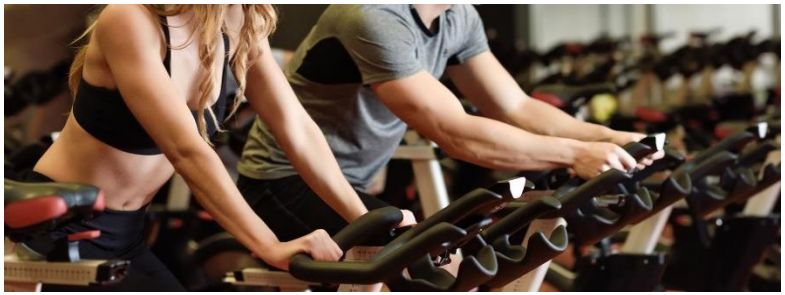
IECS273/274 [1111 3670/3671]
物件導向設計與實習

10#1 : I/O Streams in Java



01 Input/Output Streams

Input/Output Streams in Java

- Java **I/O** is used to *read* input data and *write* output data.
- Java uses the concept of **streams** to make I/O operations consistent. The java.io package contains all the classes required for *input and output operations*.
- A stream is a *sequence of data*, and in Java, a stream is composed of *bytes*. It's called a stream because it's like a continuous creek.
- There are three basic streams in Java, these streams are connected to the *console*.
  - System.out: standard output stream
  - System.in: standard input stream
  - System.err: standard error stream

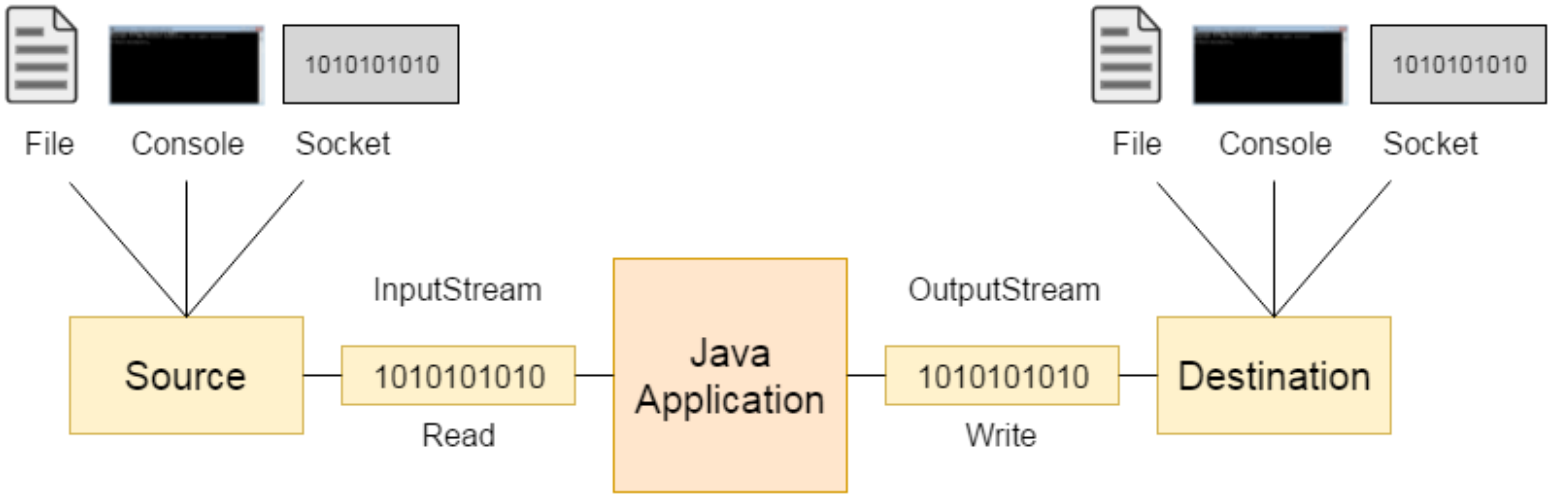
Input/Output Streams in Java

- Output and error **messages** sent to the console:

```
System.out.println("simple message");  // Output message
System.err.println("error message");  // Error message
```

