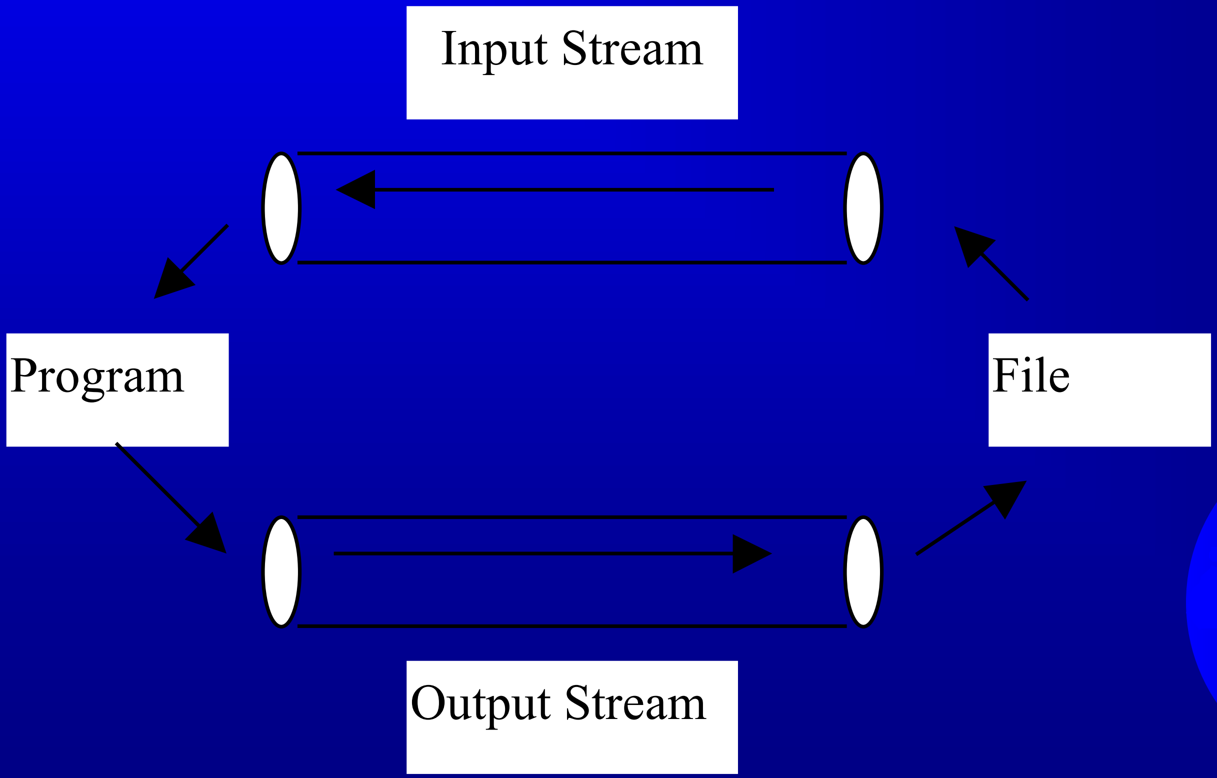
Chapter 15: Input and Output

- Stream Classes
- Processing External Files
- Data Streams
- Print Streams
- Buffered Streams
- Use `JFileChooser`
- Text Input and Output on the Console
- Random Access Files
- Parsing Text Files

