Bölüm 7: Giriş Çıkış

JAVA ile Nesne Yönelimli Programlama

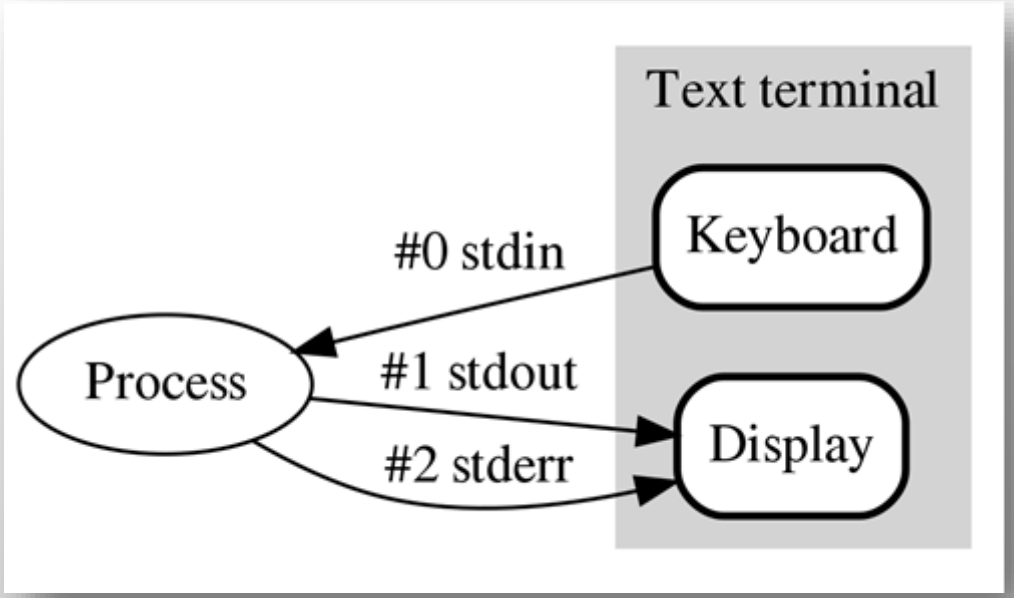
Akış (Stream)

- Akışlar, veri taşımak için temel bir yapı sağlar.
- Veri karakterler, sayılar veya ikili rakamlardan oluşan baytlar olabilir.
- Programa veri girişi ve çıkışı için kullanılır.
- **Girdi Akışı**: Verinin programa girdiği akışa denir. Örnek: *System.in*.
- **Çıktı Akışı**: Verinin programdan çıktığı akışa denir. Örnek: *System.out*.

```
// Giriş akışından veri okuma
Scanner scanner = new Scanner(System.in);
int kullaniciGirdisi = scanner.nextInt();
// Çıkış akışına veri yazma
System.out.println("Merhaba, Dünya!");
```