- Input messages from the console:

```
int i=System.in.read(); // Returns the ASCII code of the first character.
System.out.println((char) i); // Print the entered characters.
```



取材自：https://www.javatpoint.com/java-io

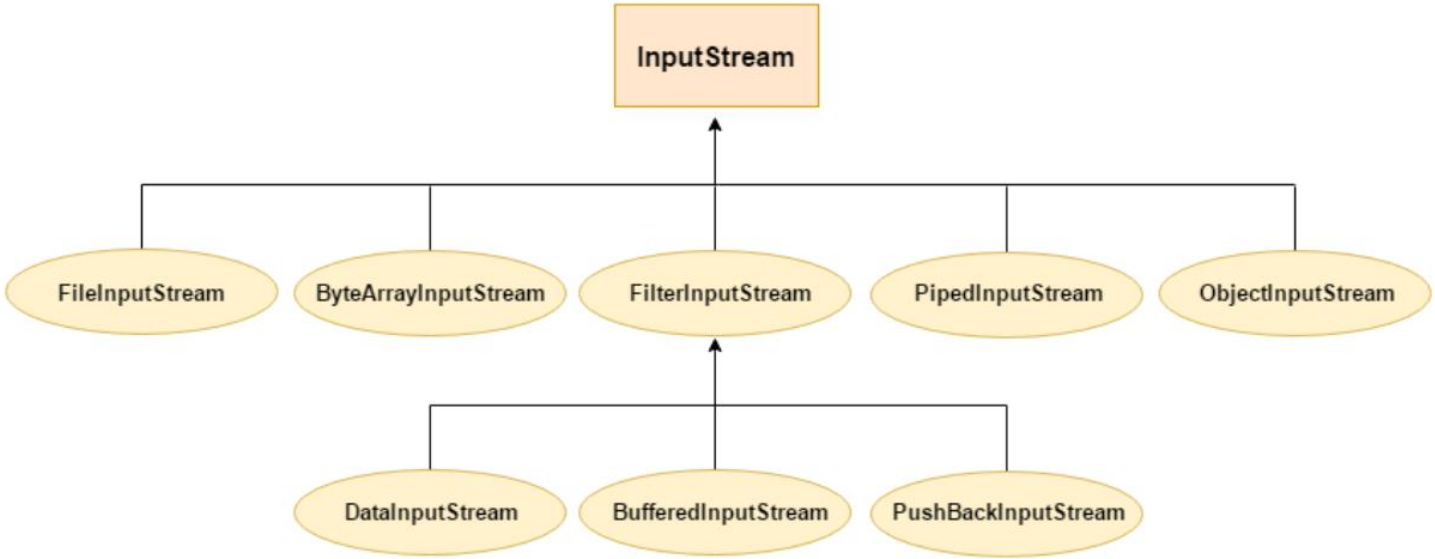
02

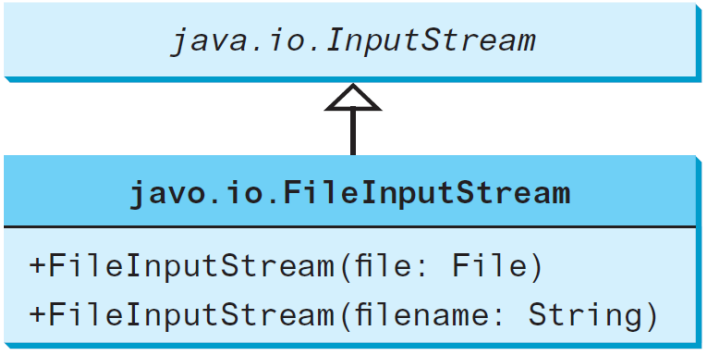
File I/O Stream Class

File Input Stream Class

- FileInputStream inputs a stream of bytes from a file.

InputStream Hierarchy





Creates a `FileInputStream` from a `File` object.
Creates a `FileInputStream` from a file name.

