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.
- Giriş Akışı: Verinin programa girdiği akışa denir. Örnek: System.in.
- Çıkış 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

- Dosyalar: Veriyi kalıcı olarak saklama yöntemini sağlar.
- Veri Depolama: 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.

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 veya var olan bir dosya açılır.
- Yazma ve Okuma: Veri dosyaya yazılır veya dosyadan okunur.
- Kapatma: Dosya işlemi tamamlandığında kapatılır.

Metin Dosyaları

- Metin dosyaları, yazdırılabilir karakterleri içerir.
- Bitler: Yazdırılabilir karakterleri temsil eder (insanlar tarafından okunabilir).
- Metin dosyaları metin düzenleyicileri kullanılarak düzenlenebilir. Örnekler: Program kaynak dosyaları (.java, .c), Notepad.exe gibi metin düzenleyici ile kaydedilen dosyalar.
- Karakter Seti: Karakterlerin bilgisayar tarafından anlaşılabilir formda kodlanmasını sağlar.
  - ASCII: Temel İngilizce karakterleri içerir.
  - ISO-8859-1: Latin alfabesi temel karakter setini içerir.
  - utf-8: Evrensel karakter setini destekler, çok dilli metinleri kapsar.

İkili Dosyalar

- İkili dosyalar, bilgisayar tarafından yürütülebilen veya özel programlarla işlenebilen kodlanmış verileri içerir.
- İkili dosyalar, bilgisayar tarafından anlaşılabilir ancak insanlar için doğrudan okunamaz.
- Genellikle yürütülebilir dosyalar, görüntüler, müzik veya video dosyaları gibi verileri içerir.
- İkili: Bilgisayar tarafından kolayca okunabilir, ancak insanlar tarafından okunamaz.

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 Karakterler: İngilizce alfabesi, rakamlar, noktalama işaretleri ve temel semboller.
  - Kontrol Karakterleri: Bilgi iletimi sırasında özel işlevlere sahip karakterler.
- ASCII, her karakteri bir sayı ile temsil eder. Örneğin, 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 setini genişleterek, özel karakterlerin ve sembollerin eklenmesini sağlayan bir karakter kodlama sistemidir.
- Temel Karakterler: Standart ASCII karakter setinden alınan temel 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 Yazma

- Sayısal verilerin depolanması, veri tipine ve kullanım senaryosuna bağlı olarak farklı dosya formatları arasında değişebilir.
- İkili dosyalar, sayısal verileri daha verimli bir şekilde depolamak için tercih edilebilir.
- ASCII Kodlu Metin Dosyasına Yazma
  - Her karakter için üç byte kullanılır: 1, 2 ve 7.
  - Bu karakterlerin ikili değerleri: 00110001, 00110010, 00110111.
- İkili Dosyaya Yazma
  - 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 karakter için 3 byte.
  - İkili Dosya (Int): 4 byte.
- Kullanım Alanları
  - ASCII, insanlar tarafından okunabilir veriler için kullanılır.
  - İkili dosyalar, sayısal verilerin daha verimli depolanması için kullanılır.

java.io.File Sınıfı

- java.io.File sınıfı, dosya sistemine erişim sağlamak ve dosya ile ilgili çeşitli işlemleri gerçekleştirmek için kullanılır.
  - Dosya yolu var mı yok mu kontrol edilebilir.
  - Belirtilen yol bir dosya mı yoksa bir klasör mü kontrol edilebilir.
  - 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 varlik = 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

- Scanner sınıfını kullanarak dosya içeriğini okumak, dosyadan veri almak ve işlemek için Java'da yaygın bir yöntemdir.
- Scanner sınıfı, Java'da giriş verilerini okumak için kullanılır.
- Dosya okuma işlemi sırasında FileNotFoundException kontrol edilir.