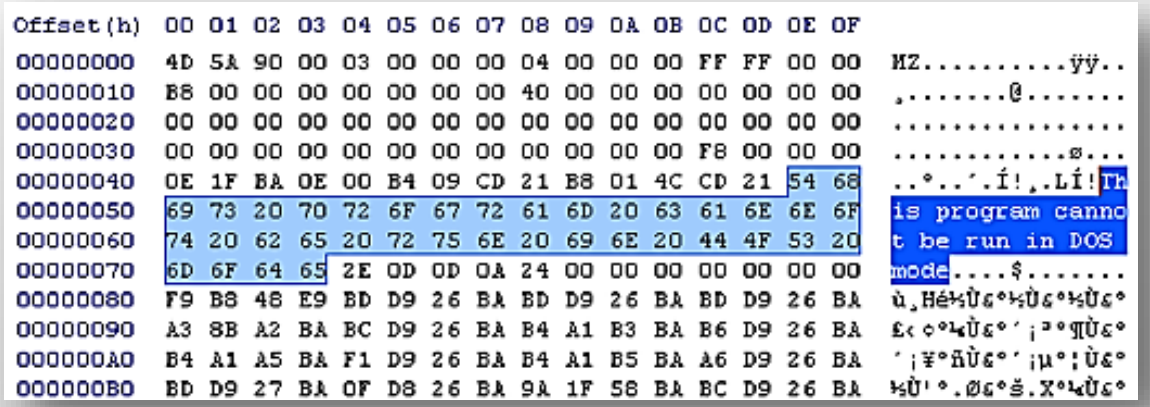
Klavye ve Ekran

- **Geçici Veri İşleme**: Klavye ve ekran, geçici veri üzerinde çalışır.
- **Veri akışı**: Anında giriş ve çıkış sağlar.



Dosyalar

- Veriyi kalıcı olarak saklar.
- Tüm dosyadaki veri, 0'lar ve 1'ler olarak saklanır.
- Dosya Türleri
  - **Metin** Dosyaları: İnsanlar tarafından okunabilir metin içerir.
  - **İkili** Dosyalar: 0 ve 1'lerden oluşan bit verilerini içerir.



Offset(h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	
00000000	4D	5A	90	00	03	00	00	00	04	00	00	00	FF	FF	00	00	MZ.....ÿÿ..
00000010	B8	00	00	00	00	00	00	00	40	00	00	00	00	00	00	00	¸.......@.....
00000020	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
00000030	00	00	00	00	00	00	00	00	00	00	00	00	F8	00	00	00	.....ø...
00000040	0E	1F	BA	0E	00	B4	09	CD	21	B8	01	4C	CD	21	54	68	..º..´.Í!¸.LÍ!Th
00000050	69	73	20	70	72	6F	67	72	61	6D	20	63	61	6E	6E	6F	is program canno
00000060	74	20	62	65	20	72	75	6E	20	69	6E	20	44	4F	53	20	t be run in DOS
00000070	6D	6F	64	65	2E	0D	0D	0A	24	00	00	00	00	00	00	00	mode....$.....
00000080	F9	B8	48	E9	BD	D9	26	BA	BD	D9	26	BA	BD	D9	26	BA	ù¸Hé½Ù&º½Ù&º½Ù&º
00000090	A3	8B	A2	BA	BC	D9	26	BA	B4	A1	B3	BA	B6	D9	26	BA	£‹¢º¼Ù&º´¡³º¶Ù&º
000000A0	B4	A1	A5	BA	F1	D9	26	BA	B4	A1	B5	BA	A6	D9	26	BA	´¡¥ºñÙ&º´¡µº¦Ù&º
000000B0	BD	D9	27	BA	0F	D8	26	BA	9A	1F	58	BA	BC	D9	26	BA	½Ù'º.Ø&ºš.Xº¼Ù&º

Dosyalar

```
// Metin dosyasına yazma
FileWriter writer = new FileWriter("dosya.txt");
writer.write("Merhaba, dünya!");
writer.close();

// İkili dosyadan okuma
FileInputStream stream = new FileInputStream("veri.dat");
int veri = stream.read();
stream.close();
```

Dosya İşleme Süreçleri

- **Oluşturma**: dosya oluşturulur.
- **Açma**: var olan dosya açılır.
- **Yazma:** veri dosyaya yazılır.
- **Okuma**: veri dosyadan okunur.
- **Kapatma**: dosya, işlem tamamlandığında kapatılır.

Metin Dosyaları

- Metin dosyaları, yazdırılabilir karakterleri içerir.
- Metin dosyaları metin düzenleyicileri kullanılarak düzenlenebilir.
- Kaynak kod dosyaları (.java, .c), Notepad.exe ile kaydedilen dosyalar.
- **Karakter Seti**: Karakterlerin bilgisayar tarafından anlaşılabilir formda kodlanmasını sağlar.
  - **ASCII**: Temel İngilizce karakterleri içerir.
  - **ISO-8859-1**: Latin alfabesi temel karakter kümesini içerir.
  - **UTF-8**: Evrensel karakter kümesini destekler, çok dilli metinleri kapsar.