取材自：Y. D. Liang, *Introduction to Java Programming and Data Structures (11th Edition)*, Pearson

File Input Stream Class

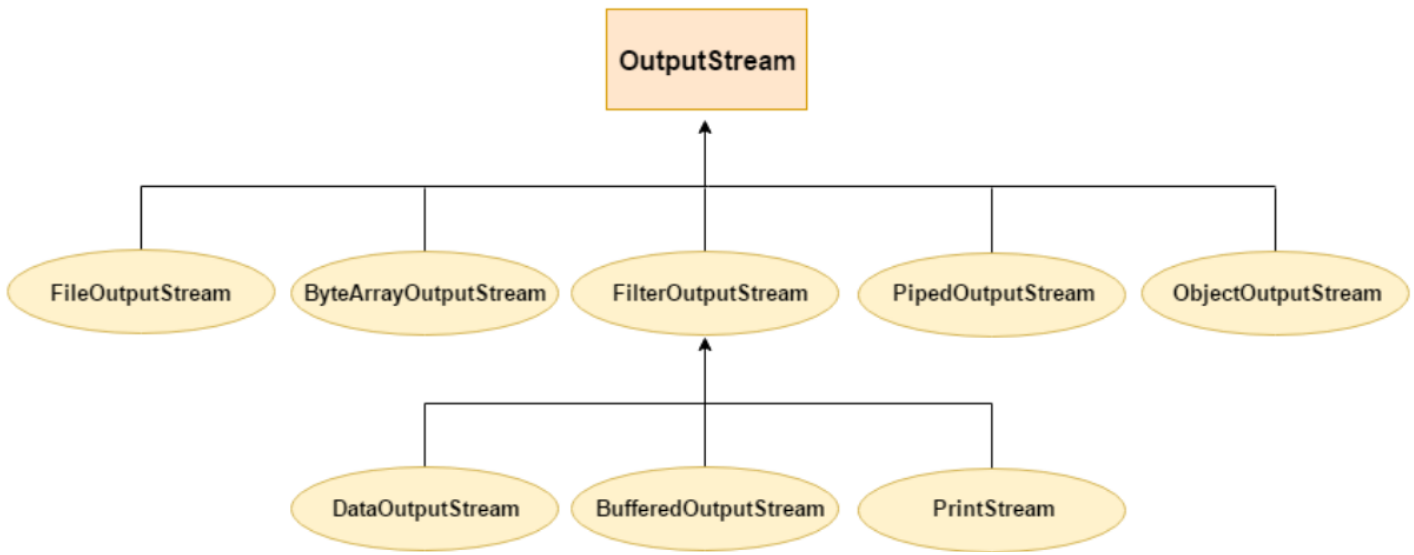
- Java FileInputStream class is an input stream that gets input bytes from a document.

  public class FileInputStream **extends** InputStream

  - Methods of FileInputStream class:
    - ✓ FileInputStream(String fName): create an input document with filename fName.
    - ✓ **int** available(): returns the estimated number of bytes that can be read from the input stream.
    - ✓ **int** read(): read data bytes from the input stream.
    - ✓ **int** read(**byte** [ ] data): read up to data.length bytes of data from the input stream.
    - ✓ **int** read(**byte** [ ] data, **int** offset, **int** length): read length bytes of data from the input stream data starting at offset.
    - ✓ **void** close(): close the input stream document.

File Output Stream Class

- FileOutputStream outputs a stream of bytes to a file.



java.io.OutputStream

java.io.FileOutputStream

```
+FileOutputStream(file: File)
+FileOutputStream(filename: String)
+FileOutputStream(file: File, append: boolean)
+FileOutputStream(filename: String, append: boolean)
```

Creates a `FileOutputStream` from a `File` object.
Creates a `FileOutputStream` from a file name.
If `append` is true, data are appended to the existing file.
If `append` is true, data are appended to the existing file.

