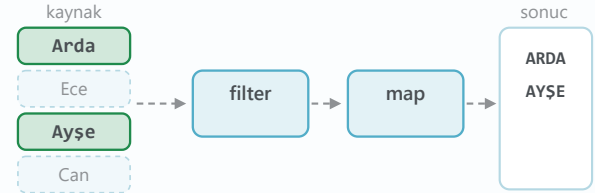
sonuc = [ARDA, AYŞE]

ÇIKTI

```
hat hazır
filter Arda
map Arda
filter Ece
filter Ayşe
map Ayşe
filter Can
```

Tembel Akış Hattı: Adım Adım

```
1 List<String> adlar = Arrays.asList("Arda", "Ece", "Ayşe",  
  "Can");  
2 Stream<String> hat = adlar.stream()  
3   .filter(s -> { System.out.println("filter " + s);  
4     return s.startsWith("A"); })  
5   .map(s -> { System.out.println("map " + s);  
6     return s.toUpperCase(); });  
7 System.out.println("hat hazır");  
8 List<String> sonuc = hat.collect(Collectors.toList());  
► System.out.println(sonuc);
```



Sonuç yazılır. Çıktının sırası, öğelerin hattan **tek tek** geçtiğini gösterir.

Adım 10 / 10

DEĞİŞKENLER

sonuc = [ARDA, AYŞE]

ÇIKTI

```
hat hazır
filter Arda
map Arda
filter Ece
filter Ayşe
map Ayşe
filter Can
[ARDA, AYŞE]
```

reduce, Collectors ve İlkel Akışlar

```
List<String> meyveler = Arrays.asList(
    "elma", "kivi", "muz", "erik", "incir");
int toplam = Arrays.asList(4, 8, 15, 16, 23)
    .stream().reduce(0, Integer::sum);
Map<Integer, List<String>> boy = meyveler
    .stream().collect(
        Collectors.groupingBy(String::length));
String hepsi = meyveler.stream()
    .collect(Collectors.joining(", "));
int t = IntStream.rangeClosed(1, 100).sum();
Optional<String> z = meyveler.stream()
    .filter(m -> m.startsWith("z")).findFirst();
System.out.println(z.orElse("yok"));
```

DEĞERLER

```
toplam = 66
boy = {3=[muz],
      4=[elma, kivi, erik],
      5=[incir]}
hepsi = elma, kivi, muz,
        erik, incir
t = 5050
yok
```

- **reduce** : $0 + 4 + 8 + 15 + 16 + 23$.
- **groupingBy** gruplar, **joining** birleştirir.
- **IntStream** : kutulamasız sayı akışı; **sum** , **average** hazır.
- **Optional** : sonuç olmayabilir; **orElse** , **ifPresent** .

Yan Etkiler ve Paralel Akışlar

```
List<Integer> liste = new ArrayList<>();
IntStream.range(0, 10000).parallel()
    .forEach(liste::add); // YANLIŞ: paylaşılan liste

List<Integer> dogru = IntStream.range(0, 10000).parallel()
    .boxed()
    .collect(Collectors.toList()); // DOĞRU: 10000 öge
```

BİRİNCİ HATTIN SONUCU (HER ÇALIŞTIRMADA FARKLI)

```
ArrayIndexOutOfBoundsException
ya da 10000'den az öge
```

- Lambda'lar dış değişkenleri **değiştirmemeli**; sonuç `map`, `collect`, `reduce` ile üretilmeli.
- `parallel()` / `parallelStream()` işi ortak `ForkJoinPool` iş parçacıklarına böler. `ArrayList` iş parçacığı güvenli olmadığı için ilk hat bozulur.
- Paralel akış **her zaman hızlı değildir**: küçük veri, ucuz işlemler, G/Ç bekleyen işler ve sıraya bağlı işlemler (`sorted`, `findFirst`) yavaşlatabilir. Önce ölçün.

Sık hatalar: akışı ikinci kez kullanmak; asıl işi `peek` içinde yapmak; `forEach` ile dış koleksiyon doldurmak; paralel akışın kendiliğinden hızlı olduğunu varsaymak.