Streams

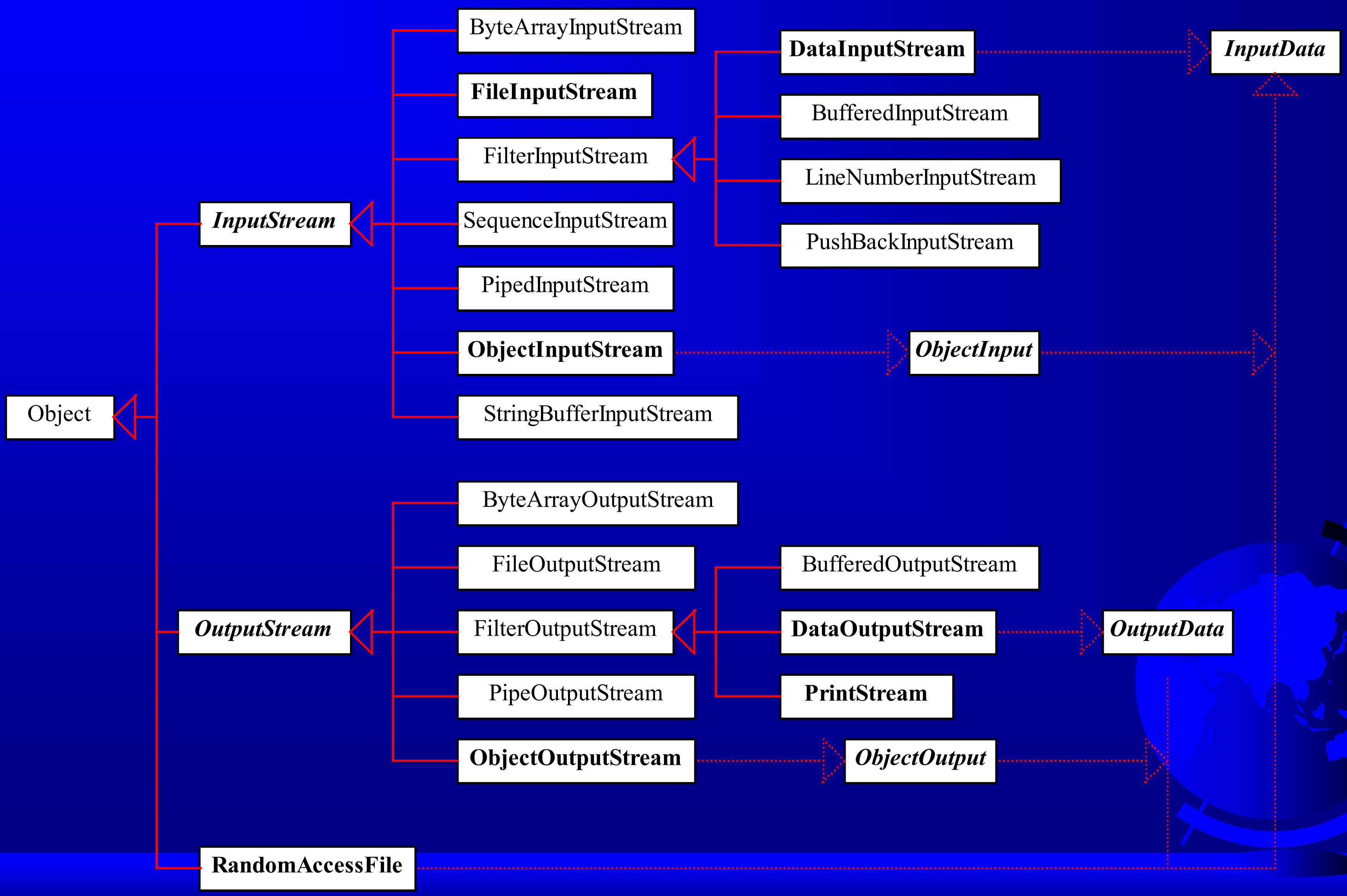
- A *stream* is an abstraction of the continuous one-way flow of data.