取材自：Y. D. Liang, *Introduction to Java Programming and Data Structures (11th Edition)*, Pearson

File Output Stream Class

- The Java FileOutputStream class is an output stream that writes data to a document.

  public class FileOutputStream **extends** OutputStream

  - Methods of FileOutputStream class:
    - ✓ FileOutputStream(String fName): create an output document with the filename fName.
    - ✓ **void** write(**int** data): write bytes data to the document output stream.
    - ✓ **void** write(**byte** [ ] data): write data.length bytes from the byte array to the document output stream.
    - ✓ **void** write(**byte** [ ] data, **int** offset, **int** length): write length bytes starting at offset from the byte array data to the document output stream.
    - ✓ **void** close(): close output stream document.

IOStreamFile.java

```
import java.io.FileOutputStream; // File output stream package.
import java.io.FileInputStream;  // File input stream package.
public class IOStreamFile {

  public static void main(String[] args) {
    try {
      String dataOut = "FengChia University"; // The string to be output.
      FileOutputStream fileOut=new FileOutputStream("D:\\test.txt"); // Create a file output stream with name “test.txt".
      fileOut.write(dataOut.getBytes()); // Convert string to byte array and write to file.
      fileOut.close(); // Close the file output stream. 。
      System.out.println("File test.txt is written completely."); // Print a message.
      System.out.println("Output text: " + dataOut); // Print the output text.
    }
    catch (Exception e) {System.out.println(e);} // Handle exception.

    System.out.println("============================");

     try {
      FileInputStream fileIn=new FileInputStream("D:\\test.txt"); // Create a file input stream with name “test.txt"
      int n = fileIn.available(); // Get the number of bytes.
      char dataIn[] = new char[n]; // Create the byte array for the input data. 。
      int i = 0; // Loop variable.
```

IOStreamFile.java

```
        int c; // A character to be read.

        System.out.println("Length of the input file: " + n);   // Print the file length.
        // When the file ends, the bytes read is EOF (end of file), with a value of -1.
       // Each loop reads one character and stores it in data[]; until the end of file.
        while ((c = fileIn.read())!=-1) dataIn[i++] = (char) c;
        fileIn.close(); //  Close the file input stream.
        System.out.println("File test.txt is read completely."); // Print a message.
        System.out.print("Input text: ");
        for (i=0; i<n; i++) System.out.print(dataIn[i]); // Print the input text.
       }
       catch (Exception e) {System.out.println(e);} // Handle exception.
   }
}
```

IOStreamFile.class

```
File test.txt is written completely.
Output text: FengChiaUniversity
============================
Length of the input file: 19
File test.txt is read completely.
Input text: FengChia University
```