ÖZET

Özet

Konu	Anahtar sınıf / metot	Hatırlanacak
Akışlar	<code>System.in</code> , <code>System.out</code> , <code>System.err</code> , <code>Scanner</code>	veri kaynaktan hedefe sırayla akar
Metin / ikili	ASCII, ISO-8859-9, UTF-8; <code>DataOutputStream</code>	127: metin 3 bayt, int 4 bayt; aynı kodlamayla oku-yaz
Dosya sistemi	<code>File</code> , <code>Path</code> , <code>Files</code>	<code>File</code> yalnızca yoldur, içeriği okumaz
Metin okuma/yazma	<code>Scanner</code> , <code>BufferedReader</code> , <code>PrintWriter</code> , <code>FileWriter</code>	<code>readLine()</code> sonda <code>null</code> ; <code>FileWriter(ad, true)</code> ekler
Bayt akışları	<code>FileInputStream</code> , <code>ByteArray...</code> , <code>Piped...</code> , <code>Sequence...</code>	<code>read()</code> sonda <code>-1</code> ; okunan <code>int</code> 'te tutulur
Tamponlama	<code>Buffered...</code> , <code>flush()</code>	disk erişimini toplu yapar; kapatmayı unutma
Kopyalama	<code>read(byte[])</code> , <code>write(b, 0, n)</code> , <code>Files.copy</code>	yalnızca okunan <code>n</code> bayt yazılır
Rastgele erişim	<code>RandomAccessFile</code> , <code>seek</code>	$k. \text{ kayit} = k \times \text{ kayit boyu}$
Serileştirme	<code>Serializable</code> , <code>ObjectOutputStream</code> , <code>transient</code>	<code>static</code> / <code>transient</code> yazılmaz
NIO	<code>ByteBuffer</code> , <code>FileChannel</code> , <code>Selector</code>	yazdıktan sonra <code>flip()</code> , tekrar yazmadan önce <code>clear()</code> / <code>compact()</code>

Sık Yapılan Hatalar (1)

Kontrol edilen istisnayı ele almamak

error: unreported exception

FileNotFoundException;

must be caught or declared to be thrown

```
new FileReader("a.txt") try - catch
```

içinde olmalı ya da metot throws IOException demeli.

import unutmak

error: cannot find symbol

symbol: class Scanner

```
import java.util.Scanner; (G/Ç sınıfları için  
import java.io.*; ).
```

Windows yolunda tek ters bölü

error: illegal escape character

```
new File("C:\\dersler\\notlar.txt")
```

\n , \d kaçış dizisi sayılır.

"C:\\dersler\\notlar.txt" veya

"C:/dersler/notlar.txt" yazın.

finally'de null kontrolü yok

Exception in thread "main"

java.lang.NullPointerException:

Cannot invoke "java.util.Scanner.close()"

Dosya açılmayınca scanner null kalır. if
(scanner != null) ya da try-with-resources
kullanın.

Sık Yapılan Hatalar (2)

nextInt'ten sonra nextLine

Yaş: 19 Ad: Ayşe → çıktı: [] 19

`nextInt()` satır sonunu tamponda bırakır;
ardından gelen `nextLine()` boş satırı okur. Araya
bir `klavye.nextLine();` ekleyin.

flip() unutmak

`b.put('J');` `b.put('A');`
`b.get()` → 0 (konum 2'deki boş hücre)

Yazdıktan sonra okumadan önce `flip()` şarttır;
yoksa yazılan veri değil, sonrası okunur.

read()'i byte'ta tutmak

`byte b; while ((b = (byte) in.read()) != -1)`
255 değerli byte (-1'e dönüşür) dosya sonu sanılır.
`int` kullanın.

Tamponlu yazıcıyı kapatmamak

`bw.write("kayıt");` // close yok
`not.txt boyutu: 0`

Veri tamponda kalır, dosya boş görünür. `close()`
(veya `flush()`) çağırın; en iyisi try-with-resources.

Serializable unutmak

`java.io.NotSerializableException: Araba`

`writeObject` edilen sınıf (ve alanlarının tipleri)
`implements Serializable` olmalı.

write(parca) yazmak

`out.write(parca);` // son blokta eski veri
de gider

Son blokta dizi tam dolmaz; `write(parca, 0,`
`boyut)` yazın.

