Veri Akımları

Veri Akımı Nedir?

Giriş/çıkış eylemleri, bilgisayar ile dış dünya arasında iletişim kurar. Aslında *giriş/çıkış* işlemeleri işletim sisteminin görevidir. Ama hemen her program kendisine gerekli olan *giriş/çıkış* işlemlerinin yapılabilmesi için, işletim sistemiyle ortak çalışmalıdır. Java'nın ilk sürümlerinde *giriş/çıkış* işlemleri, *byte akımları* (byte stream) ile yapılıyordu. *Byte akımları*, tek tek byte'ların açılan bir yoldan (*Stream* nesnesi) bilgisayara (programa) ard arda girişi ya da çıkışıdır. Aslında bu yöntem, makina diline yakın olduğu için etkin bir yöntemdir.

JDK 1.1 sürümüne kadar veri akımları yalnızca *byte akımlarından* ibaret kaldı. Başka bir deyişle, veriler 8-bitlik diziler halinde akıyordu. Gelen veri için *InputStream* sınıfı, giden veri için *OutputStream* sınıfı yeterli oluyordu. 16-bitlik Unicode karekterleri için akım, JDK 1.1 sürümünde ele alındı ve *java.io.Reader* ile *java.io.Writer* sınıfları yazıldı.

J2SE 1.4.2 sürümü, giriş/çıkış işlemlerinde büyük değişiklik getiren *java.nio* paketini ortaya koydu. Aslında bu paket *java.io* paketi üzerine kuruludur ve onun yeteneklerini artırır. Çünkü *giriş/çıkış* işlemlerine *byte akımı* yanına *blok* giriş/çıkış işlemini de koydu. Bu yöntem verileri bloklar halinde alır ve verir. Bu yeni sisteme *NIO* (new io, yeni giriş/çıkış) denildi. *NIO*'nun getirdiği yenilik, tek tek *byte* giriş-çıkışı yerine *blok* giriş çıkışını koyarak, işlemleri hızlandırmaktır. Java 7 sürümüyle gelen NIO.2 paketi sözkonusu yetenekleri daha da artırdı.

Şimdiye dek giriş/çıkış (I/O) işlemleriyle ilgili örnek programlar yazmadık. Yalnızca, ekran çıktıları için `System.out.print()`, `System.out.println()` ve `System.out.printf()` metotlarını içeren basit deyimlerle yetindik. Bunun apaçık bir nedeni vardır. Java programlarının çoğu konsol ile basit text giriş/çıkışı yapmanın ötesindedir. Java'da, çoğunlukla, giriş/çıkış işlemleri için görsel estetiğe sahip applet'ler ve başka grafik arayüzler kullanır. Bunlar, Java'nın *Abstract Windowing Toolkit (awt)* ve onun üzerine kurulan *javax.swing* paketine dayanır. Elbette, konsoldan text tabanlı giriş/çıkış işlemleri, diğer dillerde olduğu gibi, Java öğretimi için de önem taşır. Ama, text tabanlı giriş/çıkış işlemleri Java'da sınıfları, nesneleri ve metotları bilmeyi gerektiriyor. O nedenle, giriş/çıkış işlemleri bilinçli olarak geriye bırakılmıştır. Böyle yapılmış olmasının, java dilini öğrenmeyi geciktirdiği söylenemez. Aksine, bu aşamada, öğrenci giriş/çıkış eylemlerini daha iyi kavrayacaktır.