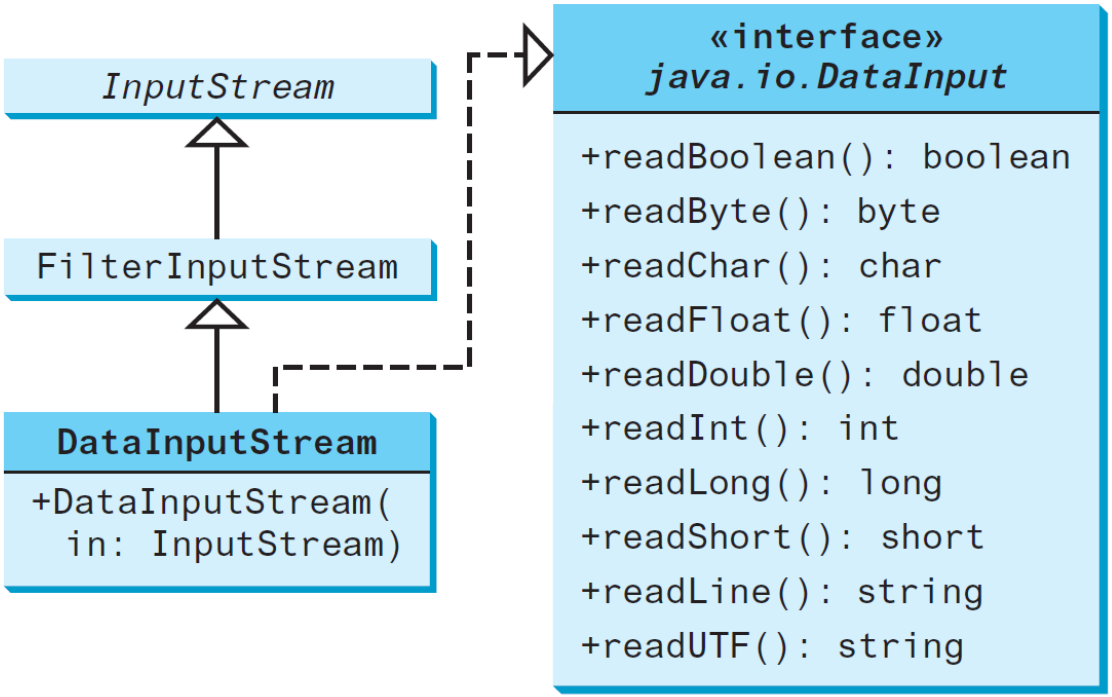
03 Data I/O Stream Class

Filter Input/Output Stream Class

- Filter streams are streams that filter bytes for some purpose.
- The basic byte input stream provides a read method that can be used only for reading bytes. If you want to read integers, doubles, or strings, you need a filter class to wrap the byte input stream. Using a filter class enables you to read integers, doubles, and strings instead of bytes and characters.
- FilterInputStream and FilterOutputStream are the base classes for filtering data.

Data Input Stream Class

- DataInputStream filters an input stream of bytes into primitive data-type values and strings.



Reads a `Boolean` from the input stream.	
Reads a `byte` from the input stream.	
Reads a character from the input stream.	
Reads a `float` from the input stream.	
Reads a `double` from the input stream.	
Reads an `int` from the input stream.	
Reads a `long` from the input stream.	
Reads a `short` from the input stream.	
Reads a `line` of characters from input.	
Reads a `string` in UTF format.	

取材自：Y. D. Liang, *Introduction to Java Programming and Data Structures (11th Edition)*, Pearson

Data Input Stream Class

- The Java DataInputStream class allows applications to read raw data from an input stream in a machine-independent manner.

  public class DataInputStream **extends** FilterInputStream
  implements DataInput

  - Methods of the DataInputStream class:
    - ✓ DataInputStream(FileInputStream): Use the file input stream, create the data input stream, the file name of the data input stream is specified when creating the file input stream. That is, Create the file input stream first, then create the data input stream.

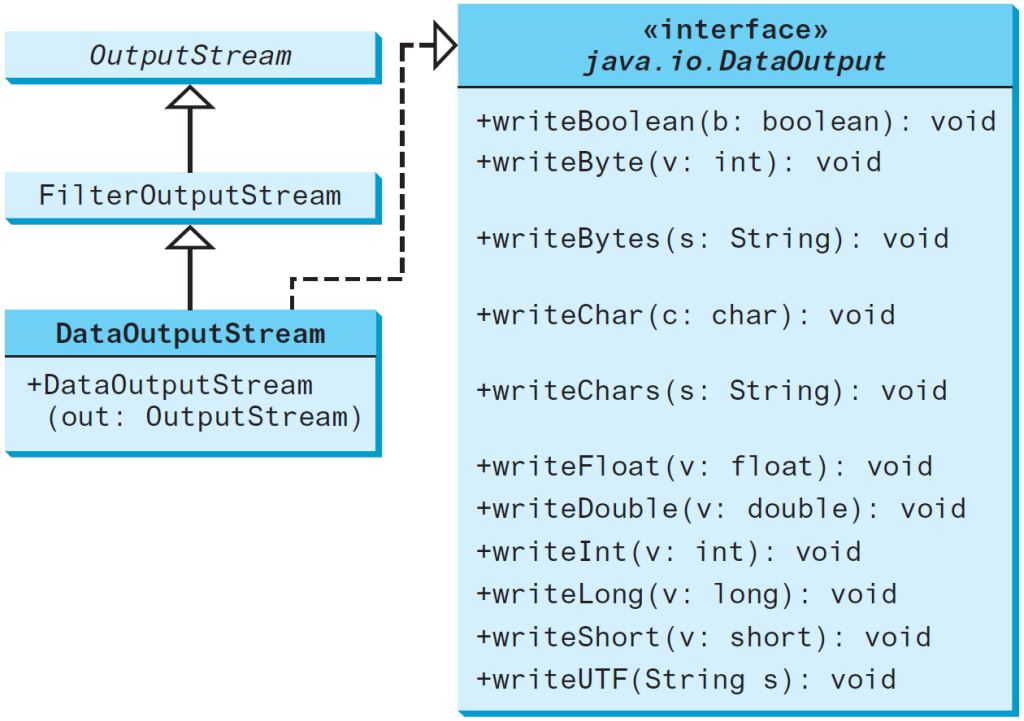
      FileInputStream fileIn = **new** FileInputStream("text.txt");
      DataInputStream dataIn = DataInputStream(fileIn);
    - ✓ **int** read(**byte** [ ] data): read byte array from input stream, return the length of byte array.
    - ✓ **int** read(**byte** [ ] data, **int** offset, **int** length): read length bytes of data from the input stream, store it in the data byte array starting at offset, and return the length of the read byte array.

