دانشکده مهندسی و علوم کامپیوتر

برنامه نویسی پیشرفته

وحیدی اصل

برنامه نویسی ورودی/خروجی (I/O)-بخش دوم

RandomAccessFile

- این کلاس به شما امکان می‌دهد تا درون یک فایل حرکت کرده (آن را پیمایش کنید) و بتوانید از آن بخوانید یا در آن بنویسید.

- با استفاده از FileInputStream و FileOutputStream می‌توانید بخشهایی از فایل را با بخشهایی جدید جایگزین کنید.

- نحوه ایجاد یک RandomAccessFile:
- یک شیئ را به صورت زیر از این کلاس ایجاد می‌کنیم:

```
RandomAccessFile file = new RandomAccessFile("c:\\data\\file.txt", "rw");
```

- پارامتر دوم در سازنده (rw) مُد باز کردن فایل را مشخص می‌کند.

- "rw"به معنای مُد خواندن/ نوشتن است.

حرکت برروی RandomAccessFile

- برای خواندن یا نوشتن در یک محل خاص از یک RandomAccessFile باید ابتدا اشاره‌گر فایل را در محلی که می‌خواهیم بخوانیم یا در آن بنویسیم، قرار دهیم.
- اینکار را با متد seek() انجام می‌دهیم.
- با متد getFilePointer() از مکان فعلی اشاره‌گر فایل آگاه می‌شویم.
- یک مثال ساده:

```
RandomAccessFile file = new RandomAccessFile("c:\\data\\file.txt", "rw");
file.seek(200);
 long pointer = file.getFilePointer();
 file.close();
```

سوال: اگر بعد از آخرین دستور، getFilePointer() را فراخوانی کنیم، خروجی چه خواهد بود؟

خواندن از یک RandomAccessFile

- برای خواندن از یک RandomAccessFile می‌توانید یکی از متدهای read() که مناسب کاربردتان است، برگزینید.
- یک مثال ساده:

```
RandomAccessFile file = new
RandomAccessFile("c:\\data\\file.txt", "rw");
 int aByte = file.read()
 file.close();
```

- متد read() بایتی را که در محل اشاره‌گر فایل در شیئ ایجاد شده از RandomAccessFile قرار دارد، می‌خواند.
- توجه کنید که متد read() پس از عمل خواندن، اشاره‌گر فایل را درون فایل مربوطه یک بایت به جلو می برد! این به این معناست که می‌توانید بدون دغدغه جلوبردن دستی اشاره‌گر، متد read را به میزان لازم فراخوانی کنید.

نوشتن در RandomAccessFile

- نوشتن در RandomAccessFile می‌تواند با استفاده از یکی از چندین متد write() انجام شود.
- مثال:

```
RandomAccessFile file = new RandomAccessFile("c:\\data\\file.txt", "rw");
file.write("Hello World".getBytes());
file.close();
```

- مانند متد read() متد write() اشاره‌گر فایل را پس از اجرا به جلو می‌برد.

Java IO: Streams

- Java IO streams جریانهایی از داده‌ها هستند که هم می‌توانند خوانده و هم نوشته شوند.
- اغلب به منابع داده‌ای مبدا و مقصد، نظیر فایلها و اتصالات شبکه وصل می‌شوند.
- در یک جریان، اندیس (برخلاف آرایه) برای خواندن یا نوشتن داده‌ها معنا ندارد.
- معمولاً نمی‌توانیم برروی جریان به جلو یا عقب حرکت کنیم. در حالی که در آرایه و RandomAccessFile اینکار امکان پذیر است.
- یک جریان در حقیقت یک دنباله پیوسته از داده‌ها می‌باشد.
- برخی پیاده‌سازیها (زیرکلاسهای Streams) نظیر PushbackInputStream به شما امکان می‌دهند داده ها را در جریان push back کنید تا بعدا دوباره خوانده شود. اما تنها بخشی محدودی از داده‌ها قابل push back هستند و نمی‌توانید پیمایش کاملی برروی جریان داشته باشید.
- در جریان، داده‌ها به صورت ترتیبی مورد دستیابی قرار می‌گیرند.
- جریانهای داده‌ای یا بایتی و یا کاراکتری هستند.
- InputStream یا OutputStream مبتنی بر بایت هستند. این جریانها در هر لحظه یک بایت را می‌خوانند یا می‌نویسند.
- DataInputStream و DataOutputStream دو استثنا هستند که می‌توانند در هر لحظه یک مقدار int، long ، float و double بخوانند یا بنویسند.