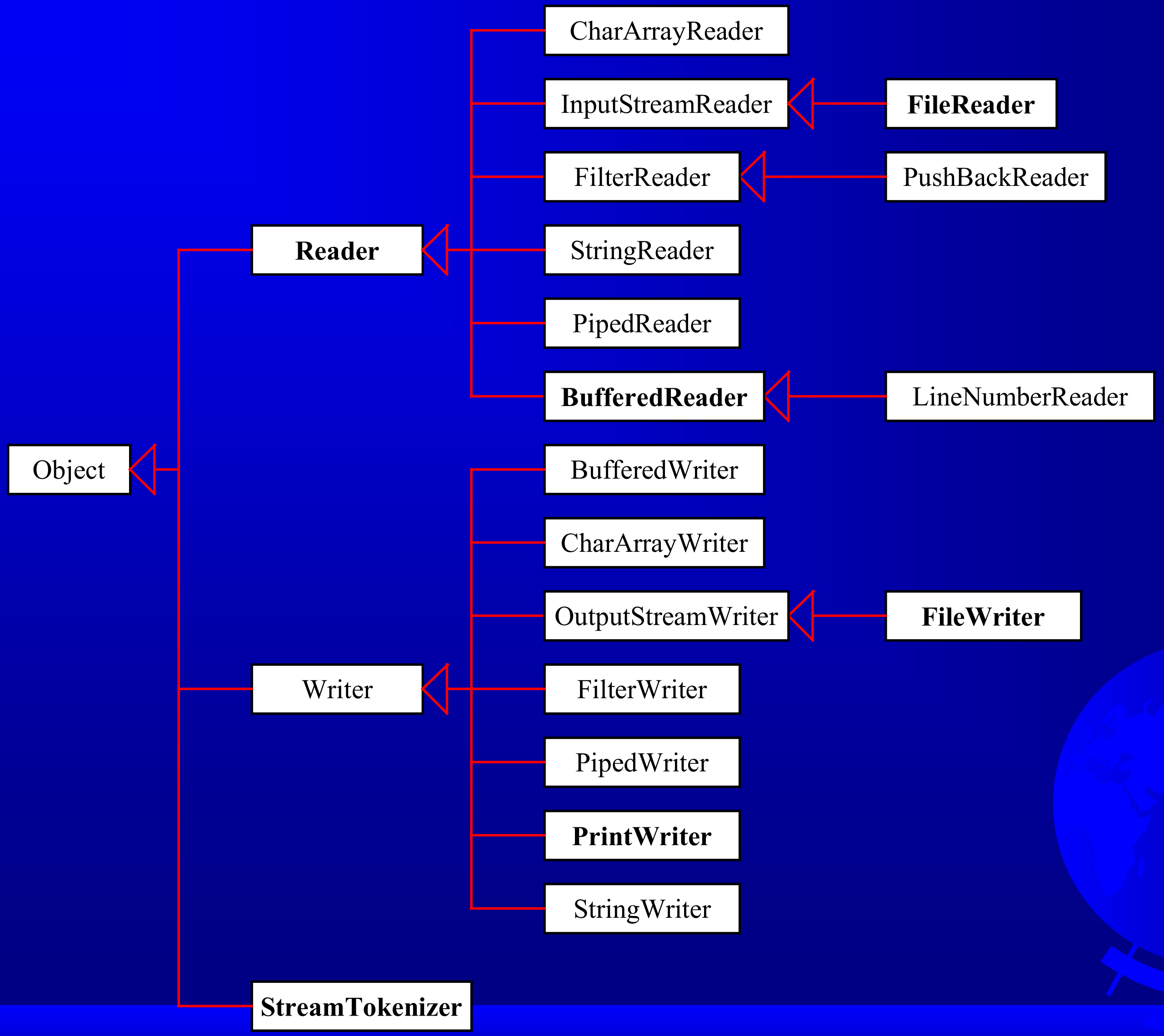
Stream Classes

- The stream classes can be categorized into two types: *byte streams* and *character streams*.
- The `InputStream/OutputStream` class is the root of all byte stream classes, and the `Reader/Writer` class is the root of all character stream classes. The subclasses of `InputStream/OutputStream` are analogous to the subclasses of `Reader/Writer`.

Byte Stream Classes



Character Stream Classes



InputStream

- abstract int read() throws IOException
- int read(byte b[]) throws IOException
- void close() throws IOException
- void available() throws IOException
- void skip() throws IOException

Reader

The `Reader` class is similar to the `InputStream` class. The methods in `Reader` are subject to character interpretation.

