# 第十七章 二进制I/O

## 17.1 引言

1.文本可以分为文本或二进制的。进行处理（读取、创建或修改）的文件成为文本文件。所有的其他文件成为二进制文件。例如：Java源程序存储在文本文件中，可以使用文本编辑器读取，而Java类是二进制文件，由Java虚拟机读取。二进制文件的优势在于它的处理效率比文本文件高。

## 17.2 在Java中如何处理文本I/O

1.要点提示：使用Scanner类读取文本数据，使用PrintWriter类写文本数据。Java中有许多用于各种目的的I/O类。通常，可以将他们分为输入类和输出类。输入类包含读数据的方法，而输出类包含写数据的方法。PrintWriter是一个输出类的例子，而Scanner是一个输入类的例子。

## 17.3 文本I/O与二进制I/O

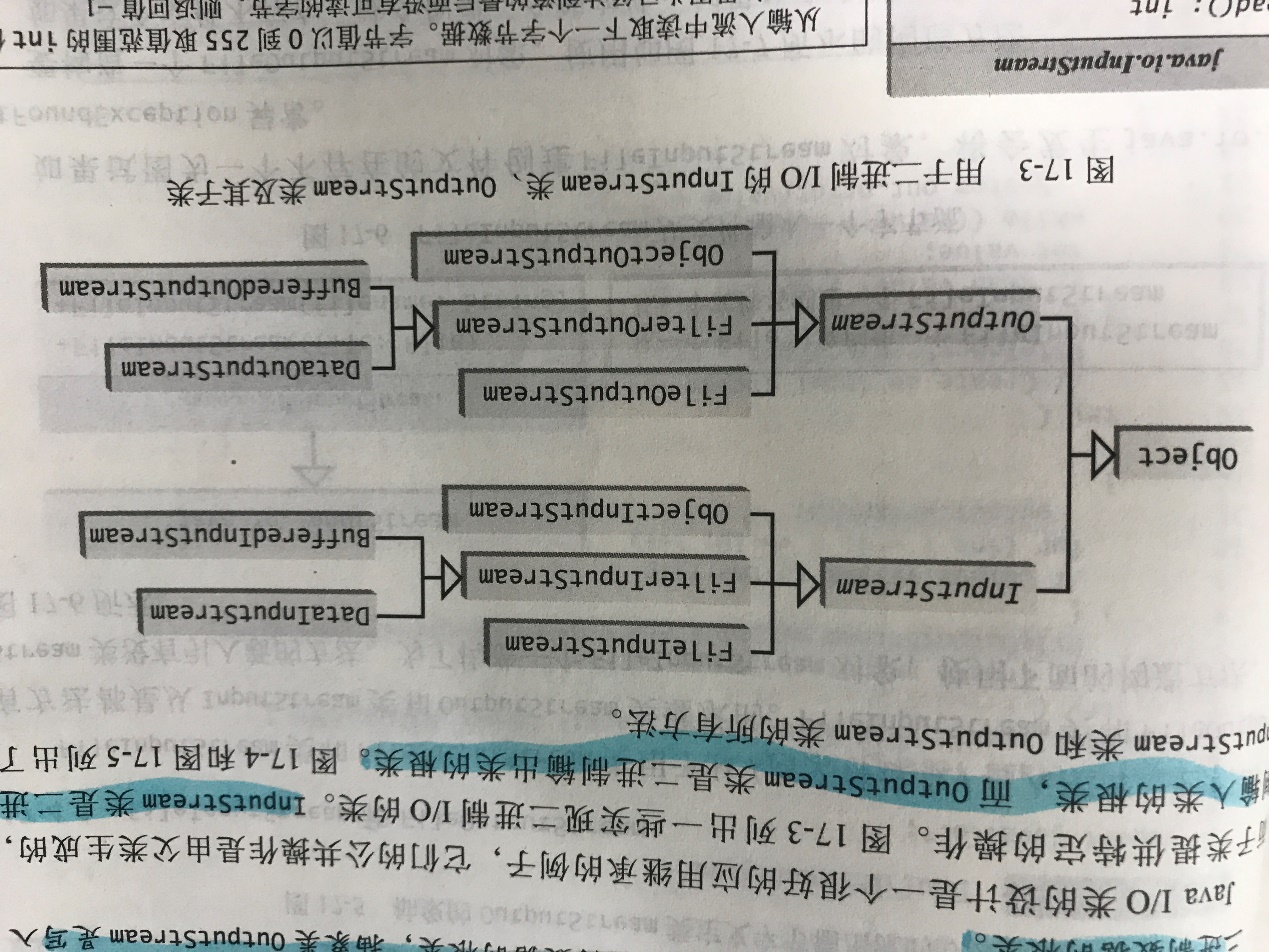
1.要点提示：二进制I/O不涉及编码和解码，因此比文本I/O更高效。计算机并不区分二进制文件与文本文件。所有的文件都是以二进制的形式存储的，因此，从本质上说，所有的文件都是二进制文件。文本I/O建立在二进制I/O的基础上，它能提供一层抽象，用于字符层次的编码和解码，对于文本I/O来讲，编码和解码是自动进行的。在写入一个字符时，Java虚拟机会将统一码转换成文件指定的编码；而在读取字符时，将文件指定的编码转换成统一码。

2.二进制I/O不需要转码和解码。所以，它比文本I/O效率高。二进制的文件与主机的编码方案无关，因此，它是可移植的。在任何机器上的Java程序可以读取Java程序所创建的二进制文件。这就是为什么Java类文件存储为二进制文件的原因。Java类文件可以在任何具有虚拟机的机器上运行。