Bazı kaynaklar Java'da giriş/çıkış işlemlerini gereğinden çok karmaşık bulur. Hattâ, giriş/çıkış işlemlerini kolaylaştıran arayüzlerin yazıldığı da görülür. Oysa, başlangıçta öğrenciye karmaşık görünen Java'nın giriş/çıkış işlemleri, ona aşina olanlara esneklik ve kolaylık sağlar. Java Dili'nde giriş/çıkış işlemleri *fortran, cobol, basic, Pascal, C* gibi prosedural dillerden oldukça farklıdır. Bunun apaçık bir nedeni vardır. Prosedural diller, hangi donanım üzerinde ve hangi işletim sistemi altında çalışacaksa, ona uyan giriş/çıkış yordamlarını tanımlayabilirler. Bir platform için tanımlı olan yordam başkasında çalışmaz. Oysa, Java platformdan bağımsızdır. Donanım ve işletim sistemine bağlı

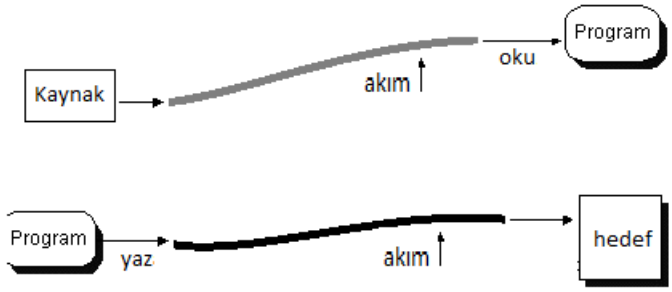
olmaksızın, ağ üzerindeki her bilgisayardan veri alabilir ve her bilgisayara veri gönderebilir. Bunu yapabilmesi için, giriş/çıkış yöntemlerinin amaca uygun çeşitlilik göstermesi doğaldır.

Giriş/Çıkış Akımları

(Input - Output Streams)

java.io'da veri giriş/çıkış işlemlerini, bir yerden başka bir yere (byte dizileri halinde) akan veriler olarak düşünebiliriz. Böyle olunca, basit veri giriş/çıkış (I/O) işlemlerini aşan çok kapsamlı bir olgu karşımıza çıkar. Veriler, bir kanaldan akan su gibi bir yerden başka bir yere akıtılır. Bu akış, basitçe, aynı makinada bir bellek alanından başka bir bellek alanına olabileceği gibi uzaktaki iki makinanın birinden ötekine de olabilir. Verinin kalkış veya varış yerleri bir bellek alanı, bir dosya, ağ üzerinde bir bilgisayarın kayıt ortamı gibi veri tutabilen bir yerdir. Tabii, bunlara klâvye, ekran, ses birimleri ve benzerlerini de eklemek gerekir. Karmaşayı önlemek için, teknik bir tanım yapabiliriz.

Giriş akımı, bir veri kaynağından bir programa veri girişidir. *Çıkış akımı* ise, bir programdan bir hedefe veri gidişidir. Burada *program* terimi, giriş ya da çıkışı yapan programın gelen/giden veriye ana bellekte ayırdığı bellek alanının yerine kullanıldığına dikkat ediniz.



Verinin geldiği ve gittiği fiziksel ortamlar birbirlerinden tamamen farklı olsalar bile, Java'da `giriş/çıkış` akımları birbirine benzer. Dolayısıyla, Java'nın `I/O` sınıfları her türlü veriye ve donanıma uygulanabilme yeteneğine sahiptirler.

Java'da veri giriş/çıkışı için daima şu yol izlenir:

Okuma	Yazma
`akımı aç`	`akımı aç`

```
while (veri varsa)          while (veri varsa)
   veriyi oku                  veriyi yaz
akımı kapat                 akımı kapat
```

java.io Paketi

Veri akımları ile sisteme giriş/çıkış, serileştirme ve dosya sistemi ile ilgili işleri yapan yapı taşlarını içeren pakettir.

java.io paketinde 16 arayüz, 50 sınıf ve 16 hata yakalayıcı (exceptions) vardır. Bu pakette yer alan sınıflar, platformdan bağımsız olarak, bir ortamdan başka bir ortama veri akımını sağlayacak yeteneklere sahiptirler. *java.io* paketinde yer alan öğeler Java 7 API 'den görülebilir. Bu kitap yazıldığı sırada, anılan API'nin web adresi şudur:

http : //download . o r a c l e . com/ j a v a s e /7/ docs / api / java / i o /package summary . html