İkili Dosyalar

- Bilgisayar tarafından yürütülebilen verileri içerir.
- Bilgisayar tarafından anlaşılabilir, insanlar tarafından doğrudan okunamaz.
- Yürütülebilir dosyalar, görüntü, müzik veya video dosyaları.

```
InitInstance  ^@    Self    @  Instance   / L]@ CleanupInstance      ^@
Self   ) |K  ClassType    @   ^@   Self   4 è[@         ClassName   .@
Self  @.@    9  \@
ClassNameIs    @      Self  .@   Name   + |\@
ClassParent    @      Self   ) `@       ClassInfo    @        Self   , |K
InstanceSize  œ@      Self   < ð_@
InheritsFrom    @      Self    @  AClass   ; a@
MethodAddress    @       Self  ä@   Name   ;  a@
MethodAddress    @       Self  .@   Name   F Èa@
MethodName   .@      Self    @   Address  @.@    = |K QualifiedClassName   .@
```

ASCII Tablosu

- ASCII (American Standard Code for Information Interchange): Bilgisayarlar arasında metin tabanlı verilerin değişimini standartlaştırmak için kullanılan bir karakter kodlama sistemidir.
- ASCII Karakterleri
  - **Temel** : harf, rakam, noktalama işaretleri ve temel semboller.
  - **Kontrol** : Bilgi iletimi sırasında özel işlevlere sahip karakterler.
- ASCII, her karakteri bir sayı ile temsil eder.
- Büyük harf "A" ASCII'de 65'e, küçük harf "a" ise 97'ye karşılık gelir.

Genişletilmiş ASCII Kodları

- Genişletilmiş ASCII (Extended ASCII): Standart ASCII karakter kümesini genişleterek, özel karakter ve sembollerin eklenmesini sağlayan bir karakter kodlama sistemidir.
- **Temel** Karakterler: Standart ASCII kümesinden alınan karakterler.
- **Özel** Karakterler: Ş, ç, ğ gibi Türkçe karakterler ve özel semboller.
- Genişletilmiş ASCII, 8-bit (1 byte) karakter temsilini kullanır.
- Bu, toplamda 256 farklı karakteri temsil etmeyi sağlar.

Bir Dosyaya Sayı (127) Yazma

- Sayısal verilerin saklanması, veri tipi, kullanım senaryosu ve dosya formatına bağlı olarak değişebilir.
- İkili dosyalar, sayısal verileri verimli bir şekilde saklar.
- ASCII Kodlu Metin Dosyası
  - Her karakter için üç *byte* kullanılır: 1, 2 ve 7.
  - Karakterlerin ikili değerleri: 00110001, 00110010, 00110111.
- İkili Dosya
  - Bir byte (byte): 01111111
  - İki byte (short): 00000000 01111111
  - Dört byte (int): 00000000 00000000 00000000 01111111

Bir Dosyaya Sayı Yazma

- Karşılaştırma
  - ASCII Metin Dosyası: 3 byte.
  - Unicode Metin Dosyası: 6 byte.
  - İkili Dosya (*byte*): 1 byte.
  - İkili Dosya (*short*): 2 byte.
  - İkili Dosya (*int*): 4 byte.
- Kullanım Alanları
  - ASCII, insanlar tarafından okunabilir veriler için kullanılır.
  - İkili dosyalar, sayısal verilerin daha verimli saklanması için kullanılır.

java.io.File Sınıfı

- Dosya sistemine erişmek ve ilgili işlemleri gerçekleştirmek için kullanılır.
  - Dosya yolu var mı yok mu?
  - Belirtilen yol bir dosya mı yoksa bir klasör mü?
  - Dosya veya klasörün özellikleri kontrol edilebilir veya düzenlenebilir.
  - Yeni dosya veya klasör oluşturabilir, mevcut olan silinebilir.
  - Bir klasörün içeriği elde edilebilir.
  - Dosya veya klasörün son değiştirme tarihi ve saati alınabilir.