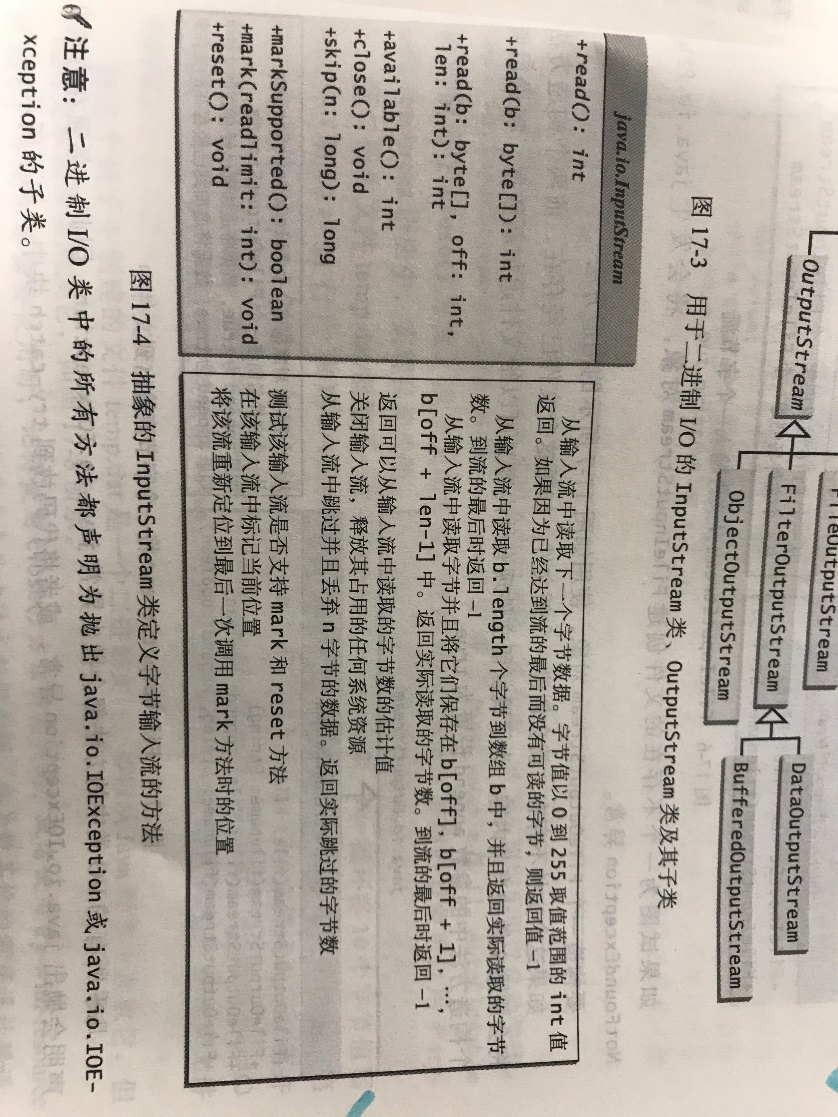
3.注意：为了保持一致性，本书使用扩展名.txt来命名文本文件，使用.dat来命名二进制文件。

## 17.4 二进制I/O类

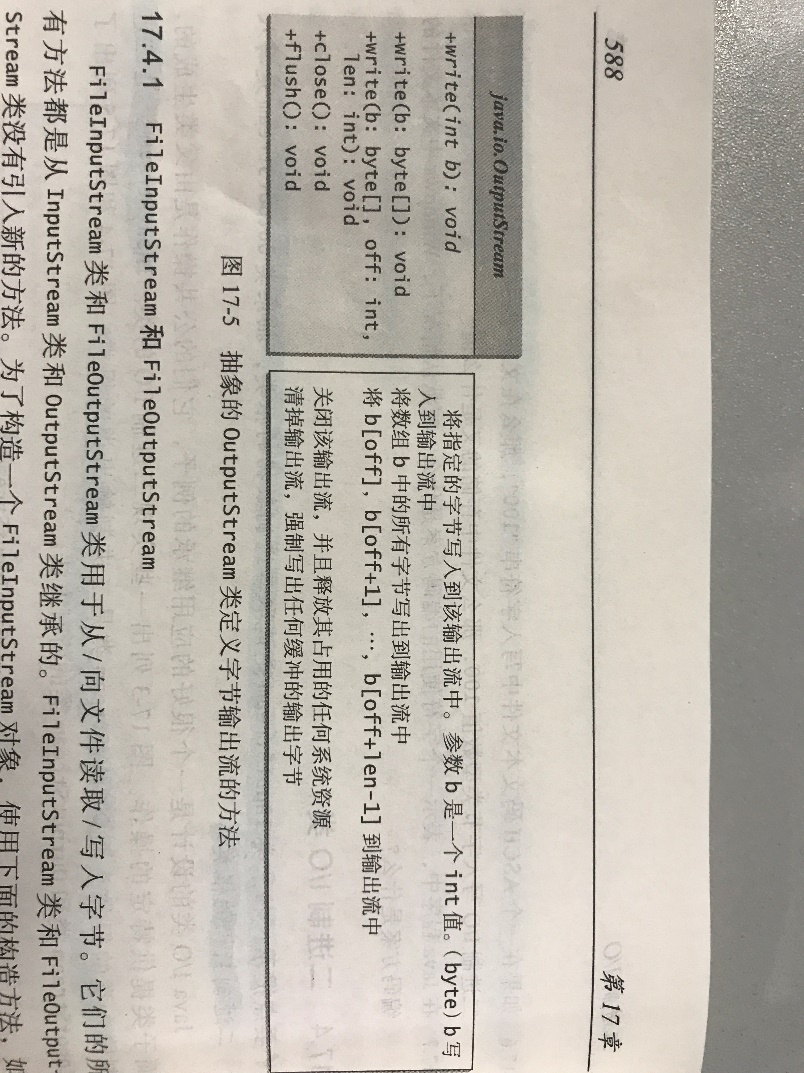
1.要点提示：抽象类InputStream是读取二进制数据的根类，抽象类OutputStream是写入二进制数据的根类。下图是用于二进制I/O的InputStream类、OutputStream类以及子类