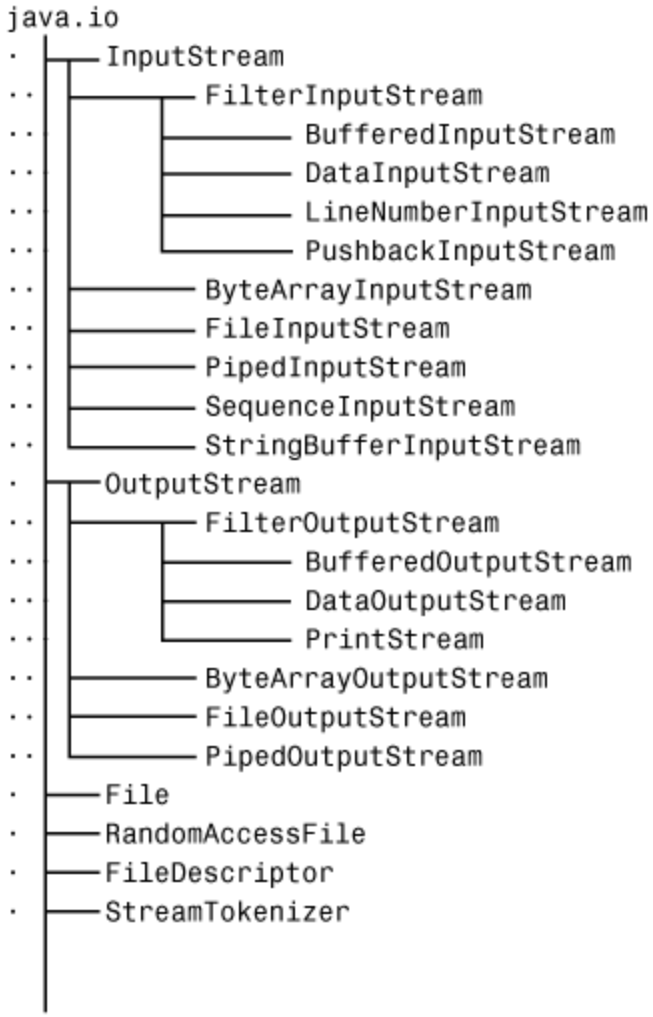
java.io paketi giriş/çıkış birimleri, dosya sistemleri, ağ, bellek alanı gibi farklı ortamların birinden ötekine veri akımını sağlayan ve serileştirme işlemlerini yapan arayüzler, sınıflar, metotlar ve hata yakalayıcılardan oluşan büyük bir pakettir. Bu paketle kaynaktan hedefe veri akımları yapılabilir. Bu akımlar makina düzeyine yakın olduğu için yetenekleri çoktur, ama hızları göreli olarak yavaştır. Daha hızlı giriş/çıkış işlemleri için, J2SE 1.4.2 sürümüyle gelen `java.nio` (new io) paketi kullanılabilir. Bu bölümde `java.io`'yu, sonraki bölümde ise `java.nio` paketini ele alacağız.

Java'da `byte` ve `character` akımlarını yapan sınıflar ve metotlar, `java.lang.Object` sınıfından türetilen dört tane soyut sınıftan elde edilir:

1. java.io.InputStream
2. java.io.OutputStream
3. java.io.Reader
4. java.io.Writer

Tartışma: Bu dört soyut sınıftan elde edilen veri akımı (stream) sayısı 60'dan çoktur. Bu kadar çok sayıda veri yolunun olması bazılarına göre, "programcıları öğrenmekten alıkoyuyor". Ancak, bazılarına göre, farklı giriş/çıkış türleri için farklı akımların (stream) olması, programın hatasız çalışmasını sağlar ve dolayısıyla programcılar için bir avantaj sayılır.

`Character akımları` için kullanılan teknikler, büyük ölçüde `byte akımları` için kullanılan tekniklere benzer. Byte akımını yapan sınıf adları ile character akımını yapan sınıf adları, çoğunlukla son takılarıyla birbirlerinden ayrılırlar. Böyle oluşu, öğrenmeyi kolaylaştıracaktır.