- `abstract int read() throws IOException`
- `int read(char b[]) throws IOException`
- `void close() throws IOException`
- `void skip() throws IOException`

OutputStream

- `abstract void write(int b) throws IOException`
- `void write(byte[] b) throws IOException`
- `void close() throws IOException`
- `void flush() throws IOException`

Writer

- abstract void write(int b) throws IOException
- void write(char[] b) throws IOException
- void close() throws IOException
- void flush() throws IOException

Processing External Files

You must use file streams to read from or write to a disk file. You can use `FileInputStream` or `FileOutputStream` for byte streams, and you can use `FileReader` or `FileWriter` for character streams.

File I/O Stream Constructors

Constructing instances of `FileInputStream`, `FileOutputStream`, `FileReader`, and `FileWriter` from file names:

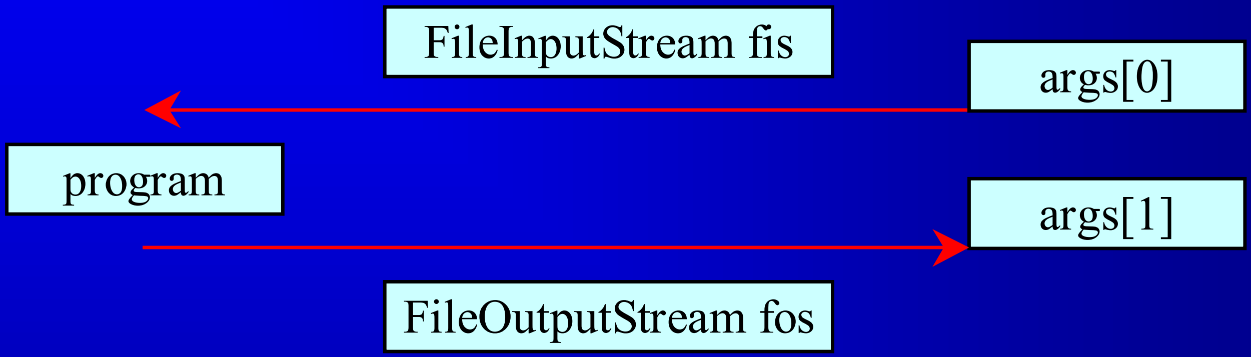
```
FileInputStream infile = new FileInputStream("in.dat");

FileOutputStream outfile = new FileOutputStream("out.dat");

FileReader infile = new FileReader("in.dat");

FileWriter outfile = new FileWriter("out.dat");
```

Example 15.1
Processing External Files



CopyFileUsingByteStream

Run

Click the Run button to access the DOS prompt; then type `java CopyFileUsingByteStream ButtonDemo.java t.java` and press Enter. (If working from the CD, add a path to the hard disk or floppy disk drive for t.java.)

Data Streams

The data streams (`DataInputStream` and `DataOutputStream`) read and write Java primitive types in a machine-independent fashion, which enables you to write a data file in one machine and read it on another machine that has a different operating system or file structure.

DataInputStream Methods

- int readByte() throws IOException
- int readShort() throws IOException
- int readInt() throws IOException
- int readLong() throws IOException
- float readFloat() throws IOException
- double readDouble() throws IOException
- char readChar() throws IOException
- boolean readBoolean() throws IOException
- String readUTF() throws IOException

DataOutputStream Methods

- `void writeByte(byte b) throws IOException`
- `void writeShort(short s) throws IOException`
- `void writeInt(int i) throws IOException`
- `void writeLong(long l) throws IOException`
- `void writeFloat(float f) throws IOException`
- `void writeDouble(double d) throws IOException`
- `void writeChar(char c) throws IOException`
- `void writeBoolean(boolean b) throws IOException`
- `void writeBytes(String l) throws IOException`
- `void writeChars(String l) throws IOException`
- `void writeUTF(String l) throws IOException`