下图是抽象的InputStream类定义字节输入流的方法



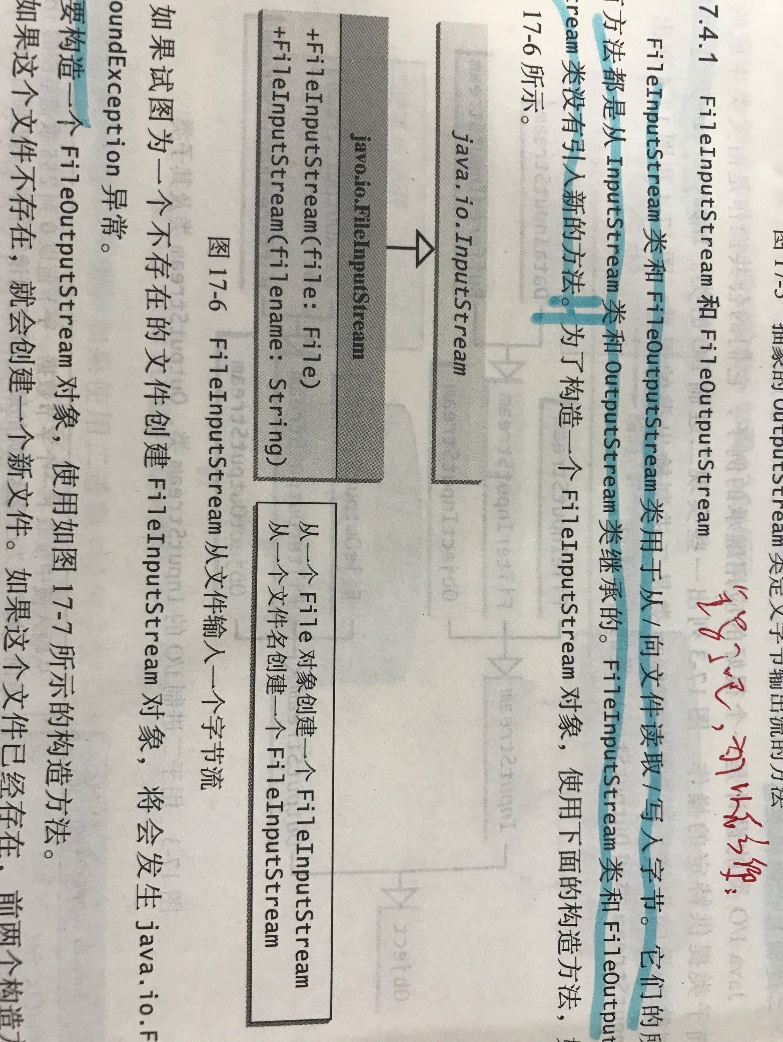
下图是抽象的OutputSream类定义字节输出流的方法



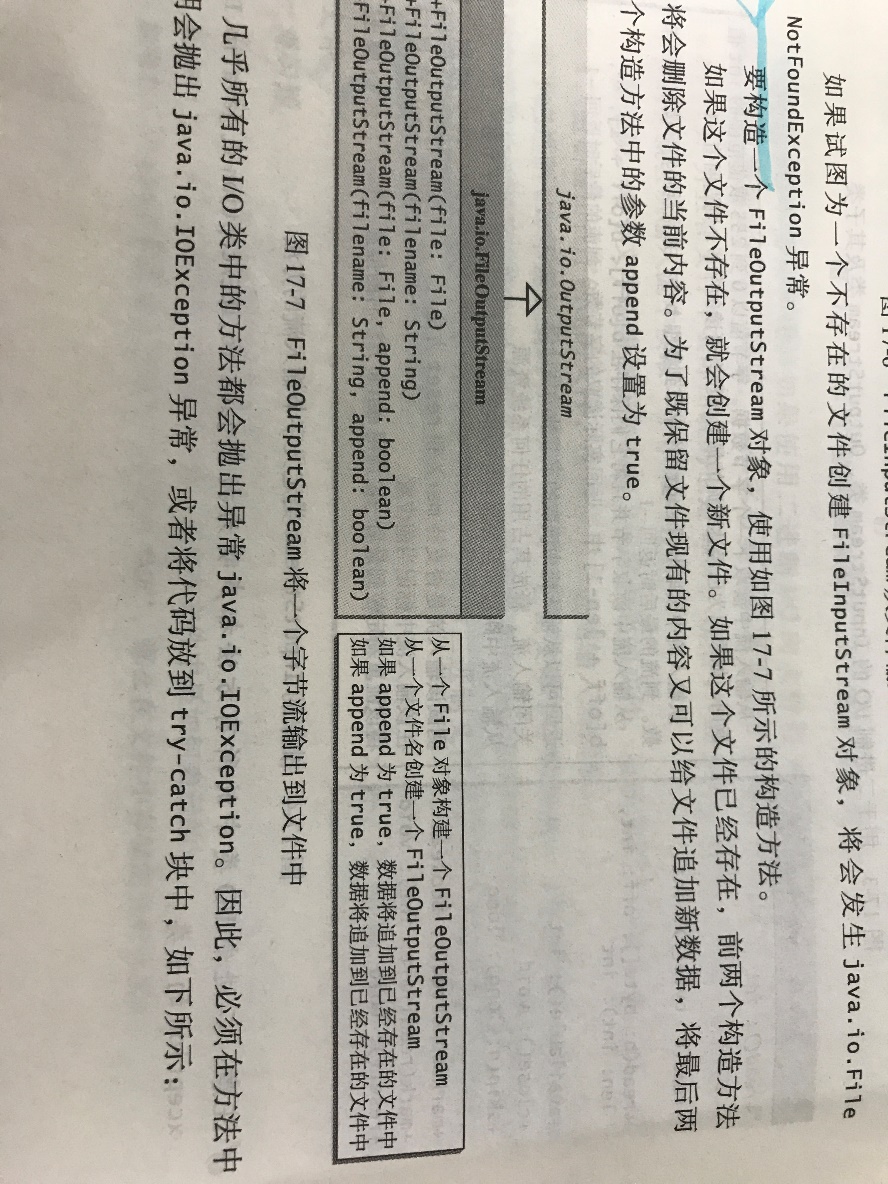
### 17.4.1 FileInputStream 和 FileOutputStream

1.FileInputStream类和FileOutputStream类用于从/向文件读取/写入字节。他们的方法都是从InputStream类和OutputStream类继承的。FileInputStream类和FileOutputStream类没有引入新的方法。

2.为了构造一个FileInputStream对象，使用下面的构造方法，如图



3.要构造一个FileOutputStream对象，使用如图所示的构造方法。如果这个文件不存在，就会创建一个新的文件。如果这个文件已经存在，前两个构造方法将会删除文件的当前内容。为了既保留文件的现有内容又可以给文件追加新数据，将最后两个构造方法中的参数append设置为true。



4.以下程序使用二进制I/O将从1到10的10个字节值写入一个名为temp.dat的文件，再将它们从文件中读出来。



当流不再需要使用时，记得使用close( )方法将其关闭，或者使用try-with-resources语句自动关闭。不关闭流可能会在输出文件中造成数据受损，或导致其他的程序设计错误。

5.FileInputStream类的实例可以作为参数去构造一个Scanner对象，而FileOutputStream类的实例可以作为参数构造一个PrinterWriter对象。可以创建一个PrintWriter对象来向文件中追加文本。如果temp.txt文件不存在，就会创建这个文件。如果temp.txt文件已经存在，就将数据追加到该文件中。

New PrintWriter（new FileOutputStream(“temp.txt”,true)）;

### 17.4.2 FilterInputStream和FilterOutputStream

1.过滤器数据流为某种目的过滤字节的数据流。基本字节输入流提供的读取方法read只能用来读取字节。如果要读取整数值、双精度值或字符串，那就需要一个过滤器来包装字节输入流。使用过滤器类就可以读取整数值、双精度值和字符串，而不是字符或字节。FilterInputStream类和FilterOutputStream类是过滤数据的基类。需要处理基本数值类型时，就使用DataInputStream类和DataOutputStream类来过滤字节。

