File Input and Output

Types of I/O

- Text I/O
  - Read from and write to a text file
  - Text files can be read with Notepad
  - Text files can be transmitted through networks and firewalls
  - Text files are larger and easier to read (less secure)
- Binary I/O
  - Read from and write to a binary file (note: all files are binary!)
  - Binary files must be translated to be read
  - Binary files have trouble transmitting through networks, firewalls
  - Text files are smaller, but not more secure!

Text I/O

- The File class
  - Represents the file as an object (think: “file descriptor”)
  - Can be a file *or a directory*
  - Does not do any reading or writing
- Scanner
  - Reads from text files, just like from console input
  - Always check to see if the file has another line!
  - Close when complete (use finally!)
- PrintWriter
  - Writes to text files, same construction as System.out.printXXX
  - Always replaces content, never appends
  - Close when complete (use finally!)

Text I/O

```
File aFile = new File(“C:\\temp\\sample.txt”);

PrintWriter output = new PrintWriter(aFile);

double dMoney = 19.9;
output.println(“Printing a line to a file...”);
output.print(“Printing something to a file...”);
output.printf(“Printing formatted text %1.2f”, dMoney);
output.close();

Scanner input = new Scanner(aFile);
while(input.hasNextLine()) {
   System.out.println(input.nextLine());
   System.out.println(input.next());
   System.out.println(input.nextDouble());
}
input.close();
```

Text I/O: Good methods to know

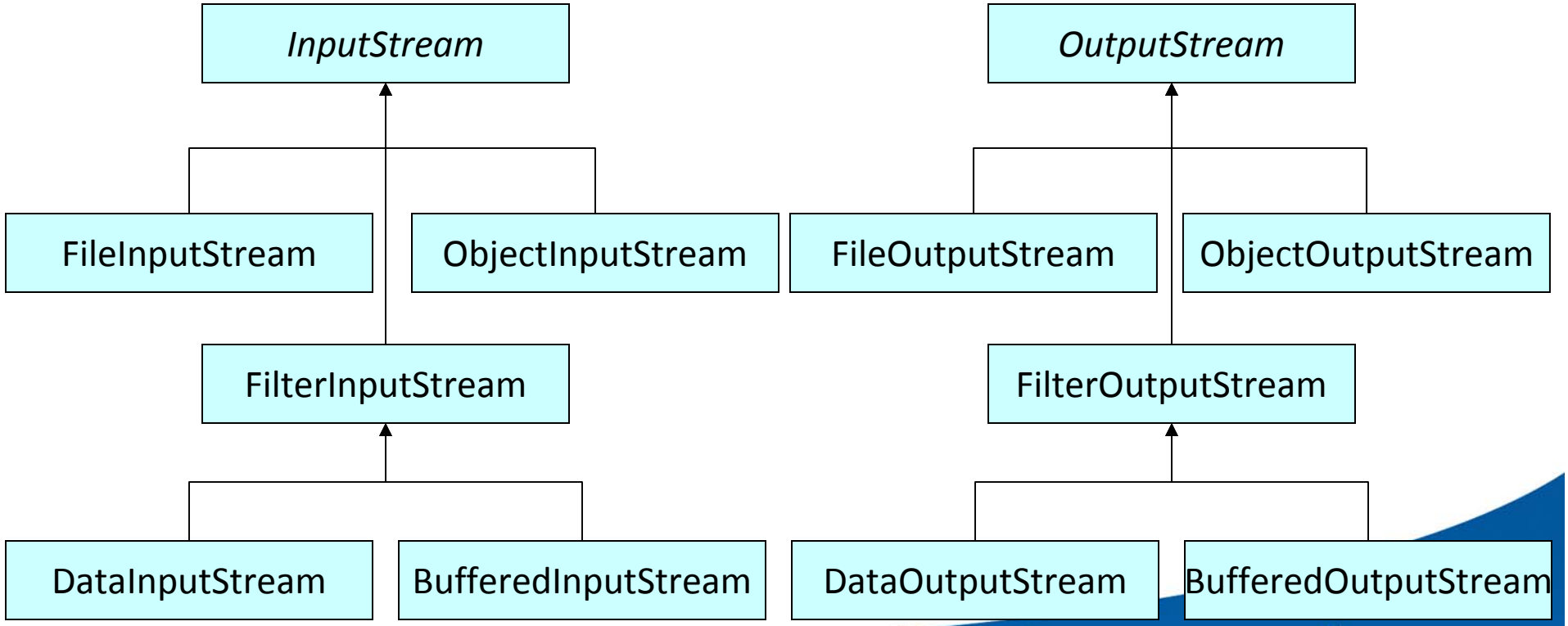
File
+exists() : boolean +isFile() : boolean +isDirectory() : boolean +createNewFile() : boolean +mkdir() : boolean +getName() : String +getPath() : String +getParent() : String +isHidden() : boolean +lastModified() : long +length() : long +listFiles() : File[] +delete() : boolean