Java.io paketinde yer alan sınıflar

Verinin geldiği ve gittiği fiziksel ortamlar birbirlerinden tamamen farklı olsalar bile, Java'da giriş/çıkış akımları birbirine benzer. Dolayısıyla, Java'nın I/O sınıfları her türlü veriye ve donanıma uygulanabilecek esnekliğe ve kolay kullanılma özeliğine sahiptirler.

Her şeyden önce, Java'da veri akımlarının iki ana gruba ayrıldığını söylemeliyiz:

1. Byte akımları (byte streams)
2. Character akımları (character streams)

Byte akımları (byte streams)

Java'da byte giriş/çıkış işlemleri `java.io` paketi içindeki `InputStream` ve `OutputStream` sınıfları ile bunların altsınıfları tarafından yapılır. Bu sınıfları ayrı bölüm olarak ele alacağız.

Byte akımları binary verilerin giriş/çıkışını yaparlar. Karekter tipinden olmayan resim, grafik, ses ve benzer verileri taşıyan bütün veri akımları byte akımları'dır. Karekter akımları, adından da anlaşılacağı üzere, `char` tipinden verilerin giriş/çıkışını yapar. `Unicode` kullandığı için, farklı ülkelerin alfabelerine uyum sağlar. Byte akımında bir karekter ascii kodu ile okunur; yani int değer alır. Onun ait olduğu karektere dönüştürülmesi gerekir. Oysa karekter akımında, dönüştürme eylemi ilgili metot tarafından yapılır; o iş için programcının dönüşüm deyimi yazması gerekmez. O nedenle, metin okununacaksa, karekter akımları byte akımlarından daha kolay uygulanır.

Tabii, `InputStream` sınıfı, byte ya da karekter akımından farklı türdeki veri girişini sağlayan altsınıflara sahiptir ve onların her veri tipine uyacak kadar çok sayıda metotları vardır.

Ama, bütün akımların esasta byte akımları olduğunu, karakter akımının ve öteki akımların onun üzerine kurulduğunu bilmeliyiz. Önceden belirtildiği üzere, (Java 1.0) sürümünde karekter akımı yoktu. Bütün giriş/çıkış işlemleri byte akımları olarak düzenlenmişti. JDK 1.1, bu eksikliği gidermiş ve karakter akımlarını sisteme katmıştır.

Çoğunlukla, veriyi okuyup yazarken, onu amaca uygun hale getirmek gerekir. Bu işi yapan sınıflar vardır:

Byte akımı yapan altsınıflardan bazıları şunlardır:

`InputStream`	`BufferedInputStream`	`ByteArrayInputStream,`
`DataInputStream`	`FilterInputStream`	`PrintStream`
`PushbackInputStream`	`RandomAccessFile`	`FileInputStream`
`PipedInputStream`	`SequenceInputStream`	
`OutputStream`	`BufferedOutputStream`	`ByteArrayOutputStream,`
`DataOutputStream`	`FileOutputStream`	`FilterOutputStream,`
`PipedOutputStream`		

Karakter akımları

(character streams)

Karakter akımları java.io paketi içindeki `Reader` ve `Writer` sınıfları ve onların altsınıfları tarafından yapılır. Karekterleri birer birer okumak, karekter dizilerini okumak, bütün satırı okumak, dosya okumak ve dosyaya yazmak gibi eylemeleri yapan altsınıfları ve onların çok sayıda metodu vardır.

Karakter akımları yapan alt-sınıflar:

`Reader,`	`BufferedReader,`	`CharArrayReader,`	`FileReader,`
`FilterReader,`	`InputStreamReader,`	`LineNumberReader,`	`PipedReader,`
`PushBackReader,`	`StringReader`		
`Writer,`	`BufferedWriter,`	`CharArrayWriter,`	`FileWriter,`
`FilterWriter,`	`OutputStreamWriter,`	`PipedWriter,`	`PrintWriter,`
`StringWriter`			

Her iki gruptan çok kullanılan bazı sınıfları ayrı bölümlerde ele alacağız.