InputStream

- کلاس java.io.InputStream، یک کلاس پایه (والد) برای همه جریانهای ورودی جاوا (Java IO input streams) می‌باشد.
- این کلاس انتزاعی (abstract) است.
- برای خواندن داده‌ها از یک منبع داده‌ای اغلب از این کلاس استفاده می‌کنند. در نتیجه، برنامه شما روی انواع مختلفی از داده‌های جریانی کار خواهد کرد.
- برای خواندن داده‌ها در اشیای ساخته شده از **فرزندان** این کلاس، از متد read() استفاده می‌شود.
- این متد، یک مقدار int برمی‌گرداند که حاوی مقدار بایتی خوانده شده است.
- اگر داده‌ای برای خواندن نمانده باشد، این متد مقدار ۱- برمی‌گرداند.

یک مثال:

```
InputStream input = new FileInputStream("c:\\data\\input-file.txt");
int data = input.read();
while(data != -1)
  {
   data = input.read();
  }
```

مثال Java InputStream

- FileInputStream یکی از فرزندان کلاس انتزاعی Java InputStream است.
- در جاوا برای خواندن داده‌های بایتی از فایلها از این کلاس استفاده می‌شود.
  - یک بایت در هر لحظه
- مثال:

```
InputStream inputstream = new FileInputStream("c:\\data\\input-text.txt");
 int data = inputstream.read();
 while(data != -1) {
 //do something with data... doSomethingWithData(data);
data = inputstream.read(); }
inputstream.close();
```

- این مثال یک شیئ جدید از FileInputStream ایجاد می‌کند.

read()

- این متد، یک مقدار int برمی‌گرداند که حاوی مقدار بایتی خوانده شده است.

- int data = inputstream.read();
- مقدار int را می‌توان به صورت زیر به char تبدیل نمود:
- char aChar = (char) data;
- زیرکلاسهای InputStream ممکن است دارای متدهای read() مختلفی باشند.
- برای مثال، این متد در کلاس **DataInputStream** به شما امکان می‌دهد داده‌های اصلی مانند int، long، float، double، boolean و غیره را با متدهای readBoolean() و readDouble() و غیره بخوانید.

End of Stream

- اگر متد read() مقدار ۱- برگرداند، به پایان جریان رسیده‌ایم.
- به این معنا که داده بیشتری برای خواندن در InputStream باقی نمانده است.
- هرگاه به پایان جریان رسیدیم، می‌توانیم با close() آن را ببندیم.

خواندن از InputStream با متد read(byte[])

- کلاس InputStream دارای دو متد read() دیگر هم هست که می‌توانند داده‌ها را از منبع داده ای InputStream به درون یک آرایه از نوع بایت بریزند.
- این متدها :

int read(byte[])

int read(byte[], int offset, int length)

- خواندن یک آرایه از بایتها در یک لحظه به مراتب سریعتر از خواندن بایت به بایت داده‌ها می‌باشد.

