Object-Oriented Programming

Streams and Files

Outline

- Concepts of Data Streams
- Streams and Files
- Manipulate Text Files
- Manipulate Binary Files
- RandomAccessFile Class

Data Streams

- Data are stored as a sequence of bytes:
  - But, we can consider data as having some higher-level structure such as being a sequence of characters or objects
- Streams: a sequence of data that is read from a source or written to a destination
  - Source: file, memory, keyboard,…
  - Destination: file, memory, screen,…
- Java programs send and receive data via objects of some data stream types

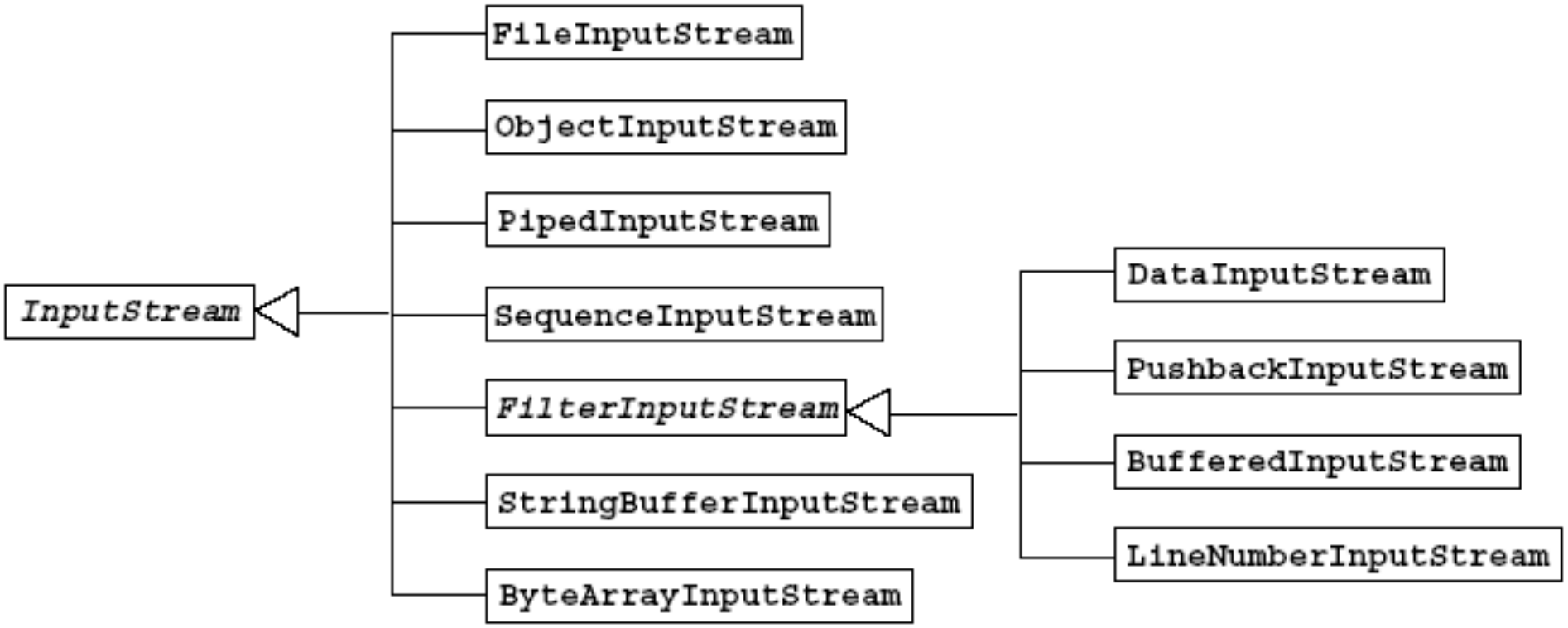
Data Streams

- Streams: may be connected to a file on floppy, a file on a hard disk, a network connection or may even just be in memory
- We abstract away what the stream is connected to, and just focus on performing I/O operations on the stream