### 17.4.3 DataInputStream和DataOutputStream

1.DataInputStream从数据流读取字节，并将它们转换为合适的基本类型值或字符串。DataOutputStream将基本类型的值或字符串转换为字节，并将字节输出到数据流。

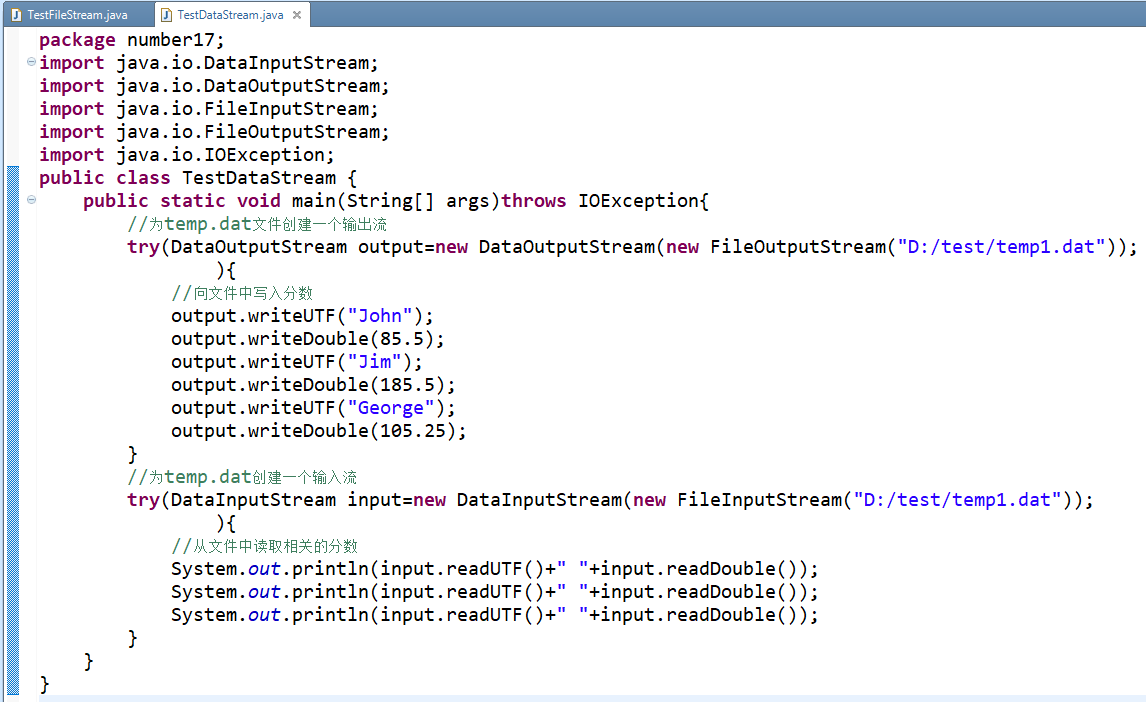
2.DataInputStream实现了定义在DataInput接口中的方法来读取基本数据类型值和字符串。DataOutputStream实现了定义在DataOutput接口中的方法来写入基本数据类型和字符串。基本类型的值不需要做任何转换就可以从内存复制到数据流。

3.二进制I/O中的字符与字符串

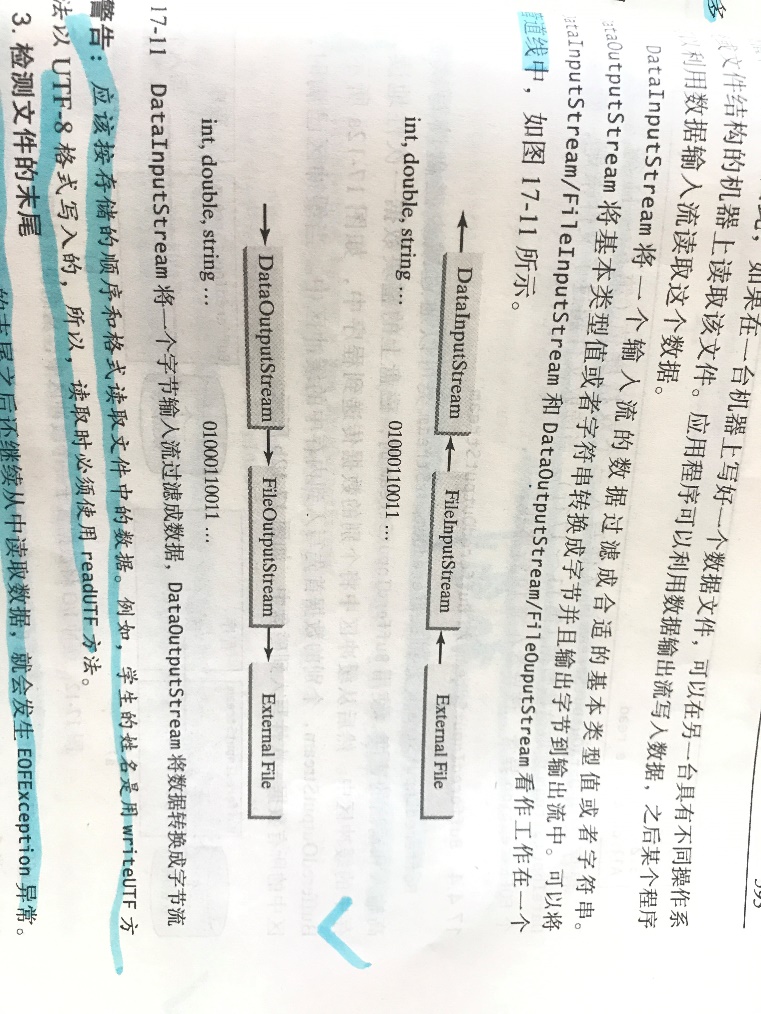
UTF-8是一种编码方案，它允许系统可以同时操作统一码及ASCII码。大多数操作系统使用ASCII码，Java使用统一码。

UTF-8格式具有存储每个ASCII码就节省一个字节的优势，因为统一码字符的存储需要两个字节，而在UTF-8格式中的ASCII码字符仅占一个字节。如果一个长字符串的大多数字符都是普通的ASCII字符，采用UTF-8格式存储更加高效。