Data Input Stream Class

- ✓ **byte** readByte(): read 1 byte from the input stream, convert it into a byte value, and return the byte value, which requires explicit casting.
- ✓ **byte** readUnsignedByte(): read 1 byte from the input stream, convert it into an unsigned byte value, and return the byte value, which requires explicit casting.
- ✓ **byte** readShort(): read 2 bytes from the input stream, convert it into a **short** integer value, and return the **short** integer value, which requires explicit casting.
- ✓ **byte** readUnsignedShort(): Read 2 bytes from the input stream, convert it into an *unsigned* **short** integer value, and return the **short** integer value, which requires explicit casting.
- ✓ **int** readInt(): read 4 bytes from the input stream, convert it into an **int** value, and return the integer value, which requires explicit casting.
- ✓ **int** readDouble(): Read 8 bytes from the input stream, convert it into a floating-point (**double**) value, and return the floating-point value, which requires explicit casting.
- ✓ **boolean** readBoolean(): Read 1 byte from the input stream, convert it to a **boolean** value, and return the **boolean** value, which requires explicit casting.

Data Output Stream Class

- DataOutputStream enables you to write primitive data-type values and strings into an output stream.



Writes a `Boolean` to the output stream.	
Writes the eight low-order bits of the argument `v` to the output stream.	
Writes the lower byte of the characters in a `string` to the output stream.	
Writes a character (composed of 2 bytes) to the output stream.	
Writes every character in the `string s` to the output stream, in order, 2 bytes per character.	
Writes a `float` value to the output stream.	
Writes a `double` value to the output stream.	
Writes an `int` value to the output stream.	
Writes a `long` value to the output stream.	
Writes a `short` value to the output stream.	
Writes `s string` in UTF format.	

取材自：Y. D. Liang, *Introduction to Java Programming and Data Structures (11th Edition),* Pearson

Data Output Stream Class

- The Java DataOutputStream class allows applications to write primitive Java data types to the output stream in a machine-independent manner ◦

  public class DataOutputStream **extends** FilterOutputStream
  implements DataOutput

  - Java applications typically use a data output stream to write data that can later be read by a data input stream.
  - Methods of the DataOutputStream class:
    - ✓ DataOutputStream(FileOutputStream): Using *the file output stream*, create a data output stream, and *the file name* of the data output stream is specified when creating the document output stream. That is, create a file output stream first, then create the data output stream.

      FileOutputStream fileOut = **new** FileOutputStream(“text.txt”);
      DataOutputStream dataOut = **new** DataOutputStream(fileOut);
    - ✓ **int** size(): Returns the number of bytes written to the data output stream so far.
    - ✓ **void** write(**int** b): Write a byte (the leftmost byte of integer b) to the data output stream.