ÖZET

Örnek Problemler (1)

```
try (PrintWriter pw = new PrintWriter("s.txt")) {  
    pw.print(12);  
    pw.print(' ');  
    pw.print(345);  
}  
System.out.println(new File("s.txt").length());  
try (DataOutputStream d = new DataOutputStream(new FileOutputStream("s.dat"))) {  
    d.writeInt(12);  
    d.writeInt(345);  
}  
System.out.println(new File("s.dat").length());
```

1. Bu kod ne yazdırır? `12` ve `345` önce metin, sonra ikili olarak yazılıyor.

► Cevabı göster

ÖZET

Örnek Problemler (2)

```
try (FileWriter w = new FileWriter("a.txt")) {  
    w.write("ab");  
}  
try (FileWriter w = new FileWriter("a.txt")) {  
    w.write("cd");  
}  
try (FileWriter w = new FileWriter("a.txt", true)) {  
    w.write("ef");  
}  
try (BufferedReader r = new BufferedReader(new FileReader("a.txt"))) {  
    System.out.println(r.readLine());  
}
```

2. Bu kod ne yazdırır? `a.txt` başta yok.

► Cevabı göster

ÖZET

Örnek Problemler (3)

```
ByteBuffer b = ByteBuffer.allocate(10);  
b.put((byte) 1).put((byte) 2).put((byte) 3);  
b.flip();  
b.get();  
System.out.println(b.position() + " " + b.limit() + " " + b.remaining());  
b.compact();  
System.out.println(b.position() + " " + b.limit());  
b.clear();  
System.out.println(b.position() + " " + b.limit() + " " + b.get(0));
```

3. Bu kod üç satırda ne yazdırır?

► Cevabı göster

ÖZET

Örnek Problemler (4)

```
try (RandomAccessFile f = new RandomAccessFile("r.dat", "rw")) {  
    for (int i = 1; i <= 4; i++) f.writeInt(i * 10);  
    f.seek(4);  
    int a = f.readInt();  
    f.seek(f.length() - 4);  
    int b = f.readInt();  
    System.out.println(a + " " + b + " " + f.getFilePointer());  
}
```

4. Bu kod ne yazdırır?

► Cevabı göster

ÖZET

Alıştırmalar (1)

1. **Sayaç:** `metin.txt` dosyasının satır, kelime ve karakter sayısını (satır sonları hariç) bulan bir program yazın.
Dosya: "Java ile dosya / okuma ve yazma / (boş satır) / çok kolay".

► **Çözüm**

ÖZET

Alıştırmalar (2)

2. **Kareler:** klavyeden `n` okuyun; 1..`n` sayılarını ve karelerini `kareler.txt` 'ye yazın, sonra dosyayı okuyup kareler toplamını bulun.

► **Çözüm**

ÖZET

Alıştırmalar (3)

3. **Dizi saklama:** `int[] dizi = {3, 1, 4, 1, 5}` dizisini `dizi.ser` 'e serileştirin, geri okuyup öğe sayısını ve toplamını yazdırın (diziler serileştirilebilir).

► **Çözüm**

Alıştırmalar (4)

4. **Not dosyası:** 5 notu `RandomAccessFile` ile `int` olarak yazın. Verilen sıra numarasındaki notu dosyanın tamamını okumadan bulun. Hangi konuma `seek` edilmeli?
5. **Düşünün:** 10 000 satırlık bir kayıt dosyasının yalnızca son kaydını okumak için sıralı akış mı, rastgele erişim mi uygundur? Satır boyları farklıysa ne değişir?

► Cevaplar / ipuçları

Kaynaklar

- Oracle, *The Java Tutorials: "Basic I/O"* (I/O Streams, File I/O featuring NIO.2), "Scanning and Formatting".
- Oracle, *Java SE 21 API Documentation*: `java.io` , `java.nio` , `java.nio.channels` , `java.nio.file` paketleri.
- Oracle, *Java Object Serialization Specification*.
- J. Gosling ve diğ., *The Java Language Specification, Java SE 21 Edition* (3.10.6 kaçış dizileri, 14.20.3 try-with-resources).
- P. Deitel ve H. Deitel, *Java: How to Program, Early Objects*, 11. baskı, Pearson, 2017 (Bölüm 15: Files, Input/Output Streams, NIO and XML Serialization).
- C. S. Horstmann, *Core Java, Volume II: Advanced Features*, 12. baskı, 2022 (Bölüm 2: Input and Output).
- K. Sierra ve B. Bates, *Head First Java*, 3. baskı, O'Reilly, 2022 (Bölüm 16: Saving Objects (and Text)).
- A. B. Downey ve C. Mayfield, *Think Java*, 2. baskı, O'Reilly, 2019.

Son

Bölüm 7: Giriş Çıkış

Sonraki bölüm: Bölüm 8: Grafikler · [Terim sözlüğü](#)