Types of Streams

- Byte streams: manipulate data in bytes. Two abstract classes are provided
  - InputStream
  - OutputStream
- Character streams: manipulate data as Unicode text streams. Two abstract classes are provided
  - Reader
  - Writer

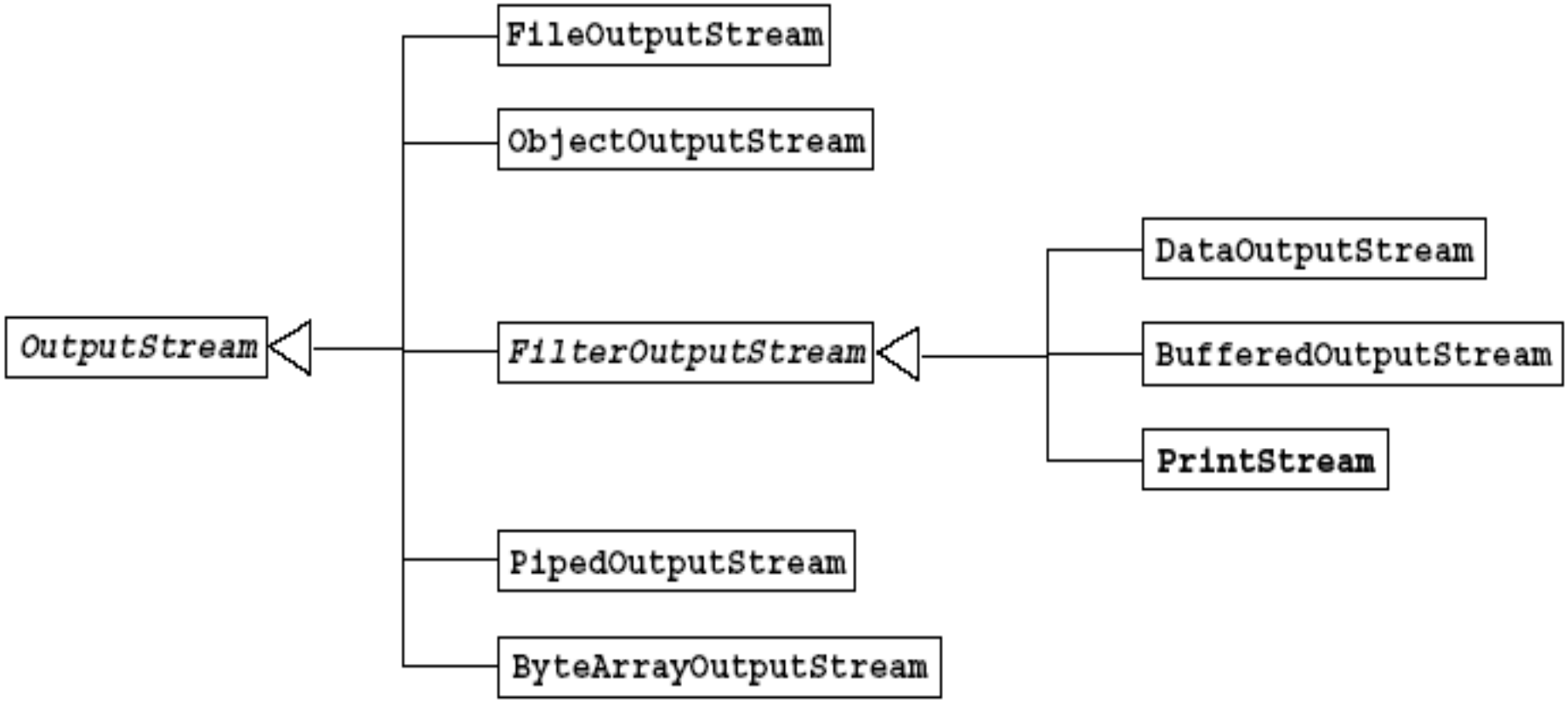
InputStream Hierarchy



Methods of InputStream

Method	Description
int read()	reads next byte of data from input stream
int read(byte[] b)	reads “b.length” bytes from input stream to array “b”
int read(byte[] b, int offset, int length)	reads “length” bytes from input stream to array “b”, starting from the “offset” address
void close()	closes input stream