Data Output Stream Class

- ✓ **void** write(**byte**[] data, **int** offset, **int** length): writes length bytes starting at offset from the byte array data to the data output stream.
- ✓ **void** writeBoolean(**boolean** v): write a **boolean** value to the data output stream as 1-byte value data.
- ✓ **void** writeChar(**int** v): write the leftmost 2 bytes of integer v as characters to the data output stream.
- ✓ **void** writeChars(String str): write a string as a sequence of characters to the data output stream.
- ✓ **void** writeByte(**int** v): write integer v as 1-byte value data to the data output stream.
- ✓ **void** writeBytes(String s): write a string as a sequence of bytes to the data output stream.
- ✓ **void** writeInt(**int** v): write the integer v as an **int** to the data output stream.
- ✓ **void** writeShort(**int** v): write the leftmost 2 bytes of the integer v as a **short** type to the data output stream.
- ✓ **void** writeLong(**long** v): write the integer v as a **long** to the data output stream.
- ✓ **void** writeUTF(String str): write a string to the data output stream using UTF-8 encoding in a portable way.
- ✓ **void** flush(): flush data output stream.

IOStreamData.java

```
import java.io.*;

public class IOStreamData {

  public static void main(String[] args)  throws IOException {
    OutputStream fileOut = new FileOutputStream("D:\\dataTest.txt");
    DataOutputStream dataOut = new DataOutputStream(fileOut);

    System.out.println("Target file: dataTest.txt");
    System.out.println(">>>> Write character sequence (bytes): \"FengChia University\"");
    dataOut.writeBytes("FengChia University");

    System.out.println(">>>> Write a short integer (short): 24581");
    dataOut.writeShort(24581);

    System.out.println(">>>> Write an integer (int):1234567890");
    dataOut.writeInt(1234567890);

    System.out.println(">>>> Write a floating point number (double)PI：3.14159...");
    dataOut.writeDouble(Math.PI);

    System.out.println(">>>> Write a boolean constant: true");
```

IOStreamData.java

```
dataOut.writeBoolean(true);

dataOut.close();
System.out.println("========================================");

InputStream fileIn = new FileInputStream("D:\\dataTest.txt");
DataInputStream dataIn = new DataInputStream(fileIn);
System.out.println("Source file: dataTest.txt");

byte bytes[] = new byte[19];
int n = dataIn.read(bytes);
System.out.print("<<<< Read byte sequence " + n + " bytes: ");
for (int i=0; i<n; i++) System.out.print((char) bytes[i]);
System.out.println();

System.out.println("<<<< Read a short integer (short): " + (short) dataIn.readShort());

System.out.println("<<<< Read an integer (int):" + (int) dataIn.readInt());

System.out.println("<<<< Read a floating point number (double) PI ： " + (double) dataIn.readDouble());
```

IOStreamData.java

```
        System.out.println("<<<< Read a boolean constant: " + (boolean) dataIn.readBoolean());

        dataIn.close();
    }
}
```

IOStreamData.class

```
Target file: dataTest.txt
>>>> Write character sequence (bytes): "FengChia University"
>>>> Write a short integer (short): 24581
>>>> Write an integer (int):1234567890
>>>> Write a floating point number (double)PI：3.14159...
>>>> Write a boolean constant: true
========================================
Source file: dataTest.txt
<<<< Read byte sequence 19 bytes: FengChia University
<<<< Read a short integer (short): 24581
<<<< Read an integer (int):1234567890
<<<< Read a floating point number (double) PI：3.141592653589793
<<<< Read a boolean constant: true
```

04 Buffered I/O Stream Class

Buffered Input/Output Stream Class

- BufferedInputStream/BufferedOutputStream can be used to speed up input and output byreducing the number of disk reads and writes.
  - Using BufferedInputStream, the whole block of data on the disk is read into the buffer in the memory once. The individual data are then loadedto your program from the.
  - Using BufferedOutputStream, the individual data are first written to the buffer in the memory. When the buffer is full, all data in the buffer are written to the disk once.