Data I/O Stream Constructors

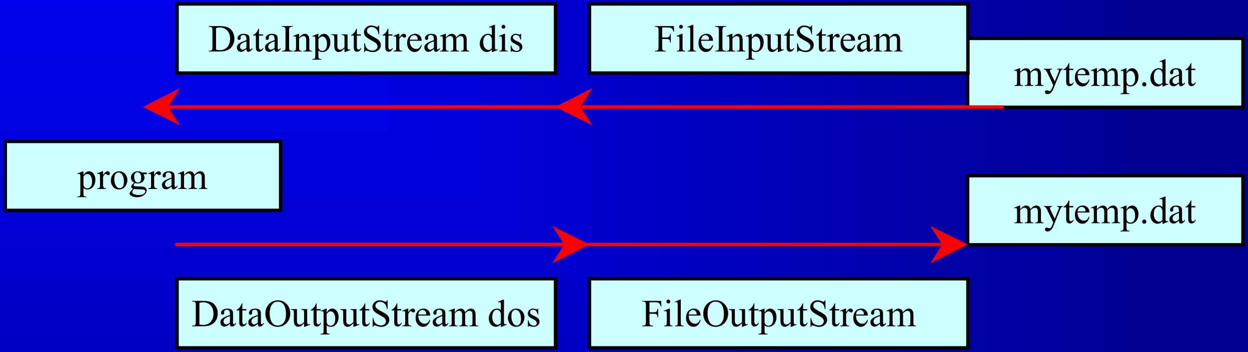
☞ `DataInputStream infile = new DataInputStream(new FileInputStream("in.dat"));`

Creates an input file for in.dat.

☞ `DataOutputStream outfile = new DataOutputStream(new FileOutputStream("out.dat"));`

Creates an output file for out.dat.

Example 15.2
Using Data Streams



TestDataStreams

Run

Click the Run button to access the DOS prompt; then type `java TestDataStreams` and press Enter. (Note: You cannot run this from the CD; the program writes to disk.)

Print Streams

The data output stream outputs a binary representation of data, so you cannot view its contents as text. In Java, you can use print streams to output data into files. These files can be viewed as text.

The `PrintStream` and `PrintWriter` classes provide this functionality.

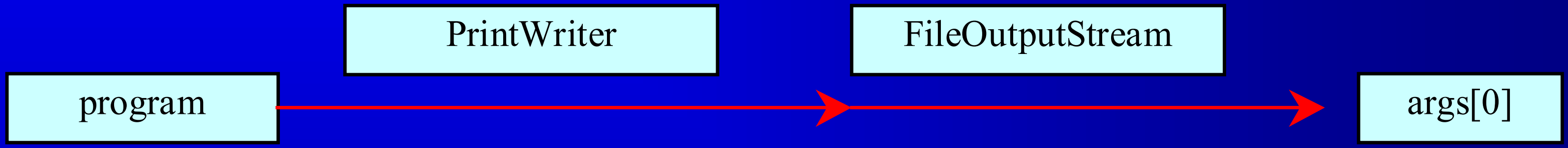
`PrintWriter` Constructors

- `PrintWriter(Writer out)`
- `PrintWriter(Writer out, boolean autoFlush)`
- `PrintWriter(OutputStream out)`
- `PrintWriter(OutputStream out, boolean autoFlush)`

PrintWriter Methods

- ☞ `void print(Object o)`
- ☞ `void print(String s)`
- ☞ `void println(String s)`
- ☞ `void print(char c)`
- ☞ `void print(char[] cArray)`
- ☞ `void print(int i)`
- ☞ `void print(long l)`
- ☞ `void print(float f)`
- ☞ `void print(double d)`
- ☞ `void print(boolean b)`

Example 15.3
Using Print Streams



TestPrintWriters

Run

Click the Run button to access the DOS prompt; then type `java TestPrintWriters t.dat` and press Enter. (Note: You cannot run this from the CD; the program writes to disk.)