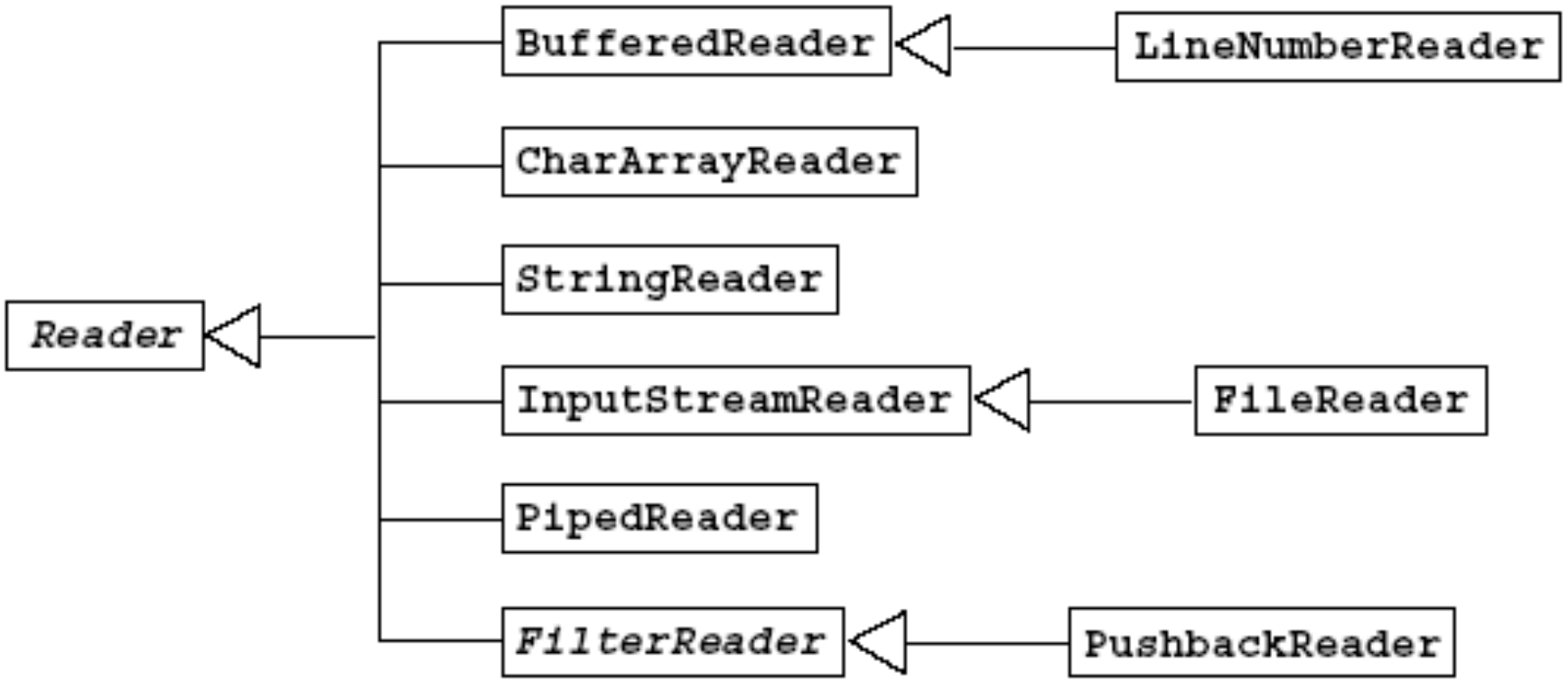
OutputStream Hierarchy



Methods of OutputStream

Method	Description
int write(int c)	writes the single byte “c” to output stream
int write(byte[] b)	writes “b.length” bytes from array “b” to output stream
int write(byte[] b, int offset, int length)	writes “length” bytes of array “b”, starting from the “offset” address to output stream
void close()	closes output stream
Void flush()	flushes the data stream from buffer to output stream