دو متد read دیگر

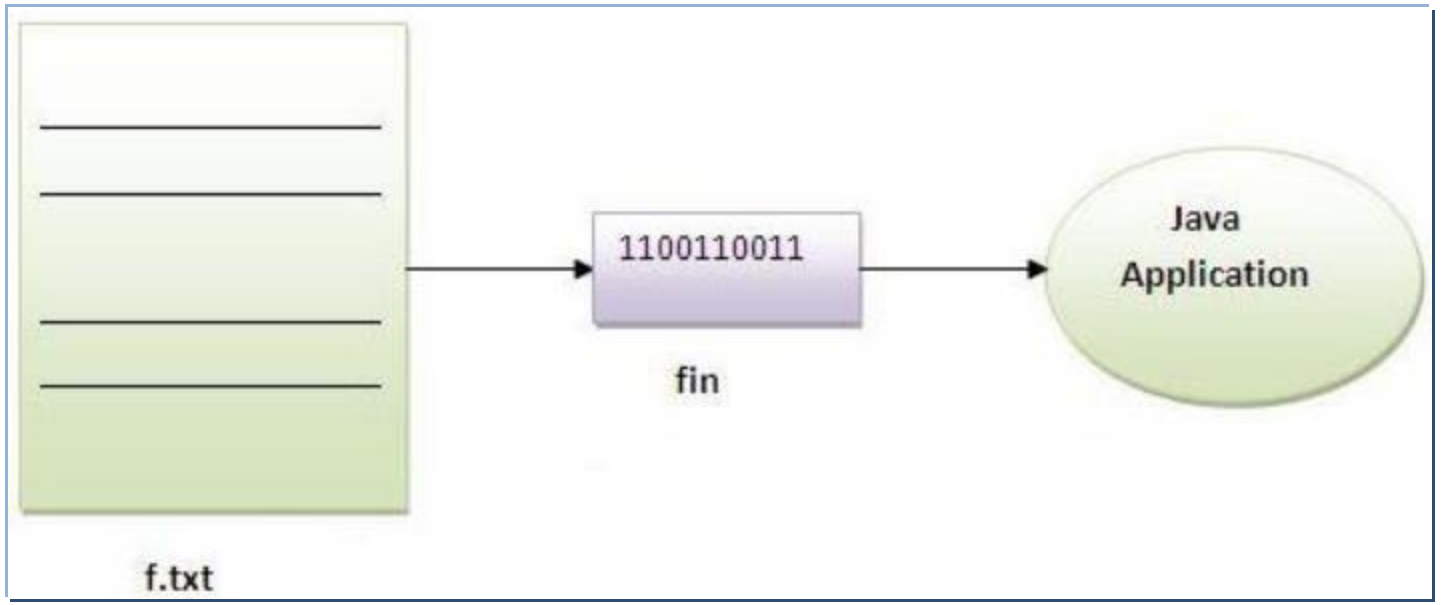
- متد read(byte[]) می‌کوشد تا حداکثر امکان بایتهای بیشتری را خوانده و در آرایه بایتی که به عنوان پارامترش مشخص شده، قرار دهد.
- این متد یک مقدار int برمی‌گرداند که می‌گوید چند بایت خوانده شده‌اند.
- در مواقعی که تعداد بایت خوانده شده از InputStream کمتر از اندازه واقعی آرایه باشد، سایر خانه‌های آرایه، مقادیری را که از قبل در خود داشتند، نگهداری خواهند کرد.
- متد read(byte[], int offset, int length) مانند متد قبلی یک آرایه از بایتها را می‌خواند: از بایت offset شروع می‌کند و به طول length در آرایه قرار می‌دهد.
- برای هر دو متد وقتی به انتهای جریان می‌رسیم، مقدار ۱- برگردانده می‌شود.

مثال

- مثال زیر را ببینید:

```
InputStream inputstream = new FileInputStream("c:\\data\\input-
    text.txt");
 byte[] data = new byte[1024];
 int bytesRead = inputstream.read(data);
 while(bytesRead != -1) {
 for(int i=0;i<bytesRead;i++)  System.out.print("  "+data[i]);
 bytesRead = inputstream.read(data);
 }
 inputstream.close();
```