java.io.File Sınıfı

```
// Dosya yolunu temsil eden bir File nesnesi oluşturma
File dosya = new File("/kullanici/ornek/dosya.txt");
boolean varMi = dosya.exists();
boolean dosyaMi = dosya.isFile();
boolean klasorMu = dosya.isDirectory();
boolean okunabilirMi = dosya.canRead();
boolean yazilabilirMi = dosya.canWrite();
boolean calistirilabilirMi = dosya.canExecute();
dosya.createNewFile();
dosya.delete();
String[] klasorIcerigi = dosya.list();
long sonDegisimZamani = dosya.lastModified();
```

Scanner Sınıfı

- Dosya içeriğini okumak, işlemek için *Scanner* sınıfı kullanılır.
- Aynı zamanda diğer girdi verilerini okumayı da destekler.
- Dosya okuma işlemi sırasında *FileNotFoundException* alınabilir.

Scanner Sınıfı

```
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.printStackTrace();
} finally {
   scanner.close();
}
```

Java I/O Kütüphanesi

- Giriş/çıkış işlemleri için güçlü ve esnek bir araç seti sunar.
- Dosya işlemlerinden ağ programlamaya kadar birçok alanda kullanılır.
- Giriş/Çıkış İşlemleri:
  - Dosya, ağ, bellek gibi kaynaklardan veri almak ve göndermek.
- İkili/Metin İşlemleri:
  - İkili ve metin tabanlı veri işlemlerini destekler.
- Sıralı/Rasgele Erişim:
  - Dosyadaki verilere hem sıralı hem rasgele erişimi destekler.

Kütüphanede Bulunan Öğeler

- Sınıflar:
  - *File*: Dosya ve dizin işlemleri için.
  - *InputStream* ve *OutputStream*: Temel giriş/çıkış işlemleri için.
  - Reader ve Writer: Metin tabanlı giriş/çıkış işlemleri için.
- Arayüzler:
  - *Closeable*, *Flushable*: Kullanılan kaynakları serbest bırakma ve temizleme işlemleri için.
- İstisnalar:
  - *IOException*: Giriş/çıkış işlemleri sırasında oluşan hata durumları için.

Java I/O Kütüphanesi

```
File dosya = new File("ornekDosya.txt");
try (BufferedReader okuyucu = new BufferedReader(
                                        new FileReader(dosya)))
{
  String satir;
  while ((satir = okuyucu.readLine()) != null) {
    System.out.println(satir);
  }
} catch (IOException e) {
  System.err.println("Dosya okuma hatası:" + e.getMessage());
  e.printStackTrace();
}
```

java.io.PrintWriter

- Metin dosyalarına kolayca yazma imkanı sağlar.
- *print* ve *println* metodları ile biçimli metin çıktısı sağlar.
- Diğer veri türlerini metin formatına dönüştürerek yazar.
- Oluşturulan dosyanın kapatılması (*close*) unutulmamalıdır.

java.io.PrintWriter

```
PrintWriter yazici = null;
try {
  yazici = new PrintWriter("yeniDosya.txt");
  yazici.println("Merhaba, dünya!");
  yazici.println("Bu bir örnek dosyadır.");
} catch (IOException e) {
  System.err.println("Dosya hatası: " + e.getMessage());
  e.printStackTrace();
} finally {
  if (yazici != null)
    yazici.close();
}
```

Java.io.FileWriter

- Dosyalara karakter tabanlı verileri yazmak için kullanılır.
- Karakter akışlarına veri yazma işlevselliği sağlar.
- Metin biçimlendirme yetenekleri sınırlıdır.
- Daha düşük seviyeli ve esnek bir yaklaşım sağlar.