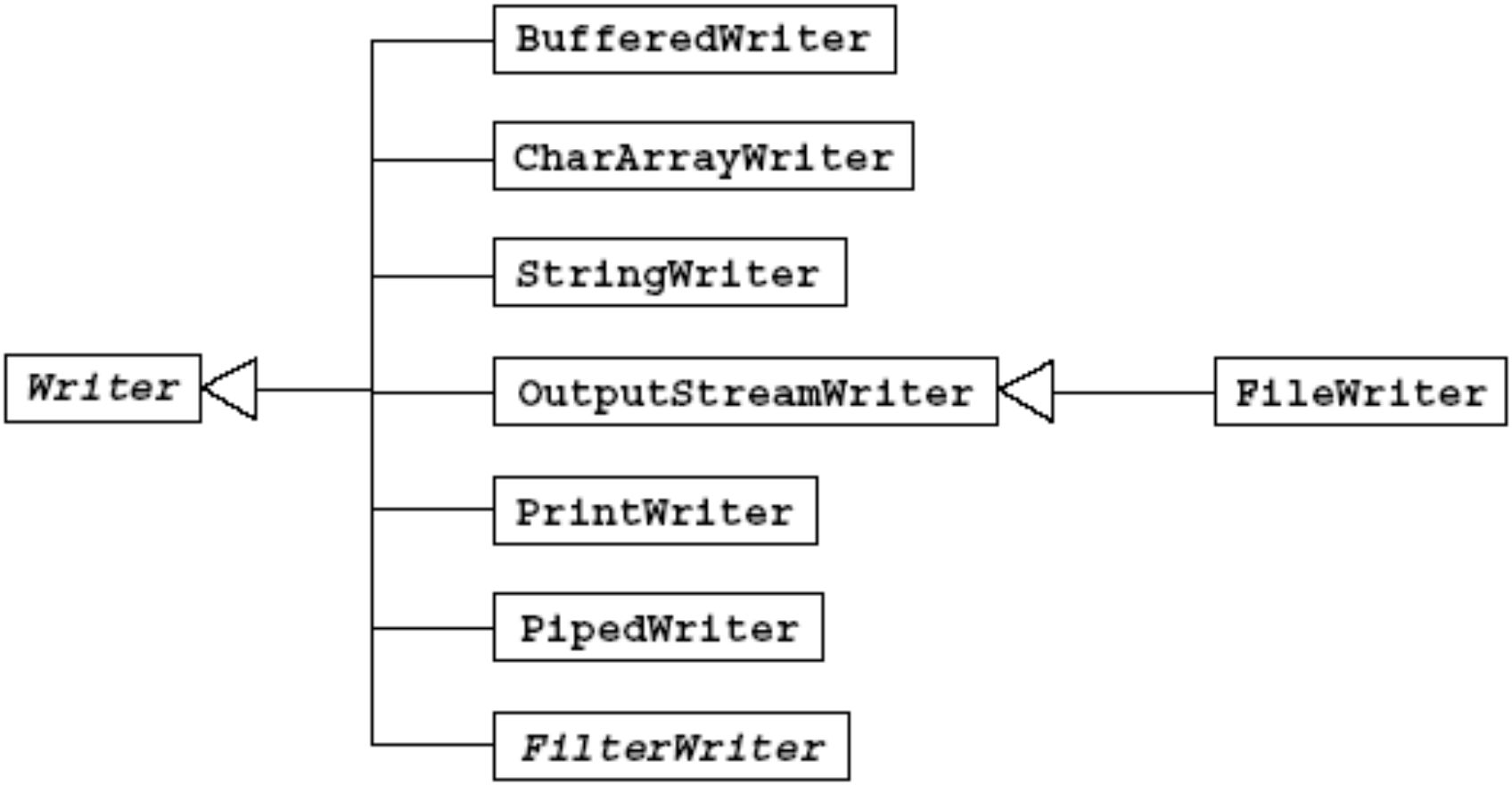
Reader Hierarchy



Methods of Reader

Method	Description
int read()	reads next character from input stream
int read(char[] b)	reads “b.length” characters from input stream to array “b”
int read(char[] b, int offset, int length)	reads “length” characters from input stream to array “b”, starting from the “offset” address
void close()	Closes input stream