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

- Java I/O Kütüphanesi, giriş/çıkış işlemlerini gerçekleştirmek için güçlü ve esnek bir araç seti sunar.
- Dosya işlemlerinden ağ programlamaya kadar birçok alanda kullanılan kapsamlı bir kütüphanedir.
- Giriş/Çıkış İşlemleri: Dosya, ağ, bellek gibi kaynaklardan veri almak ve göndermek için kullanılır.
- İkili/Metin İşlemleri: İkili ve metin tabanlı veri işleme işlemlerini destekler.
- Sıralı/Rasgele Erişim: Dosyadaki verilere sıralı veya rasgele erişim sağlamak için kullanılır.

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: Kaynakları serbest bırakma ve temizleme işlemleri için.
- İstisnalar:
  - IOException: Giriş/çıkış işlemleri sırasında oluşan genel hata durumları için.

Java I/O Kütüphanesi

```
File dosya = new File("ornekDosya.txt");
try (BufferedReader reader = new BufferedReader(
                                        new FileReader(dosya)))
{
  String satir;
  while ((satir = reader.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(): Tek bir byte veya byte dizisi okumak için kullanılır.
  - write(): Tek bir byte veya byte dizisi yazmak için kullanılır.
- Temel Sorun: Her byte için disk erişimi, uygulamayı ciddi şekilde yavaşlatabilir.
- Çözüm: Verilerin diskten okunmadan veya diske yazılmadan önce bir araya getirilmesi. Bu, fiziksel disk işlemlerinin sayısını azaltır.

Tamponlama

- java.io.BufferedInputStream ve java.io.BufferedReader:
  - Okuma işlemlerini hızlandırmak için kullanılır.
  - read() metodunu çağırarak bellek içinde bir önbellek kullanır.
- java.io.BufferedOutputStream ve java.io.BufferedWriter:
  - Yazma işlemlerini hızlandırmak için kullanılır.
  - 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());
  e.printStackTrace();
}
```

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 çalışan bir okuma işlevselliği sağlar. 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 dizisine 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 thread, PipedOutputStream ile veri yazar, diğer thread ise PipedInputStream ile bu veriyi okur.
- Sadece bir 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 birleştirmek ve ardışık bir giriş akışı oluşturmak için kullanılır.
- 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("Dosya okuma hatası: " + e.getMessage());
  e.printStackTrace();
}
```

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 ((oneByte = bis.read()) != -1) {
  bos.write(oneByte); // Okunan byte'ı çıkışa yaz
}
bos.close();
bis.close();
```

Bayt Bayt Dosya Kopyalama

- Bu program, belirtilen kaynak dosyadan okunan veriyi hedef dosyaya kopyalar.
- BufferedInputStream ve BufferedOutputStream:
  - Giriş ve çıkış işlemlerini hızlandırmak için kullanılır.
  - Bellekte bir önbellek kullanarak veri okuma ve yazma işlemlerini optimize eder.
- Okuma ve Yazma İşlemleri:
  - bis.read(): Bir byte okur. Dosyanın sonunda -1 döner.
  - bos.write(oneByte): Okunan byte'ı çıkış dosyasına yazar.

Yığın Yığın Dosya Kopyalama

```
// Veri, 16K'lık parçalar halinde okunacak
byte[] bytes = 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 ((size = bis.read(bytes)) > -1) {
  bos.write(bytes, 0, size); // Okunan parçayı çıkışa yaz
}
bos.close();
bis.close();
```

Yığın Yığın Dosya Kopyalama

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

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, veya diğer binary verilerle çalışır.
- Reader ve Writer sınıfları,
  - Java 1.1 ile kütüphaneye eklenmiştir.
  - Karakter odaklı giriş/çıkış işlevselliği sunar.
  - Unicode desteği ile metin tabanlı verilerle daha etkili çalışır.
  - Farklı dil ve alfabelerdeki metin verilerini düzgün bir şekilde işler.