Java.io.FileWriter

```
FileWriter yazici = null;
try {
  yazici = new FileWriter("yeniDosya.txt");
  yazici.write("Merhaba, dünya!\n");
  yazici.write("Bu bir FileWriter örneğidir.");
  System.out.println("Dosya oluşturuldu ve veri yazıldı.");
  yazici.close();
} catch (IOException e) {
  System.err.println("Dosya hatası: " + e.getMessage());
  e.printStackTrace();
}
```

Tamponlama

- Temel Metodlar
  - read(): byte veya byte dizisi okumak için kullanılır.
  - write(): byte veya byte dizisi yazmak için kullanılır.
- Her byte için disk erişimi, uygulamayı ciddi şekilde yavaşlatır.
- Veriler diskten okunmadan veya diske yazılmadan bir araya getirilmeli.
- Okuma ve yazma işlemleri mümkün olduğunca toplu yapılmalı.
- Bu işlem, fiziksel disk erişimlerini azaltır.

Tamponlama

- *java.io.BufferedInputStream* ve *java.io.BufferedReader*:
  - Okuma işlemlerini hızlandırır.
  - read() metodunu çağırarak bellek içinde bir önbellek kullanır.
- *java.io.BufferedOutputStream* ve *java.io.BufferedWriter*:
  - Yazma işlemlerini hızlandırır.
  - write() metodunu çağırarak bellek içinde bir önbellek kullanır.

java.io.BufferedInputStream

```
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();
  fis.close();
} catch (IOException e) {
  System.err.println("Dosya hatası: " + e.getMessage());
}
```

java.io.BufferedReader

```
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();
} catch (IOException e) {
  System.err.println("Dosya hatası: " + e.getMessage());
  e.printStackTrace();
}
```

java.io.ByteArrayInputStream

- *InputStream* sınıfından türetilmiştir.
- Bellekteki bir *byte* dizisinden okuma yapmak için kullanılır.
- Veri bellekte olduğu için hızlı bir şekilde okuma sağlar.
- Büyük veri setleri için bellek tüketimi sorun olabilir.
- Veri sadece okunabilir, yazılabilir değildir.
- Fiziksel bir kaynağa erişim gerektirmez.
- **Esnek Kullanım**: Farklı tiplerdeki *byte* dizileriyle çalışabilir.

java.io.StringBufferInputStream

- Java'da artık önerilmiyor ve kullanımı tavsiye edilmiyor.
- Yerine java.io.ByteArrayInputStream tercih edilmelidir.
- Performans sorunlarına neden olduğu belirlenmiştir.

java.io.FileInputStream

- *InputStream* sınıfından türetilmiştir.
- Dosyadan doğrudan byte okuma işlemleri için kullanılır.
- Düşük seviyeli ve genel byte verileriyle çalışır.
- Metin dışındaki verileri okumak için kullanılır.
  - Örneğin, resim veya ses dosyalarını okumak.
- Temel Metodlar
  - read(): Tek bir byte okur. Dosyanın sonuna geldiğinde -1 döner.
  - read(byte[] b): Belirtilen byte dizisi boyutu kadar veri okur.
- Doğrudan byte okuma olduğu için metin okuma zorlukları olabilir.
- *FileReader* gibi karakter tabanlı alternatiflere kıyasla daha düşük seviyeli.

java.io.PipedInputStream

- *InputStream* sınıfından türetilmiştir.
- İki thread arasında güvenli veri iletişimi sağlar.
- Senkronize okuma ve yazma işlemleri mümkündür.
- Bir iş parçacığı, *PipedOutputStream* ile veri yazar,
  - diğer iş parçacığı ise *PipedInputStream* ile bu veriyi okur.
- Sadece tek yönlü veri iletimini destekler.

java.io.PipedInputStream

```
PipedInputStream pis = new PipedInputStream();
// Diğer thread'den veri okuma
Thread readerThread = new Thread(() -> {
  try {
    int okunanByte;
    while ((okunanByte = pis.read()) != -1) {
      System.out.print((char) okunanByte);
    }
    pis.close();
  } catch (IOException e) {
    e.printStackTrace();
  }
});
```

java.io.PipedOutputStream

```
PipedOutputStream pos = new PipedOutputStream(pis);
  // Bir thread'den veri yazma
  Thread writerThread = new Thread(() -> {
  try {
    pos.write("Merhaba, PipedInputStream!".getBytes());
    pos.close();
  } catch (IOException e) {
    e.printStackTrace();
  }
});
```

java.io.SequenceInputStream

- Birden fazla *InputStream*'i birleştirerek ardışık bir giriş akışı oluşturur.
- Birinci akış tamamlandığında ikinci akışa geçer.
- Farklı kaynaklardan gelen veriyi tek bir ardışık akış halinde birleştirir.
- Eğer bir akış hatada ise diğer akışa geçilmez.