Buffered Streams

Java introduces buffered streams that speed up input and output by reducing the number of reads and writes. In the case of input, a bunch of data is read all at once instead of one byte at a time. In the case of output, data are first cached into a buffer, then written all together to the file.

Using buffered streams is highly recommended.

Buffered Stream Constructors

- ☞ `BufferedInputStream (InputStream in)`
- ☞ `BufferedInputStream (InputStream in, int bufferSize)`
- ☞ `BufferedOutputStream (OutputStream in)`
- ☞ `BufferedOutputStream (OutputStream in, int bufferSize)`
- ☞ `BufferedReader(Reader in)`
- ☞ `BufferedReader(Reader in, int bufferSize)`
- ☞ `BufferedWriter(Writer out)`
- ☞ `BufferedWriter(Writer out, int bufferSize)`

Example 15.4
Displaying a File in a Text Area

- Objective: View a file in a text area. The user enters a filename in a text field and clicks the View button; the file is then displayed in a text area.

ViewFile

Run

Example 15.5
Using File Dialogs

- Objective: Create a simple notepad using JFileChooser to open and save files. The notepad enables the user to open an existing file, edit the file, and save the note into the current file or to a specified file. You can display and edit the file in a text area.

FileDialogDemo   Run

Note: You cannot run this from the CD; the program writes to disk.

Text Input and Output on the Consoles

There are two types of interactive I/O. One involves simple input from the keyboard and simple output in a pure text form. The other involves input from various input devices and output to a graphical environment on frames and applets. The former is referred to as *text interactive I/O*, and the latter is known as *graphical interactive I/O*.

Console Output/Input

To perform console output, you can use any of the methods for `PrintStream` in `System.out`. However, keyboard input is not directly supported in Java. In order to get input from the keyboard, you first use the following statements to read a string from the keyboard.

MyInput

Object Streams

Object streams enable you to perform input and output at the object level.

To enable an object to be read or write, the object's defining class has to implement the java.io.Serializable interface or the java.io.Serializable interface or the java.io.Externalizable interface.

The `Serializable` Interface

The Serializable interface is a marker interface. It has no methods, so you don't need to add additional code in your class that implements Serializable.