4.以下程序将学生的名字和分数写入名为temp.dat的文件中，然后又将数据从这个文件中读取出来。



5.Data和File的数据流“管道线”。



6.应该按照存储的顺序和格式读取文件中的数据。例如，学生的姓名是用writeUTF方法以UTF-8格式写入的，所以读取时必须使用readUTF方法。

7.检测文件的末尾。（通过异常来检测）

如果达到InputStream的末尾之后还继续从中读取数据，就会发生EOFException异常，这个异常可以用来检测是否达到文件末尾。如下示例程序：



说明：程序使用DataOutputStream向文件写入三个双精度值，然后使用DataInputStream读取这些数据。当读取文件超过了文件末尾，就会抛出一个EOFException异常。

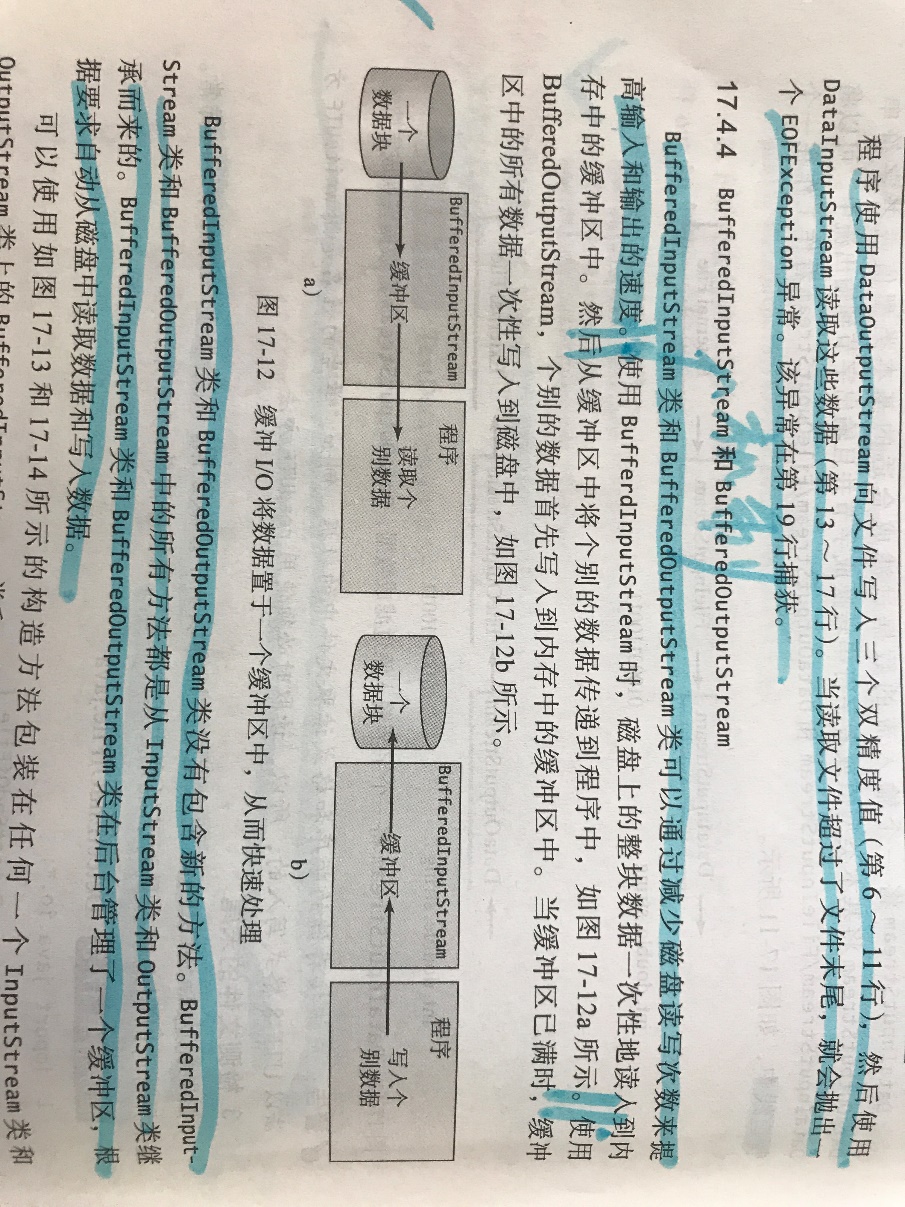
### 17.4.4 BufferedInputStream和BufferedOutputStream