java.io.SequenceInputStream

```
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();  is2.close();
} catch (IOException e) {
   System.err.println("okuma hatası: " + e.getMessage());
}
```

Bayt Bayt Dosya Kopyalama

```
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);
// Bir byte oku. Dosyanın sonunda -1 dönecektir.
while ((veri = bis.read()) != -1)
  bos.write(veri); // Okunan byte'ı çıkışa yaz
bos.close();
bis.close();
```

Bayt Bayt Dosya Kopyalama

- Belirtilen kaynak dosyadan okunan veriyi hedef dosyaya kopyalar.
- *BufferedInputStream* ve *BufferedOutputStream*:
  - Giriş ve çıkış işlemlerini hızlandırmak için kullanılır.
  - Önbellek kullanarak veri okuma ve yazma işlemlerini optimize eder.
- Okuma ve Yazma İşlemleri:
  - bis.read(): Bir byte veri okur. Dosyanın sonunda -1 döner.
  - bos.write(oneByte): Okunan veriyi çıkış dosyasına yazar.

Yığın Yığın Dosya Kopyalama

```
// Veri, 16K'lık parçalar halinde 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);
// 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();
```

Yığın Yığın Dosya Kopyalama

- Toplu okuma işlemleri, dosya kopyalama performansını artırır.
- Veriyi parçalar halinde işleyerek bellek kullanımını optimize eder.
- Okuma ve Yazma İşlemleri:
  - bis.read(parca): Belirtilen boyuttaki veriyi bir byte dizisine okur.
  - bos.write(parca, 0, size): Okunan veriyi çıkış dosyasına yazar.
- Boyut Belirleme:
  - byte[] bytes = new byte[1024 * 16]:
  - Veriyi 16K'lık parçalar halinde okumak için bir byte dizisi.

Web Sayfası İndirme

```
URL url = new URL("https://sercankulcu.github.io/");
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();
```

Byte ve Karakter Odaklı Giriş Çıkış

- *InputStream* ve *OutputStream* sınıfları,
  - *Byte* odaklı giriş/çıkış destekler.
  - Veri, *byte* dizileri şeklinde işlenir.
  - Genellikle resim, ses, gibi ikili sayısal verilerle çalışır.
- *Reader* ve *Writer* sınıfları,
  - Karakter odaklı giriş/çıkış işlevselliği sunar.
  - Unicode desteği ile metin tabanlı verilerle daha verimli çalışır.
  - Farklı dil ve alfabelerdeki metin verilerini destekler.

Satır Satır Dosya Kopyalama

```
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();
```

Dosya Erişimi: Sıralı ve Rasgele

- Sıralı Erişim:
  - Bir sonraki konumdan okuma yazma işlemidir.
  - Dosya içeriğinin bilinmediği veya
  - Kopya oluşturmak istenilen durumlarda kullanılır.
- Rasgele Erişim:
  - Bir veriye erişmek için dosya içinde **hareket** edebilme yeteneği sunar.
  - Kaydın boyutu ve konumu **biliniyorsa** rasgele erişim kullanılabilir.
  - **Belirli** bir kaydı okuma veya değiştirme ihtiyacı olduğunda kullanışlıdır.