PrintWriter
+exists() : boolean +println(s: String): void +print(s: String): void +printf(s: String): void

Scanner
+exists() : boolean +hasNext():boolean +hasNextLine():boolean +next():String +nextLine():String +nextInt():int +nextDouble():double +useDelimiter(pattern:String)

Binary I/O

- Many binary I/O classes, all subclassed from abstract streams:



Concepts in binary I/O

- Encoding
  - Encoding and decoding ‘translate’ files from binary to something we can read
- Reading and writing with binary I/O
  - Most binary readers/writers work with bytes.
  - You may need to transform your data to bytes: use `byteCode()`
- The *stream*
  - A unidirectional flow, like turning a tap on: either reading or writing
- Buffering
  - Capturing flow, then read/write it in ‘chunks’
  - Think of filling a pitcher with the tap on

File Stream

- If the file doesn't exist, file stream classes create a new one
- FileOutputStream always overwrites unless append is true

  +FileOutputStream(file : File, append : boolean)

```
FileOutputStream output = new FileOutputStream(aFileName);
output.write(aValue);  // either int or byte
output.close();
FileInputStream input = new FileInputStream(aFileName);
int value = input.read();
input.close();
```

- Types of exceptions
  - FileNotFoundException: if the file cannot be located in the path
  - IOException: generic I/O exception

Data Streams

- Offer more flexibility working with data types
  - read and write bytes, char, float, double, int, long, short, and line
  - DataInputStream, DataOutputStream

```
DataOutputStream output = new DataOutputStream(outputStream);
output.writeDouble(20.2);
output.writeInt(5);
output.close();

DataInputStream input = new DataInputStream(inputStream);
int dValue = input.readDouble();
int iValue = input.readInt();
input.close();
```

  - Use a catch block to catch EOFException

Buffered Streams

- Reduces number of disk reads/writes by adding data to buffer
- Default buffer size is 512 bytes
  - Use the BufferedXXXStream constructor to change size
  - Invalid sizes throw an exception
- No different methods from InputStream/OutputStream
  - Only difference is the use of the JVM buffer

Serialization

- Instead of individual variables, save whole objects
- Objects can only be written if they are serializable
- Implement the *Serializable* interface on any class
  - Serializable has no methods!
- What is not serializable?
  - Any class that does not implement Serializable
  - Any variable that is static
  - Any variable declared as *transient*

```
private transient String password;
```

Object Streams

```
public class Medication implements Serializable {
```

- ObjectOutputStream

```
ObjectOutputStream output =
   new ObjectOutputStream(new FileOutputStream(aFileObject));
output.writeObject(aMedication);
```

- ObjectInputStream

```
ObjectInputStream input =
   new ObjectInputStream(new FileInputStream(aFileObject));
Medication aDrug = (Medication) input.readObject();
```

JFileChooser

- Swing dialog
- Helps create a File object from the selection

```
JFileChooser fileChooser = new JFileChooser();

int option = fileChooser.showOpenDialog(null);
if(option == JFileChooser.APPROVE_OPTION) {
   File aFile = fileChooser.getSelectedFile();

// JFileChooser.CANCEL_OPTION is the other option
```

File access

- Sequential
  - Read the file sequentially, from beginning to end
  - Write the file sequentially, from beginning to end
  - Append data to the end
- But what if....?
  - You need to edit data in the file?
  - You need to insert data into the file?
  - You need to delete data from the file?

Random file access

- Random access files (`RandomAccessFile`)
  - Allow read/write streams (bidirectional)
  - Insert, read, update, delete information (CRUD)
  - Access data anywhere in the file
  - Access data in any order
- RandomAccessFile

```
RandomAccessFile raf = new RandomAccessFile(aFile, “rw”);
```

  - Written data is ‘typed’
  - Each data type is written with bytes based on type’s length (4-8 bytes)
  - Position the file pointer first before reading the respective byte

Random file access programming

- Reading is done with an appropriate readXXX method
  - readBoolean(), readByte(), readInt(), readDouble(), readLine()
- Writing is done with an appropriate writeXXX method
  - writeBoolean(), writeByte(), writeInt(), writeDouble(), writeChars()
- Position the file pointer with the **seek()** method
  - You must seek the correct byte by moving the file pointer 4-8 bytes!
- Obtain the length with the length() method
  - Change it with setLength()
  - setLength(0) erases the file!
- Hints:
  - Use primary keys
  - Use objects of a fixed length

Other I/O Concepts

- Delimiters
  - Symbols which separate fields and records (field delimiters, record delimiters)