1.BufferedInputStream类BufferedOutputStream类可以通过减少磁盘读写次数来提高输入和输出速度。

2.机制：（以Java为参考对象，可以这样理解？）

使用BufferedInputStream时，磁盘上的整块数据一次性的读入到内存中的缓冲区中，然后从缓冲区中将个别数据传递到程序中，如图a所示；

使用BufferedOutputStream，个别的数据首先写入到内存中的缓冲区中，当缓冲区已满时，缓冲区中的数据一次性写入到磁盘中，如图b所示：



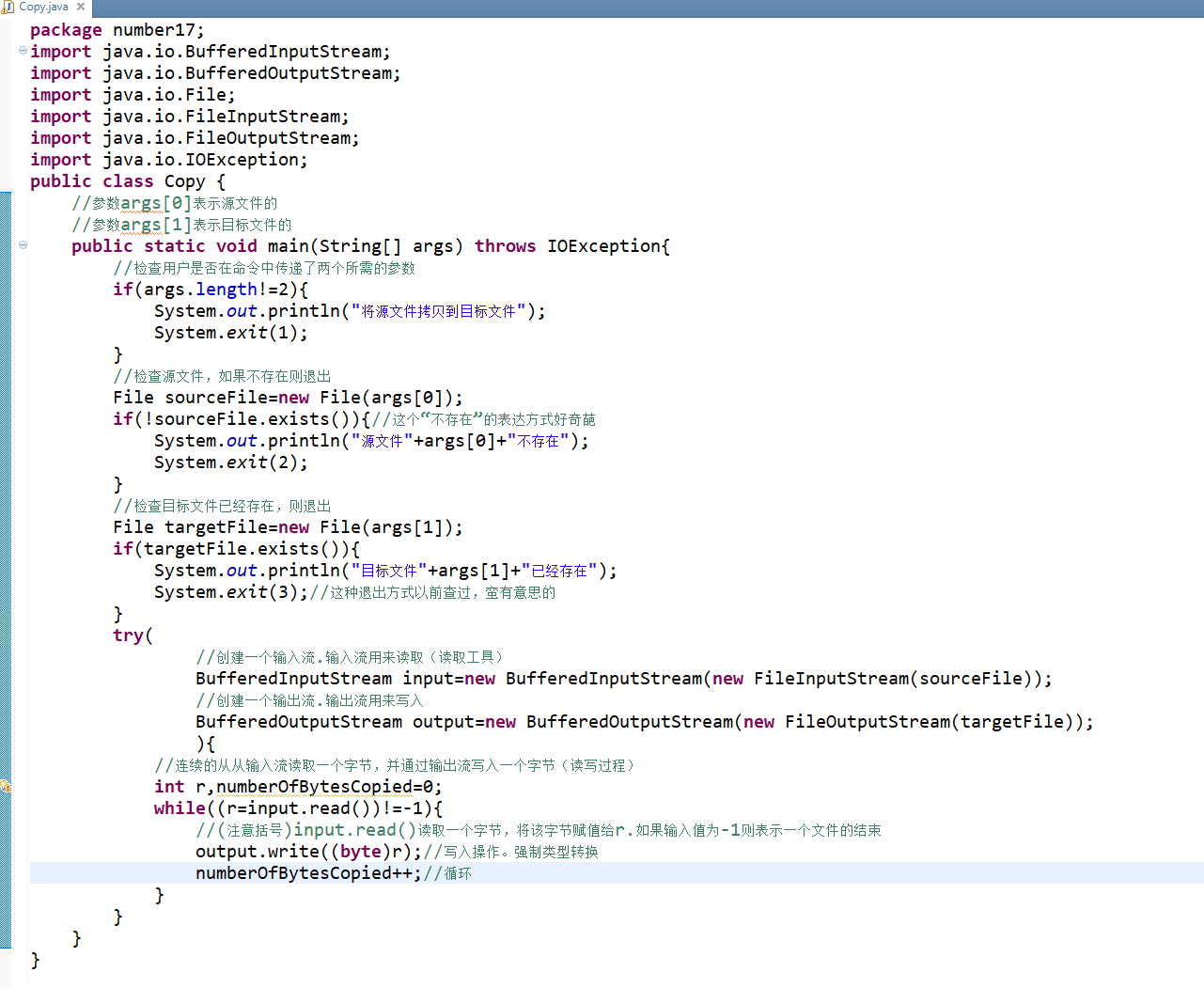
3. BufferedInputStream类和BufferedOutputStream类没有包含新的方法。BufferedInputStream和BufferedOutputStream中的所有方法都是从InputStream和OutputStream类中继承而来的。BufferedInputStream类和BufferedOutputStream类在后台管理一个缓冲区，根据要求自动从磁盘中读取数据和写入数据。

4.如果没有指定缓冲区的大小，默认的大小是512个字节。应该总是使用缓冲区I/O来加速输入和输出。对于小文件，我们可能注意不到性能的提升。但是，对于超过100MB的大文件，我们将会看到使用缓冲的I/O带来的实质性的性能提升。

17.5 示例学习：复制文件

1.编写一个让用户复制文件的程序。该程序将源文件复制到目标文件，然后显示这个文件中的字节数。如果源文件不存在，或者目标文件已经存在，程序应该给用户相应的提示。

要把源文件的内容复制到目标文件，不管文件内容如何，使用输入流从源文件中读出字节，并且使用输出流将字节写入目标文件比较合适。为源文件创建一个InputFilesStream对象，为目标文件创建一个OutputFilesStream对象。使用read（）方法从输入流中读取一个字节，使用write（b）方法将一个字节写入输出流。使用BuffferedInputStream类和BufferedOutputStream类来提高执行效率。



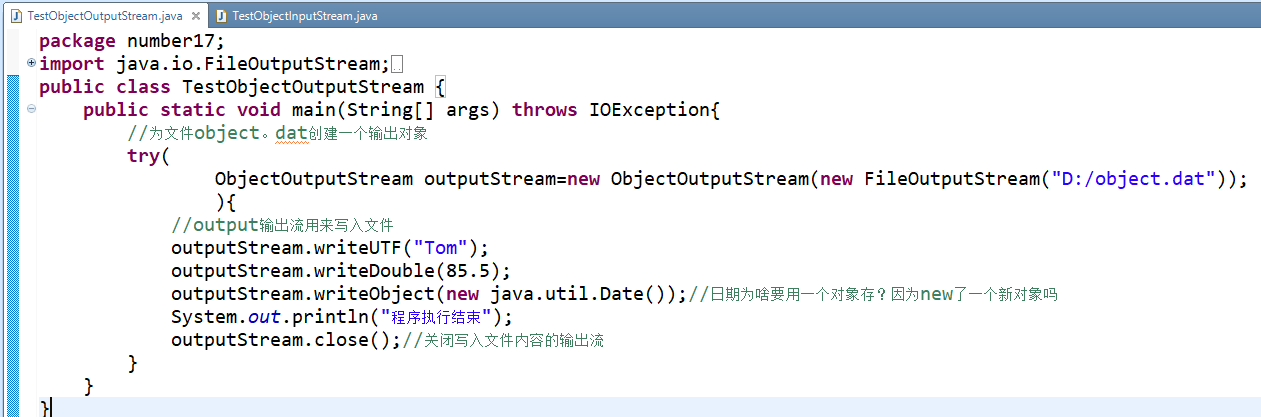
## 17.6 对象I/O

1.要点提示：ObjectInputStream类和ObjectOutputStream类可以用于读/写可序列化的对象。

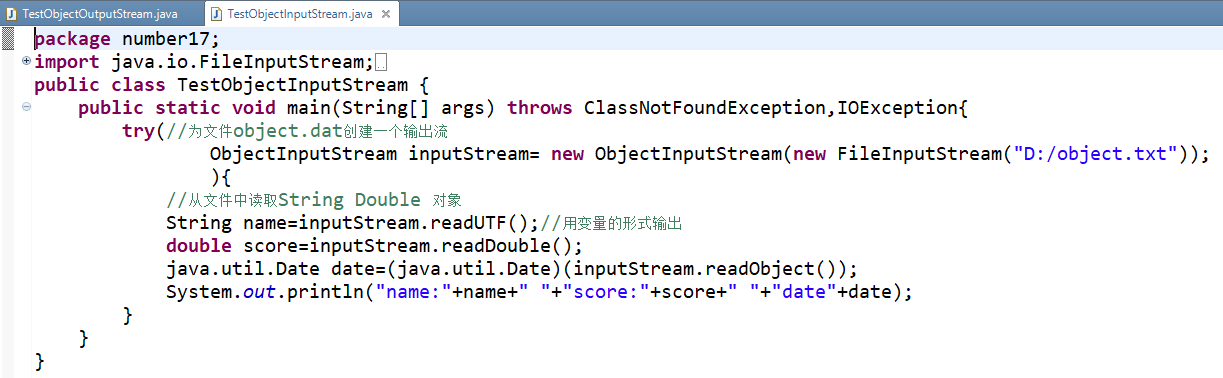
2.DataInputStream类和DataOutputStream类可以实现基本数据类型与字符串的输入和输出。而ObjectInputStream类和ObjectOutputStream类除了可以实现基本数据类型与字符串的输入和输出外，还可以实现对象的输入和输出。由于ObjectInputStream类和ObjectOutputStream类包含DataInputStream类和DataOutputStream类的所有功能，所以，完全可以用ObjectInputStream类和ObjectOutputStream类替代DataInputStream类和DataOutputStream类。