Dosya Erişimi: Sıralı ve Rasgele

```
// Sıralı Erişim
FileInputStream sirali = new FileInputStream("dosya");
int bayt = sirali.read();

// Rasgele Erişim
RandomAccessFile rast = new RandomAccessFile("dosya", "rw");
// Dosyanın 100. byte'ına git
rast.seek(100);
// 100. bytedan itibaren oku
int veri = rast.readInt();
```

java.io.RandomAccessFile

- **RandomAccessFile(String ad, String mod)**
  - Mod: "r": okuma, "rw" okuma ve yazma
- **seek(long konum):** Dosya işaretçisini belirtilen konuma taşır.
- **getFilePointer():** işaretçinin dosyanın başına olan konumunu döndürür.
- **length():** Dosyanın uzunluğunu (byte cinsinden) döndürür.
- **read():** Dosyadan bir sonraki byte'ı okur ve tamsayı olarak döndürür.
- **write(int b):** Bir byte veriyi dosyaya yazar.
- **close():** Sistem kaynaklarını serbest bırakır.

Serileştirme

- Nesneleri ikinci bir ortama taşımak veya geri getirmek için kullanılır.
- Bir sınıf serileştirilebilmesi için *java.io.Serializable* arayüzünü gerçekler.
- Serileştirilen sınıfın kimliği *serialVersionUID* değeri ile tanımlanmalıdır.
- Serileştirme sırasında bazı alanları es geçmek için *transient* tanımlanır.
- *static* veya *transient* (geçici) tanımlanmış nitelikler serileştirilmez.

Serileştirme

```
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 + "]";
  }
}
```

Serileştirme

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

Serileştirme

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

NIO (New I/O)

- Java NIO, gelişmiş I/O işlemleri için güçlü bir API sunar.
- *Buffer*, *channel*, *selector* ve dosya sistemi entegrasyonu içerir.
- Performans ve esneklik sağlar.
- Temel Unsurlar
  - **Buffer**: Veriyi geçici olarak depolayan bellek alanları.
  - **Channel**: Giriş çıkış işlemlerini yöneten nesneler.
  - **Selector**: Birden çok kanalı tek iş parçacığında yönetebilme sağlar.
    - Girdi bekleyen kanalları etkin bir şekilde izler.
  - **Path**: Dosya ve dizin yollarını temsil eder.

NIO (New I/O)

- Avantajlar
  - **Performans**: NIO, geleneksel I/O'ya göre daha verimli ve hızlıdır.
  - **Esneklik**: Tampon ve kanallar sayesinde esnek veri işleme sağlar.
  - **Çoklu Bağlantı**: Bir iş parçacığında birden çok kanalı izleyebilme.
- Dezavantajlar
  - **Karmaşıklık**: Geleneksel I/O'ya göre daha karmaşıktır.

NIO Buffer

- Geçici bellek alanıdır.
- Verileri geçici olarak saklamak için kullanılır.
- Buffer Sınıfları
  - **ByteBuffer**: *byte* türünde verilerini tutar.
  - **CharBuffer**: *char* (karakter) türünde verileri tutar.
  - **ShortBuffer**, **IntBuffer**, **LongBuffer**: tamsayı türünde verileri tutar.
  - **FloatBuffer**, **DoubleBuffer**: ondalık sayı türünde verileri tutar.

Buffer Özellikleri

- Kapasite (Capacity):
  - Tutulabilecek maksimum veri miktarını belirler.
- Konum (Position):
  - Okuma veya yazma işlemlerinin başlayacağı konumu belirler.
- Sınır (Limit):
  - Okuma veya yazma işlemlerinin boyutunu belirler.
- İşaretçi (Mark):
  - Bir pozisyonu işaretlemek için kullanılır.

Buffer

```
// 1 KB boyutunda bir ByteBuffer oluştur
ByteBuffer buffer = ByteBuffer.allocate(1024);
// Veri yazma
buffer.put("Merhaba, Dünya!".getBytes());
// Buffer'ı okuma moduna geçirme
buffer.flip();
// Veriyi okuma
while (buffer.hasRemaining()) {
  System.out.print((char) buffer.get());
}
```

Buffer İşlemleri

- **put():** Veri yazar.
- **get():** Veri okur.
- **flip():** Yazma modundan okuma moduna geçirir.
- **rewind():** Konumu sıfırlar, limit değişmez.
- **clear():** Tampon belleği temizler ve yazma moduna geçer.
- **compact():** Okunan verileri başa taşır. Verilen limit kadar veri taşınır.