Writer Hierarchy



Methods of Writer

Method	Description
int write(int c)	writes the single character “c” to output stream
int write(char[] b)	writes “b.length” characters from array “b” to output stream
int write(char[] b, int offset, int length)	writes “length” characters from array “b”, starting from the “offset” to output stream
void close()	closes output stream
void flush()	flushes the data stream from buffer to output stream

Important Types of Streams

- InputStream/OutputStream
  - Base class streams with few features
- FileInputStream/FileOutputStream
  - Specifically for connecting to files
- BufferedInputStream/BufferedOutputStream
  - Improve I/O performance by adding buffers
- BufferedReader/BufferedWriter
  - Convert bytes to Unicode Char and String data

Input/output stream object

- To read or write data, we need a stream object
- The I/O stream object needs to be attached to a data source or a destination

```
BufferedReader in =
    new BufferedReader(new FileReader(fname));
```

Use of buffered streams

- Buffering is a technique to improve I/O performance
  - Read and write data in blocks
  - Reduce number of accesses to I/O devices
- The program writes data to the buffer instead of output devices
  - When the buffer is full, data in buffer is pushed to the device in blocks
  - We can force data to be pushed by calling flush() method
- The program reads data from buffer instead of input devices
  - When the buffer is empty, data is retrieved from the input device in blocks

Standard I/O streams

- In java.lang package
- **System.out** and **System.err** are **PrintStream** objects
  - Can be used directly

```
System.out.println("Hello, world!");
System.err.println("Invalid day of month!");
```

- **System.in** is an InputStream object
  - Used with **InputStreamReader** (character stream) and **BufferedReader** (stream with buffer)

```
BufferedReader br = new BufferedReader(
        new InputStreamReader(System.in))
```

The File class

- In java.io package
- Provides basic operations on files and directories
  - Create files, open files, query directory information
- Files are not streams

Create a File object

- File myFile;
- myFile = new File(“data.txt”);
- myFile = new File(“myDocs”, “data.txt”);

- Directories are treated the same as files:
  - File myDir = new File(“myDocs”);
  - File myFile = new File(myDir, “data.txt”);

File's methods

- File/directory name
  - `String getName()`
  - `String getPath()`
  - `String getAbsolutePath()`
  - `String getParent()`
  - `boolean renameTo(File newName)`
- File/directory status
  - `boolean exists()`
  - `boolean canWrite()`
  - `boolean canRead()`
  - `boolean isFile()`
  - `boolean isDirectory()`

File's methods

- status
  - `long lastModified()`
  - `long length()`
  - `boolean delete()`
- directory
  - `boolean mkdir()`
  - `String[] list()`

Manipulate text files

- Read from files
  - **`FileReader`**: read characters from text files
  - **`BufferedReader`**: buffered, read in lines

- Write to files
  - **`FileWriter`**: write characters to text files
  - **`PrintWriter`**: write in lines

Read from a text file

```
public void readLines(String fname) {
   try {
      BufferedReader in =
           new BufferedReader(new FileReader(fname));

      String line;
      while ((line = in.readLine()) != null) {
         System.out.println(line);
      }

      in.close();
   }
   catch (IOException e) {
      e.printStackTrace();
   }
}
```

Write to a text file

```
public void writeLines(String fname) {
   try {
        PrintWriter out = new PrintWriter(new FileWriter(fname));

        out.write("This is the object-oriented programming course");

        out.close();
   }
   catch (IOException e) {
      e.printStackTrace();
   }
}
```

Manipulate binary files

- Read
  - **`FileInputStream`**: read data from files
  - **`DataInputStream`**: read primitive data
  - **`ObjectInputStream`**: read objects