نمایی از شیئ کلاس FileInputStream

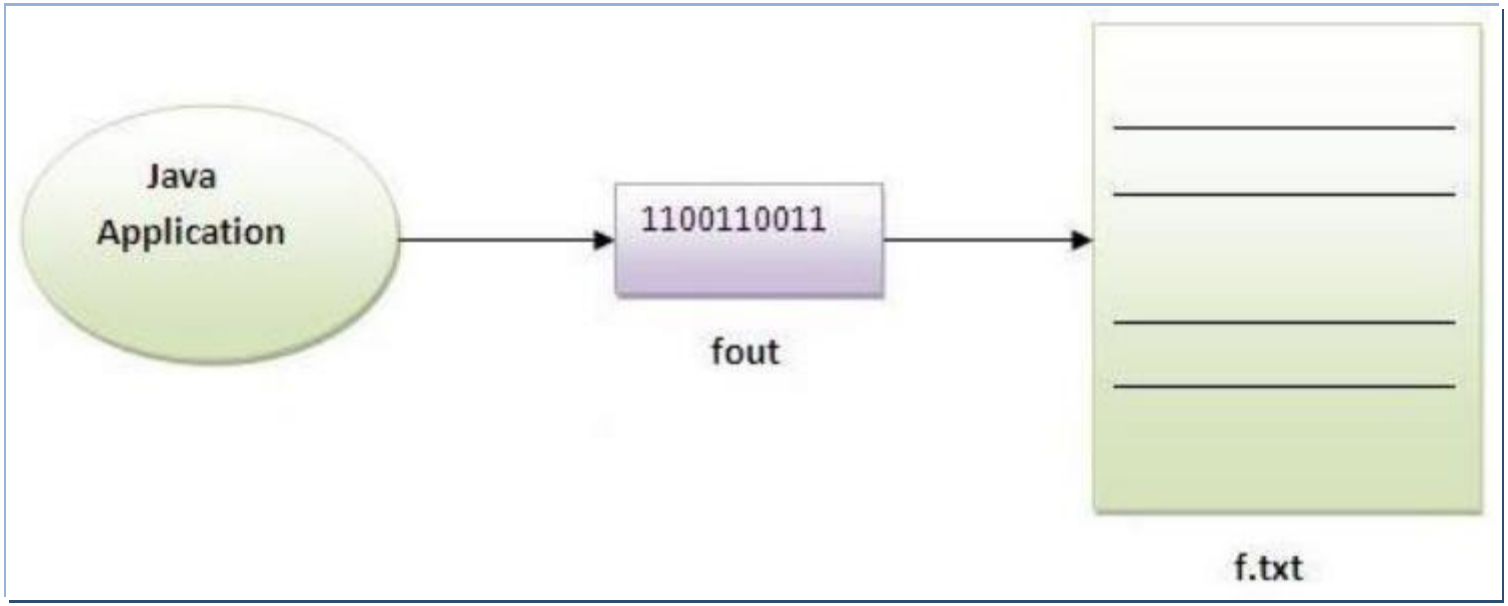


OutputStream

- کلاس java.io.OutputStream، یک کلاس پایه (والد) برای همه جریانهای خروجی جاوا (Java IO Output streams) می‌باشد.
- این کلاس انتزاعی (abstract) است.
- برای نوشتن داده‌ها در یک مقصد داده‌ای اغلب از این کلاس استفاده می‌کنند. در نتیجه برنامه شما روی انواع مختلفی از داده‌های جریانی کار خواهد کرد.
- برای نوشتن داده‌ها در اشیای ساخته شده از **فرزندان** این کلاس، از متد write استفاده می‌شود.
- مثال:

```
OutputStream output = new FileOutputStream("c:\\data\\output-file.txt");
output.write("Hello World".getBytes());
output.close();
```

نمایی از شیئ کلاس FileOutputStream



OutputStream و مقاصد داده ای

- یک OutputStream به یک مقصد داده‌ای نظیر فایل، پایپ یا اتصال شبکه‌ای وصل می‌شود.
- مقصد داده‌ای یک شیئ ساخته شده از کلاس OutputStream، جایی است که داده‌های نوشتَه شده در شیئ سرانجام به آنجا منتقل خواهد شد.