Java NIO Buffer

- Avantajlar
  - Esnek bir şekilde veri manipülasyonu sağlar.
  - Verimli bellek yönetimi sağlar.
- Dezavantajlar
  - Kullanımı karmaşıktır.
  - Boyutu dinamik olarak değiştirilemez.

NIO Kanal (Channel)

- Giriş ve çıkış verilerini yönetir.
- Kanallar, dosya, soket veya belirli protokol üzerinden veri alışverişi sağlar.
- Kanal Türleri
  - **FileChannel**: Dosya üzerinde okuma ve yazma işlemleri için kullanılır.
  - **SocketChannel**: TCP tabanlı bir ağ soketi üzerinden veri iletimi sağlar.
  - **ServerSocketChannel**: Gelen bağlantıları kabul etmek için kullanılır.
  - **DatagramChannel**: UDP tabanlı ağ soketi üzerinden veri iletimi sağlar.

Kanal Oluşturma ve Kapatma

```
// FileChannel oluşturma
FileChannel fileChannel = FileChannel.open(
            Paths.get("dosya.txt"), StandardOpenOption.READ);

// SocketChannel oluşturma
SocketChannel socketChannel = SocketChannel.open();
socketChannel.connect(new InetSocketAddress("localhost",
                                            8080));

// Kanal kapatma
fileChannel.close();
socketChannel.close();
```

FileChannel Kullanımı

```
try (FileChannel channel = FileChannel.open(
            Paths.get("dosya.txt"), StandardOpenOption.READ)) {

  ByteBuffer buffer = ByteBuffer.allocate(1024);
  // Dosyadan veriyi okuma
  int bytesRead = channel.read(buffer);
  // Buffer'ı okuma moduna geçirme
  buffer.flip();
  // Veriyi ekrana yazma
  while (buffer.hasRemaining()) {
    System.out.print((char) buffer.get());
  }
}
```

SocketChannel Kullanımı

```
try (SocketChannel socketChannel = SocketChannel.open()) {
  socketChannel.connect(new InetSocketAddress(
                                        "www.example.com", 80));
  ByteBuffer buffer = ByteBuffer.allocate(1024);
  buffer.put("GET / HTTP/1.1\r\n\r\n".getBytes());
  buffer.flip();
  // Veriyi sokete yazma
  socketChannel.write(buffer);
} catch (IOException e) {
  e.printStackTrace();
}
```

NIO Kanal (Channel)

- Avantajlar
  - Farklı kaynaklar arasında veri iletimini sağlar.
  - Birden çok kanal, tek bir iş parçacığında yönetilebilir.
- Dezavantajlar
  - Geleneksel I/O'ya göre daha karmaşıktır.
  - Hata durumları ile başa çıkma gerekir.

NIO Selector

- Birden çok kanalı tek bir iş parçacığında izleme mekanizmasıdır.
- *Selector.open()* ile yeni bir seçici oluşturulur.
- Kanallar, seçiciye kaydedilir.
- Kaydetme işlemi *channel.register(selector, ops)* metodu ile yapılır.
- Kanalın hangi etkinlikler üzerinde çalıştığı *SelectionKey* ile belirlenir.

Selector

```
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
  }
}
```

SelectionKey Etkinlik Tipleri

- OP_READ:
  - Kanaldan okuma yapılabilir durumda.
- OP_WRITE:
  - Kanala yazma yapılabilir durumda.
- OP_CONNECT:
  - Bir bağlantı kurma işlemi tamamlandığında.
- OP_ACCEPT:
  - Bir bağlantı kabul edilebilir durumda.

NIO Selector

- Avantajlar
  - Birden çok kanalı tek bir iş parçacığında izleme yeteneği.
  - Sadece aktif kanalların işlenmesi, kaynak kullanımını optimize eder.
- Dezavantajlar
  - Doğru kullanım için öğrenme eğrisi.
  - Birden çok kanalın durumunu doğru bir şekilde yönetmek gerekir.

SON