- Write
  - **`FileOutputStream`**: write data to files
  - **`DataOutputStream`**: write primitive data
  - **`ObjectOutputStream`**: write objects

DataInputStream/DataOutputStream

- **`DataInputStream`**: read primitive data
  - readBoolean, readByte, readChar, readShort, readInt, readLong, readFloat, readDouble
- **`DataOutputStream`**: write primitive data
  - writeBoolean, writeByte, writeChar, writeShort, writeInt, writeLong, writeFloat, writeDouble

Write primitive data

```
import java.io.*;

public class TestDataOutputStream {
   public static void main(String args[]) {
      int a[] = {65, 75, 86, 67, 98};

      try {
          // file name is entered as args[0]
          FileOutputStream fout = new FileOutputStream(args[0]);
          DataOutputStream dout = new DataOutputStream(fout);

          for (int i=0; i<a.length; i++)
             dout.writeInt(a[i]);
          dout.close();
      }
      catch (IOException e) {
          e.printStackTrace();
      }
   }
}
```

Read primitive data

```
import java.io.*;

public class TestDataInputStream {
   public static void main(String args[]) {
      try {
          FileInputStream fin = new FileInputStream(args[0]);
          DataInputStream din = new DataInputStream(fin);

          while (true) {
             System.out.println(din.readInt());
          }
      }
      catch (EOFException e) {
      }
      catch (IOException e) {
          e.printStackTrace();
      }
   }
}
```

File of objects

- Objects can be stored
- Data classes must implement interface **Serializable**

```
import java.io.Serializable;

class Record implements Serializable {
   private String name;
   private float score;

   public Record(String s, float sc) {
      name = s;
      score = sc;
   }

   public String toString() {
      return "Name: " + name + ", score: " + score;
   }
}
```

```
import java.io.*;

public class TestObjectOutputStream {
   public static void main(String args[]) {
      Record r[] = { new Record("john", 5.0F),
                     new Record("mary", 5.5F),
                     new Record("bob", 4.5F) };

      try {
          FileOutputStream fout = new FileOutputStream(args[0]);
          ObjectOutputStream out = new ObjectOutputStream(fout);

          for (int i=0; i<r.length; i++)
             out.writeObject(r[i]);

          out.close();
      }
      catch (IOException e) {
         e.printStackTrace();
      }
   }
}
```

```
public class TestObjectInputStream {
   public static void main(String args[]) {
      Record r;
      try {
          FileInputStream fin = new FileInputStream(args[0]);
          ObjectInputStream in = new ObjectInputStream(fin);

          while (true) {
             r = (Record) in.readObject();
             System.out.println(r);
          }
      }
      catch (EOFException e) {
          System.out.println("No more records");
      }
      catch (ClassNotFoundException e) {
          System.out.println("Unable to create object");
      }
      catch (IOException e) {
          e.printStackTrace();
      }
   }
}
```

The class RandomAccessFile

- Implement DataInput interface to read data randomly
- Implement DataOutput interface to write randomly

```
import java.io.*;

public class WriteRandomFile {
   public static void main(String args[]) {
      int a[] = { 2, 3, 5, 7, 11, 13 };

      try {
          File fout = new File(args[0]);
          RandomAccessFile out;
           out = new RandomAccessFile(fout, "rw");

          for (int i=0; i<a.length; i++)
             out.writeInt(a[i]);
          out.close();
      }
      catch (IOException e) {
          e.printStackTrace();
      }
   }
}
```

```
import java.io.*;

public class ReadRandomFile {
   public static void main(String args[]) {

      try {
          File fin = new File(args[0]);
          RandomAccessFile in = new RandomAccessFile(fin, "r");

          int recordNum = (int) (in.length() / 4);
          for (int i=recordNum-1; i>=0; i--) {
             in.seek(i*4);
             System.out.println(in.readInt());
          }
      }
      catch (IOException e) {
          e.printStackTrace();
      }
   }
}
```