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 line;
// dosya sonuna kadar bir satır oku
while ((line = br.readLine()) != null) {
  pw.println(line); // satırı yaz
}
pw.close();
br.close();
```

Dosya Erişimi: Sıralı ve Rasgele

- Sıralı Erişim:
  - Bir sonraki byte, string veya sayıyı okuma veya bir sonraki konuma yazma işlemidir.
  - Dosya içeriğinin bilinmediği veya kopya oluşturmak istenilen durumlarda kullanılır.
- Rasgele Erişim:
  - Bir dosyanın içindeki belirli bir kaydı okumak veya değiştirmek için dosya içinde hareket edebilme yeteneği sunar.
  - Kayıtların 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 name, String mode)
  - Mode: "r": okuma, "rw" okuma ve yazma
- seek(long pos): Dosya işaretçisini belirtilen konuma taşır.
- getFilePointer(): Dosya işaretçisinin dosyanın başına konumunu döndürür.
- length(): Dosyanın uzunluğunu (byte cinsinden) döndürür.
- read(): Dosyadan bir sonraki byte'ı okur ve bunu tamsayı olarak döndürür.
- write(int b): Bir byte veriyi dosyaya yazar.
- close(): Sistem kaynaklarını serbest bırakmak için önemlidir.

Serileştirme

- Serileştirme, nesneleri ikinci bir ortama taşımak ve oradan geri getirmek için kullanılır.
- Bir sınıfın serileştirilebilmesi için java.io.Serializable arayüzünü implemente etmesi gerekir.
- Serileştirilen sınıfın kimliği serialVersionUID değeri ile tanımlanmalıdır.
- Serileştirme sırasında belirli alanları dışarıda bırakmak için transient anahtar kelimesi kullanılır.
- Statik ve transient (geçici) alanlar serileştirilmez.

Serileştirme

```
class Ogrenci implements Serializable {
  private static final long serialVersionUID = 1L;
  private String ad;
  private int yas;
  // Constructor
  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();
}
```

Java NIO (New I/O)

- Java NIO, gelişmiş I/O işlemleri için güçlü bir API sunar.
- Buffer'lar, channel’lar, selector'lar ve dosya sistemi entegrasyonu içerir.
- Performans ve esneklik avantajlarına sahiptir.
- Temel Unsurlar
  - Buffer'lar: Veriyi geçici olarak depolayan bellek alanları.
  - Channel'lar: Giriş ve çıkış verilerini okuma ve yazma işlemlerini yöneten nesneler.
  - Selector: Birden çok kanalı tek bir iş parçacığında yönetme olanağı sağlar. Etkin bir şekilde giriş bekleyen kanalları izler.
  - Path ve File System: Dosya ve dizin yollarını temsil eden sınıflar.

Java NIO (New I/O)

- Avantajlar
  - Performans: NIO, geleneksel I/O'ya göre daha verimli ve hızlıdır.
  - Esneklik: Buffer'lar ve channel’lar sayesinde esnek veri işleme sağlar.
  - Çoklu Bağlantı: Bir iş parçacığında birden çok kanalı izleyebilme.
- Dezavantajlar
  - Karmaşıklık: Geleneksel I/O'ya göre daha karmaşık bir API.

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());
}
```

Channel

```
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());
  }
} catch (IOException e) {
  e.printStackTrace();
}
```

Selector

```
ServerSocketChannel channel = ServerSocketChannel.open();
Selector selector = Selector.open();
channel.bind(new InetSocketAddress(8080));
channel.configureBlocking(false);
channel.register(selector, SelectionKey.OP_ACCEPT);
while (true) {
  int readyChannels = selector.select();
  if (readyChannels > 0) {
    Set<SelectionKey> selectedKeys = selector.selectedKeys();
    for (SelectionKey key : selectedKeys) {
      if (key.isAcceptable()) {
        // Bağlantı kabul işlemi
      } else if (key.isReadable()) {
        // Veri okuma işlemi
      }
    }
    selectedKeys.clear(); // İşlenen anahtarları temizle
  }
}
```

SON