Buffered Input/Output Stream Class

- BufferedInputStream/BufferedOutputStream does not contain new methods.
- All the methods in BufferedInputStream/BufferedOutputStream are inherited from theInputStream/OutputStream classes. BufferedInputStream/BufferedOutputStream manages a buffer behind the scene and automatically reads/writes data from/to disk on demand.

05 Object I/O Stream Class

Object Input/Output Stream Class

- ObjectInputStream/ObjectOutputStream classes can be used to read/write serializable objects. They enable you to perform I/O for primitive-type values, strings and objects.
- Since ObjectInputStream/ObjectOutputStream contains all the functions of DataInputStream/DataOutputStream, you can replace DataInputStream/DataOutputStream completely with ObjectInput Stream/ObjectOutputStream.
- Not every object can be written to an output stream. Objects that can be so written are said to be serializable.
  - A serializable object is an instance of the java.io.Serializable interface, so the object's class must implement Serializable.
  - The Serializable interface is a marker interface. Since it has no methods, you don't need to add additional code in your class that implements Serializable. Implementing this interface enables the Java serialization mechanism to automate the process of storing objects and arrays.

06

Programming Practice

Practice 1：Palindrome

① A **palindrome** is a word, number, phrase, or other sequence of characters which reads the same backward as forward, such as madam or race car. Many interesting palindromes can be found in the web page http://www.palindromelist.net/ (after removing spaces and punctuation). Write a Java program to repeatedly read a string and to check whether it is a palindrome or not. The program stops when the input string is "000".

② The document longest_palindrome.txt is said to be the longest palindrome in the world. Write a Java program to do the following steps:

a. Read the file longest_palindrome.txt into a buffer of character sequence,
b. Ignore all characters that are not English letters and change all English characters to uppercase,
c. Output the first 500 characters, 80 characters in a line, of the modified text, and write the English string to the file result.txt,
d. Check if this English string is a palindrome.

```
>>>> Enter a String: abba
**** Yes, it is a palindrome.
========================================
>>>> Enter a String: acd
**** Not, it is not a palindrome.
========================================
>>>> Enter a String: caracre
**** Not, it is not a palindrome.
========================================
>>>> Enter a String: 000
```

```
>>>> Text length: 103385
========================================
>>>> File longest_palindrome.txt has been read.
>>>> The number of English characters in longest_palindrome.txt: 72061
>>>> The first 500 English characters:
AMANAPLANACARPUSAECRICKEYEKGNAVETTESSORCHABASILBSHATIZESOJAIAOULANAJUTATILD
IKOMS
AREMERABMANADOOPALINEBESSARGENAHCLAROBALOCARACALLAHAGAIDAXAPIXELALOEDAAMSRO
MRONA
LIALFROBEDIALBEDOPTASCBALIENAROMLEALTANANAOSALLBASAVROTOSDABIDANNABALKIONAA
RIMAS
SAMADUALALYESOLISNORADSMVENALONGARELLAEIRETULSAFARTSUGASEELARDISRISAVERATIH
WAAGU
YFAAANATADAGGAANAPELAMCMGEVAELGANDIADIBREGGUSCASREDANANANAMDISNEDRAHCAPTELE
DINAC
FIADUSTCASIACITNBWDCLVELAABUSTASIPPMAANAIDNBEASINNELAHARPABENEHACPSDACCAEBB
AANOV
NEBABMAGROGTANYAAILA
========================================
>>>> File result.txt has been written.
========================================
**** Yes, it is a palindrome. Final indices: i=36030, j=36030
```

Practice 2：File Copier

Write a program that lets users copy files.

The user needs to provide a source file and a target file as command-line arguments using the command

Copy source_filename target_filename

Note:

a. The program copies the source file to the target file and displays the number of bytes in the file.
b. The program should alert the user if the source file does not exist or if the target file already exists.

```
>>>> Copy source1.txt target1.txt
The FileCopier is copied 196 bytes successfully.

>>>> Copy source2.txt target2.txt
The source2.txt does not exist.
```