3. ObjectOutputStream类扩展InputStream类；ObjectOutputStream扩展OutputStream。

4.以下程序将学生的姓名、分数和当前日期写入名为object.dat的文件中。



从文件object.dat中读回数据

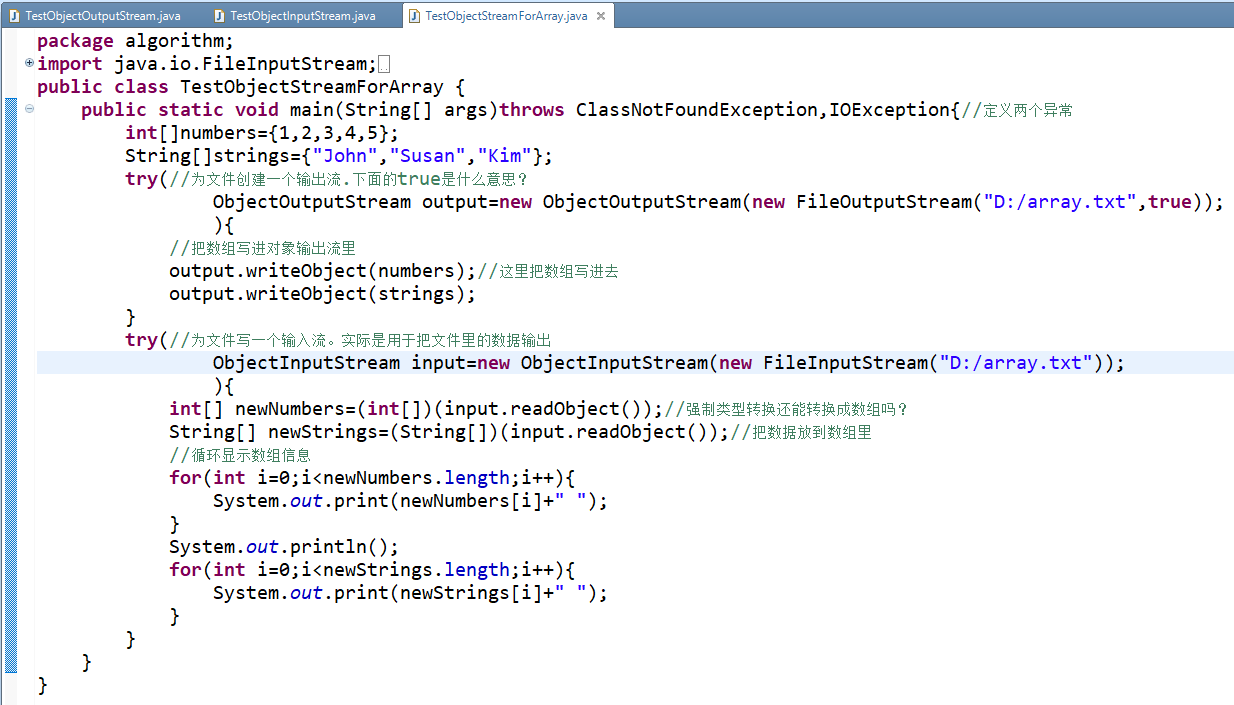


### 17.6.1 Serializable接口（没看懂，所以简单摘要了）

1.并不是每一个对象都可以写到输出流，可以写入输出流的对象成为“可序列化”得。

### 17.6.2序列化数组

1.如果数组中的所有元素都是可序列化的，这个数组就是可序列化的。一个完整的数组可以用writeobject方法存入文件，随后用readobject方法恢复。下列程序存储由五个int元素构成的数组和由三个字符串构成的数组，然后将他们从文件中读取出来显示在控制台上。



（里面是小的，外面是大的。里面是文件，外边是对象）

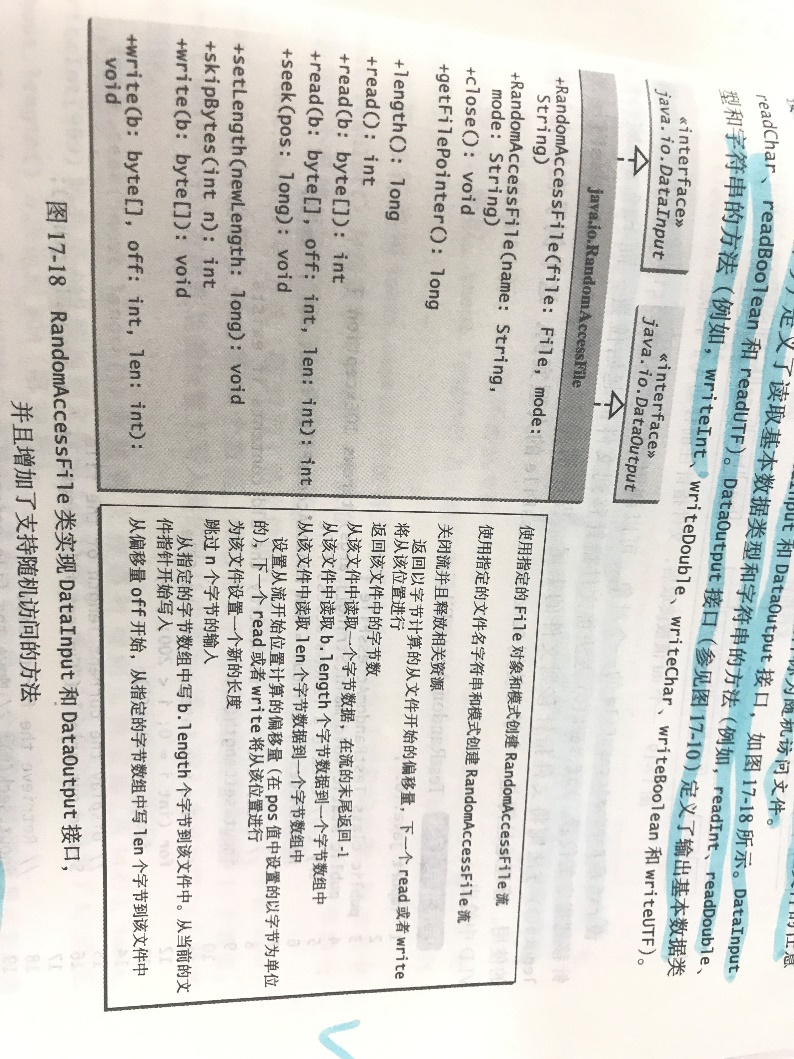
## 17.7随机访问文件

1.要点提示：Java提供了RandomAccessFile类，允许从文件的任何位置进行数据的读写。

2.到现在为止，所使用的所有的流都是只读的或只写的。这些流称为“顺序流”。

顺序访问文件：使用顺序流打开的文件称为“顺序访问文件”。顺序访问文件的内容不能更新。然而，经常需要修改文件，Java提供了RandomAccessFile类，允许在文件的任意位置上进行读写。使用RandomAccessFile类打开的文件称为随机访问文件。