- **write(byte)**
- **متد write(byte) برای نوشتن یک بایت در OutputStream به کار می‌رود.**
- **زیرکلاسهای OutputStream دارای متدهای write() دیگری نیز هستند.**
- **برای مثال، DataOutputStream به شما امکان می‌دهد داده‌هایی از انواع اصلی مانند int، long، float، double، boolean و غیره را در قالب متدهای writeBoolean()، writeDouble() و غیره در یک مقصد داده‌ای بنویسید.**

مثالی از متد write()

```
OutputStream output = new FileOutputStream("c:\\data\\output-text.txt");
while(hasMoreData()) {
  int data = getMoreData();
   output.write(data); }
   output.close();
```

- در ابتدا یک شیئ از FileOutputStream ساخته می‌شود که داده‌ها در آن نوشته می‌شوند.
- سپس در یک حلقه while تا زمانی که داده‌ای برای نوشتن وجود دارد، داده‌های در جریان نوشته می‌شوند.
- درون while مقدار داده‌ای از جایی گرفته می‌شود و در شیئ output ریخته می‌شود.
- سرانجام با بستن جریان، محتوای آن به مقصد داده‌ای که در این مثال یک فایل است، منتقل خواهد شد.

write(byte[])

- کلاس OutputStream دارای دو متد write() دیگر هم هست که می‌توانند یک آرایه از نوع بایت یا بخشی از آن را به دورن شیئ ساخته شده از OutputStream بریزند.
- این متدها :

write(byte[] bytes)

write(byte[] bytes, int offset, int length)

- متد اول همه بایتهای آرایه بایتی را به درون OutputStream می‌ریزد.
- متد دوم تعداد length بایت را که از اندیس offset آرایه bytes آغاز می‌شود در شیئ OutputStream می‌ریزد.

flush()

- متد flush() در کلاس OutputStream همه داده‌های قرارگرفته در شیئ OutputStream را به مقصد داده‌ای (نظیر فایل) منتقل می‌کند.
- برای مثال اگر OutputStream یک FileOutputStream باشد، تا قبل فراخوانی متد flush() داده‌های نوشته شده در آن به دیسک منتقل نمی‌شود.
- این داده‌ها در بخشی از حافظه شیئ، بافر می‌شود.
- با فراخوانی متد flush()، اطمینان می‌یابیم که همه داده‌های بافر شده در دیسک یا شبکه یا هر مقصد داده‌ای دیگر فلاش (نوشته) خواهد شد.

close()

- بعد از اینکه داده ها را در OutputStream نوشتیم، باید جریان را ببندیم.
- اینکار با فراخوانی متد close() انجام می‌شود.
- چون امکان بروز استثنای IOException در حین کار با write() وجود دارد، بهتر است متد close() را در بلاک finally بنویسیم.
- مثال:

```
OutputStream output = null;
 try{
    output = new FileOutputStream("c:\\data\\output-text.txt");
    while(hasMoreData()) {
      int data = getMoreData();
       output.write(data);
    } }
  finally { if(output != null) { output.close(); } }
```

مثالی از خواندن داده از یک فایل جاوا و نوشتن همزمان آن در یک فایل دیگر

- با استفاده از کلاس FileInputStream می‌توانیم از هرفایلی داده‌ها را بخوانیم.
- این فایل می‌تواند یک فایل جاوا، یک فایل تصویر، یک فایل ویدئو یا غیره باشد.
- در این مثال داده‌ها را از یک فایل جاوا به نام C.java می‌خوانیم و در فایل دیگری به نام M.java می‌نویسیم.

```
import java.io.*;

class C{
public static void main(String args[])throws Exception{

FileInputStream fin=new FileInputStream("C.java");
FileOutputStream fout=new FileOutputStream("M.java");

int i=0;
while((i=fin.read())!=-1){
fout.write((byte)i);
}

fin.close();
}
}
```

SequenceInputStream class

- هرگاه بخواهیم محتویات چند فایل را پشت سرهم بخوانیم:

```
import java.io.*;
class Simple{
  public static void main(String args[])throws Exception{
   FileinputStream fin1=new FileinputStream("f1.txt");
   FileinputStream fin2=new FileinputStream("f2.txt");

   SequenceinputStream sis=new SequenceinputStream(fin1,fin2);
   int i;
   while((i=sis.read())!=-1){
    System.out.println((char)i);
   }
   sis.close();
   fin1.close();
   fin2.close();
  }
}
```