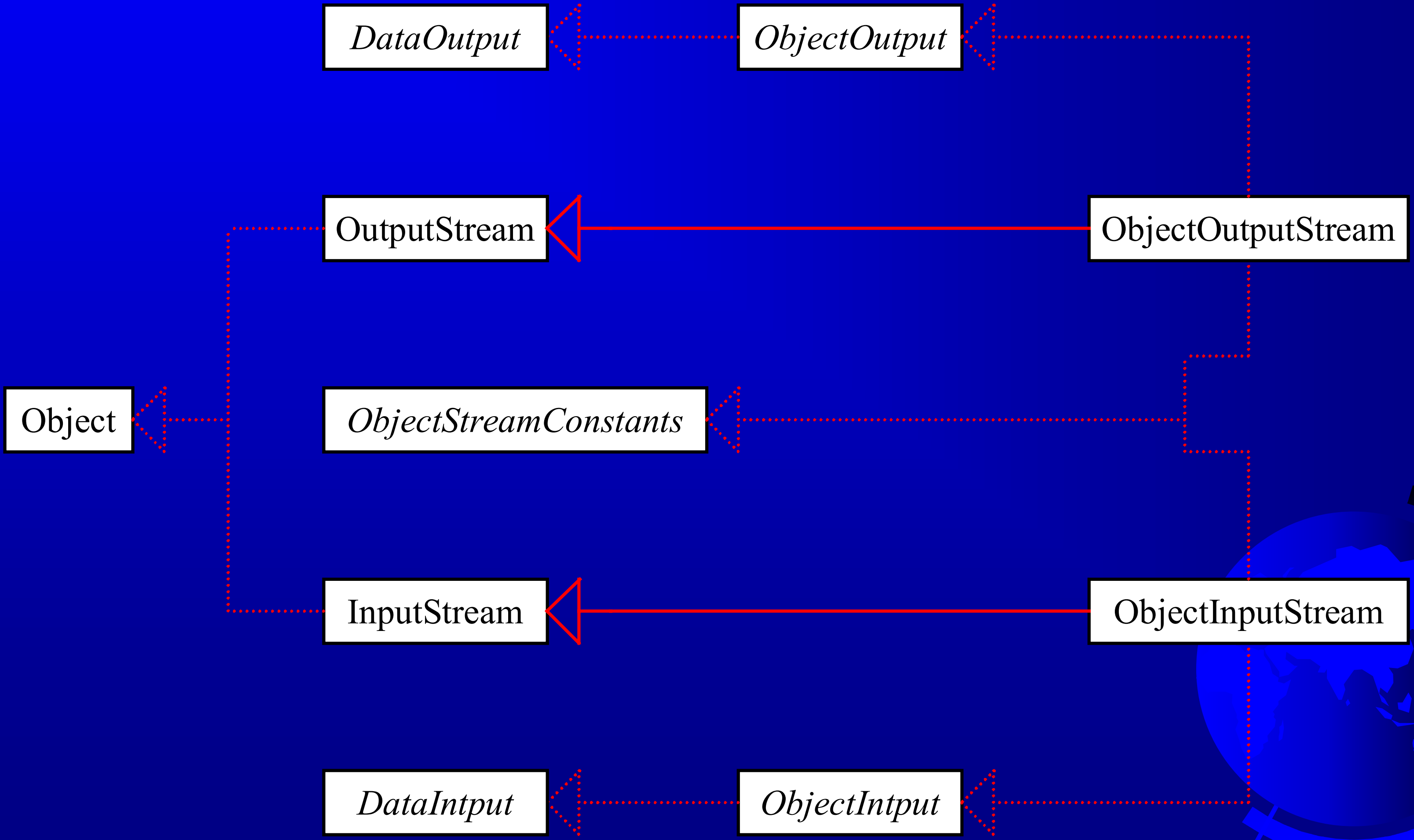
Implementing this interface enables the Java serialization mechanism to automate the process of storing the objects and arrays.

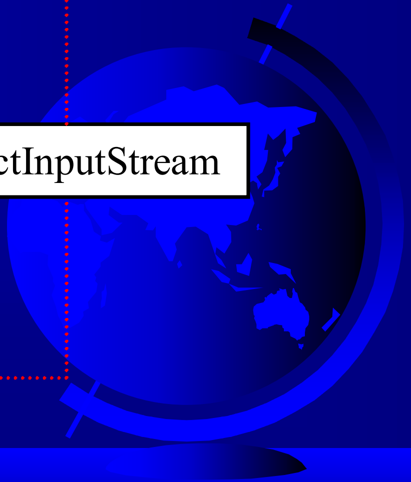
The Object Streams

You need to use the `ObjectOutputStream` class for storing objects and the `ObjectInputStream` class for restoring objects.

These two classes are built upon several other classes.

The `ObjectOutput` and `ObjectInput` Streams





Example 15.6
Testing Object Streams

- Objective: Stores objects of `MessagePanel` and `Date`, and Restores these objects.

ObjectStreamDemo

Run

Note: You cannot run this from the CD; the program writes to disk.

Random Access Files

- Java provides the `RandomAccessFile` class to allow a file to be read and updated at the same time.

- The `RandomAccessFile` class extends `Object` and implements `DataInput` and `DataOutput` interfaces.

`RandomAccessFile` Methods

Many methods in `RandomAccessFile` are the same as those in `DataInputStream` and `DataOutputStream`. For example, `readInt()`, `readLong()`, `writeDouble()`, `readLine()`, `writeInt()`, and `writeLong()` can be used in data input stream or data output stream as well as in `RandomAccessFile` streams.

`RandomAccessFile` Methods, cont.

- `void seek(long pos) throws IOException;`

  Sets the offset from the beginning of the `RandomAccessFile` stream to where the next read or write occurs.