3.RandomAccessFile类实现了DataInput和DataOutput接口，如下图所示。DataInput接口定义了读取基本数据类型和字符串的方法。DataOutput接口定义了输出基本数据类型和字符串的方法。



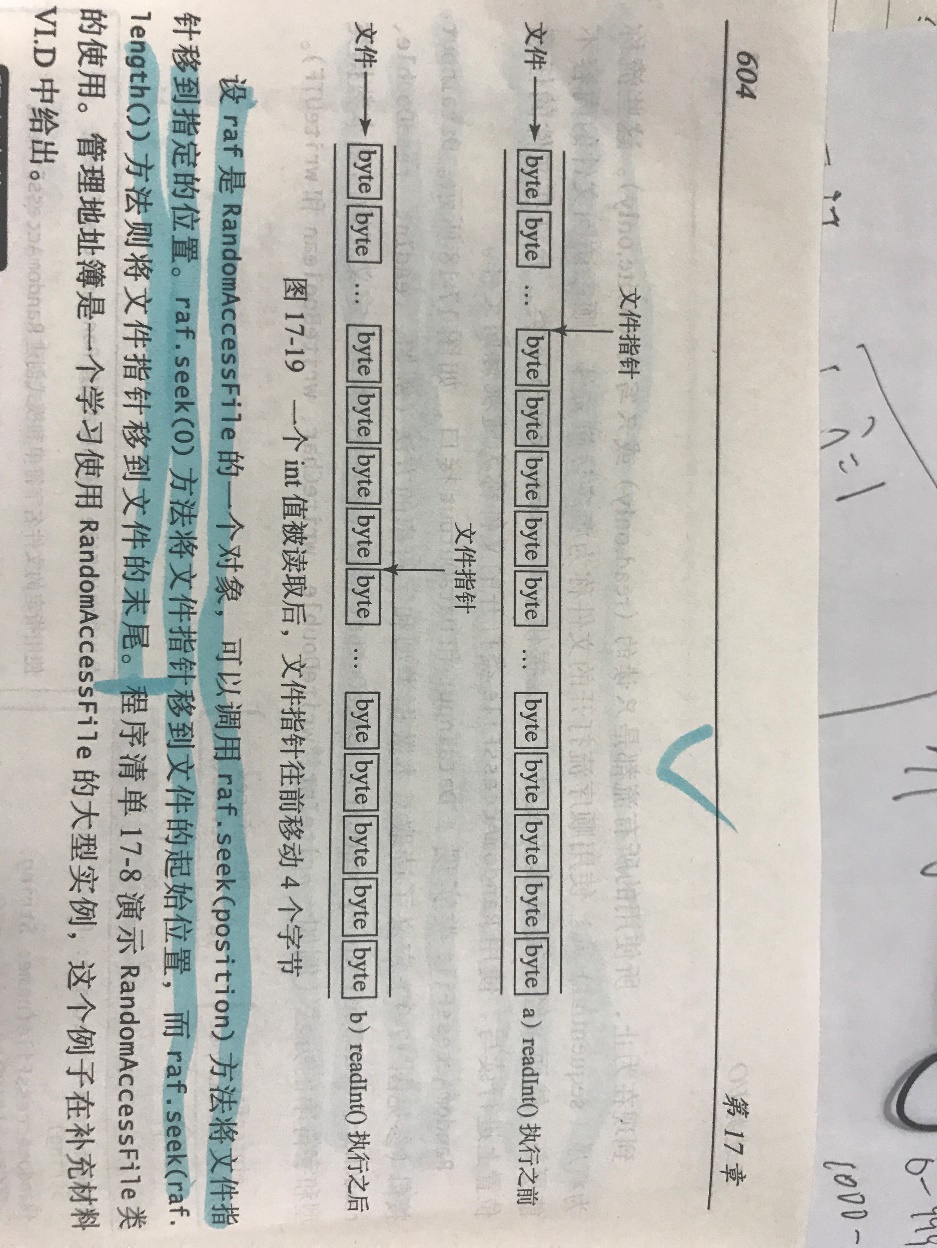
4.当创建一个RandomAccessFile时，可以指定两种模式（“r”或“rw”）之一。模式“r”表示这个数据流是只读的，模式“rw”表示这个数据流既允许读也允许写。例如，下面的语句创建一个新的数据流raf，它允许程序对文件test.dat进行读取和写入：

RandomAccessFile raf =new RandomAccessFile(“test.dat”,”rw”);

如果文件test.dat已经存在，则创建raf以便访问这个文件；如果test.dat文件不存在，则创建一个名为test.dat的新文件，再创建raf来访问这个新文件。Raf.length()方法返回在给定时刻test.dat文件的字节数。如果向文件中添加新数据，raf.length()就会增加。

5.提示：如果不想改动文件，就将文件的“r”模式打开。这样做可以防止不经意间改动文件。

6.随机访问文件是由字节序列组成的。一个称为“文件指针”的特殊标记定位这些字节中某个字节的位置。文件的读写操作就是在文件指针所指的位置上进行的。打开文件时，文件指针置于文件的起始位置。在文件中进行读写数据后，文件指针就会向前移动到下一个数据项。例如，如果使用readInt()方法读取一个int数据，Java虚拟机就会从文件指针出读取4个字节，现在，文件指针就会从它之前的位置向前移动四个字节，如下图所示：



设raf是RandomAccessFile的一个对象，可以调用raf.seek(position)方法将文件指针移到指定的位置。Raf.seek(0)方法将文件指针移到文件的起始位置，而raf.seek(raf.length())方法则将文件指针移动到文件的末尾。

7.下面是管理地址薄的大型实例，只有其中一部分，慢慢用心体会吧。（为什么跟课本的结果不一致？）