خواندن از دو فایل و نوشتن در یک فایل دیگر

```
import java.io.*;
class Simple{
  public static void main(String args[])throws Exception{

    FileinputStream fin1=new FileinputStream("f1.txt");
    FileinputStream fin2=new FileinputStream("f2.txt");

    FileOutputStream fout=new FileOutputStream("f3.txt");

    SequenceinputStream sis=new SequenceinputStream(fin1,fin2);
    int i;
    while((i.sisread())!=-1)
    {
      fout.write(i);
    }
    sis.close();
    fout.close();
    fin.close();
    fin.close();

  }
}
```

Java IO: Readers and Writers

- کلاسهای Reader و Writer مشابه دو کلاس InputStream و OutputStream هستند .
- تفاوت آن است که Reader و Writer مبتنی بر کاراکتر می‌باشند و به هدف خواندن و نوشتن متن به کار می‌روند.
- اما دو کلاس InputStream و OutputStream مبتنی بر بایت هستند.

Reader

- Reader یک کلاس انتزاعی برای همه کلاسهای Reader در API جاوا می‌باشد.
- زیرکلاسهایی نظیر BufferedReader، InputStreamReader، StringReader و PushbackReader از Reader ارثبری می‌کنند.
- مثالی ساده :

```
Reader reader = new FileReader("c:\\data\\myfile.txt");
 int data = reader.read();
while(data != -1){
 char dataChar = (char) data;
 data = reader.read(); }
```

- توجه کنید که InputStream در هر لحظه یک بایت برمی‌گرداند که مقدارش بین 0 تا 255 است (۱- هم انتهای جریان را مشخص می‌کند)
- کلاس Reader یک کاراکتر را در هر لحظه برمی‌گرداند که مقداری بین 0 تا 65535 (۱- هم انتهای جریان را مشخص می‌کند) دارد.
- این به معنای آن است که در هر لحظه دو بایت می‌خواند.

کلاس FileReader

- کلاس FileReader برای خواندن داده‌های کاراکتری از یک فایل استفاده می‌شود.
- متد read() در این کلاس یک مقدار int برمی‌گرداند که حاوی مقدار کاراکتری کاراکتر خوانده شده است. اگر read() مقدار 1- را برگرداند یعنی داده بیشتَری در FileReader برای خواندن وجود ندارد و می‌توانیم آن را close() کنیم.

```
import java.io.*;
class Simple{
 public static void main(String args[]) {

  FileReader fr=new FileReader("abc.txt");
  int i;
  while((i=fr.read())!=-1)
  System.out.println((char)i);

  fr.close();
 }
}
```

```
Output:my name is sachin
```

Writer

- Writer یک کلاس انتزاعی برای همه کلاسهای writer در API جاوا می‌باشد.
- زیرکلاسهایی نظیر BufferedWriter و PrintWriter از Writer ارثبری می‌کنند.
- مثالی ساده :

```
Writer writer = new FileWriter("c:\\data\\file-output.txt");
writer.write("Hello World Writer");
 writer.close();
```

کلاس FileWriter

- کلاس FileWriter برای نوشتن داده‌های کاراکتری درون یک فایل استفاده می‌شود. شرکت میکروسیستم Sun پیشنهاد کرده در مواردی که فایلهای مورد استفاده متنی هستند، از همین کلاس و کلاس FileReader به ترتیب برای نوشتن و خواندن داده‌ها استفاده شود و کلاسهای FileInputStream و FileOutputStream مورد استفاده قرار نگیرند.

```
import java.io.*;
class Simple{
 public static void main(String args[]){
  try{
   FileWriter fw=new FileWriter("abc.txt");
   fw.write("my name is sachin");
   fw.flush();

   fw.close();
  }catch(Exception e){System.out.println(e);}
  System.out.println("success");
 }
}
```

```
Output:success...
```

Append to file

```
try
{
   String filename= "MyFile.txt";
   FileWriter fw = new FileWriter(filename,true); //the true will append the
     new data
   fw.write("add a line\n");//appends the string to the file
   fw.close();
}
catch(IOException ioe)
{
   System.err.println("IOException: " + ioe.getMessage());
}
```