- `long getFilePointer() IOException;`

  Returns the current offset, in bytes, from the beginning of the file to where the next read or write occurs.

`RandomAccessFile` Methods, cont.

- `long length()IOException`

  Returns the length of the file.

- `final void writeChar(int v) throws IOException`

  Writes a character to the file as a two-byte Unicode, with the high byte written first.

- `final void writeChars(String s) throws IOException`

  Writes a string to the file as a sequence of characters.

`RandomAccessFile` Constructor

```
RandomAccessFile raf =
  new RandomAccessFile("test.dat",
  "rw"); //allows read and write


RandomAccessFile raf =
  new RandomAccessFile("test.dat",
  "r"); //read only
```

Example 15. 7 Using Random Access Files

- Objective: Create a program that registers students and displays student information.

TestRandomAccessFile

Run

Note: You cannot run this from the CD; the program writes to disk.

Parsing Text Files (Optional)

The `StreamTokenizer` class lets you take an input stream and parse it into words, which are known as *tokens*. The tokens are read one at a time. The following is the `StreamTokenizer` constructor:

```
StreamTokenizer st =
  StreamTokenizer(Reader is)
```

StreamTokenizer Constants

- TT_WORD

  The token is a word.

- TT_NUMBER

  The token is a number.

- TT_EOL

  The end of the line has been read.

- TT_EOF

  The end of the file has been read.

`StreamTokenizer` Variables

- `int ttype`

  Contains the current token type, which matches one of the constants listed on the preceding slide.

- `double nval`

  Contains the value of the current token if that token is a number.

- `String sval`

  Contains a string that gives the characters of the current token if that token is a word.

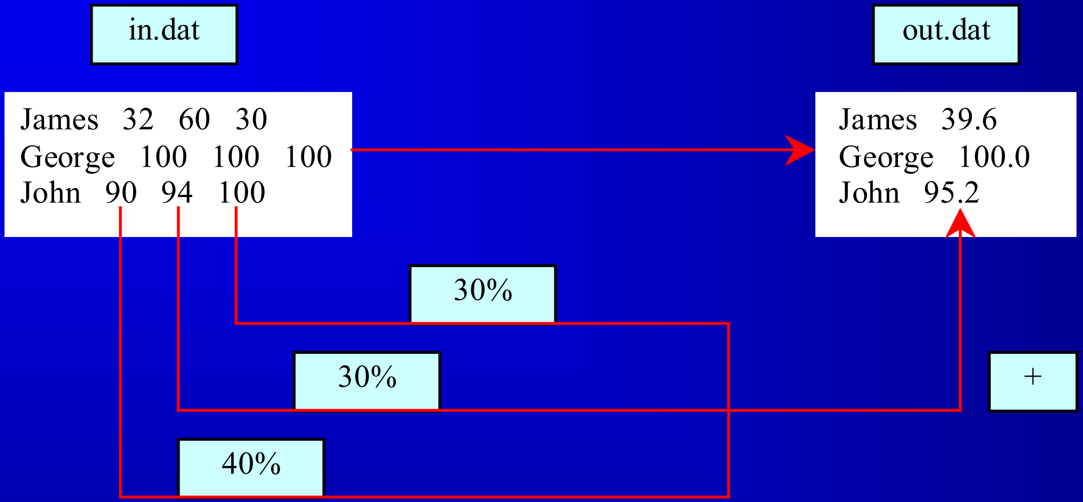
`StreamTokenizer` Methods

```
public int nextToken() throws
   IOException
```

Parses the next token from the input stream of this `StreamTokenizer`.

The type of the next token is returned in the `ttype` field. If `ttype == TT_WORD`, the token is stored in `sval`; if `ttype == TT_NUMBER`, the token is stored in `nval`.

Example 15.8 Using `StreamTokenizer`



ParsingTextFile

Run

Click the Run button to access the DOS prompt; then type `java ParsingTextFile` and press Enter. (Note: You cannot run this from the CD; the program writes to disk.)