File I/O

CSE 142, Summer 2002

Computer Programming 1

http://www.cs.washington.edu/education/courses/142/02su/

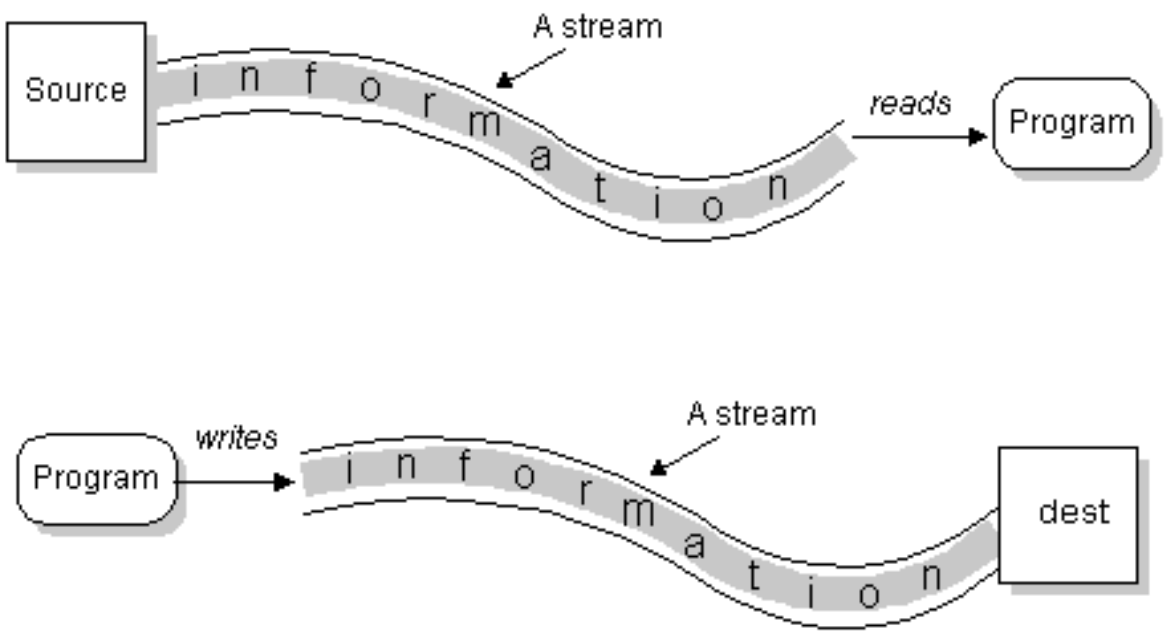
Readings and References

- Reading

- Other References
  - » Section "I/O" of the Java tutorial
  - » http://java.sun.com/docs/books/tutorial/essential/io/index.html

"Streams" are the basic I/O objects

keyboard,
disk file,
network,
etc



display,
disk file,
network,
etc

from Sun tutorial on I/O

The stream model

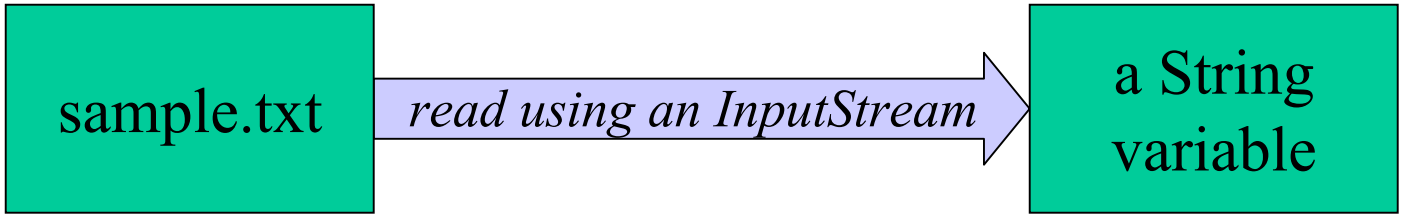
- Recall that the stream model views all data as coming from a source and going to a sink

Sources and Sinks - Files

- When reading from a file
  - » the file is the source
  - » a data structure in your application is the sink
- When writing to a file
  - » a data structure in your application is the source
  - » the file is the sink

The stream model applied to files

- The source can be a file on disk
  - » in this case, the sink is some variable in your program



FileInputStream and FileOutputStream

- The file streams read or write from a file on the native file system
  - » FileInputStream

    retrieve bytes from a file and provide them to the program
  - » FileOutputStream

    send bytes to a file from your program
- If used by themselves, FileInputStream and FileOutputStream are for binary I/O
  - » just plain bytes in and out with no interpretation as characters or anything else

FileInputStream methods

`int available()`

Returns the number of bytes that can be read from this file input stream without blocking.

`void close()`

Closes this file input stream and releases any system resources associated with the stream.

`protected void finalize()`

Ensures that the close method of this file input stream is called when there are no more references to it.

`FileDescriptor getFD()`

Returns the FileDescriptor object that represents the connection to the actual file in the file system being used by this FileInputStream.

`int read()`

Reads a byte of data from this input stream.

`int read(byte[] b)`

Reads up to b.length bytes of data from this input stream into an array of bytes.

`int read(byte[] b, int off, int len)`

Reads up to len bytes of data from this input stream into an array of bytes.

`long skip(long n)`

Skips over and discards n bytes of data from the input stream.

`void mark(int readlimit)`

Marks the current position in this input stream.

`boolean markSupported()`

Tests if this input stream supports the mark and reset methods.

`void reset()`

Repositions this stream to the position at the time the mark method was last called on this input stream.

"bytes from a file" and "bytes as text"

- Create new FileInputStream and connect it to a specific file
- "decorate" the stream with an InputStreamReader that will do Unicode translation for you

```
InputStreamReader(InputStream in)
    Create an InputStreamReader that uses the default character encoding.

InputStreamReader(InputStream in, String enc)
    Create an InputStreamReader that uses the named character encoding.
```

"bytes from a file as text"

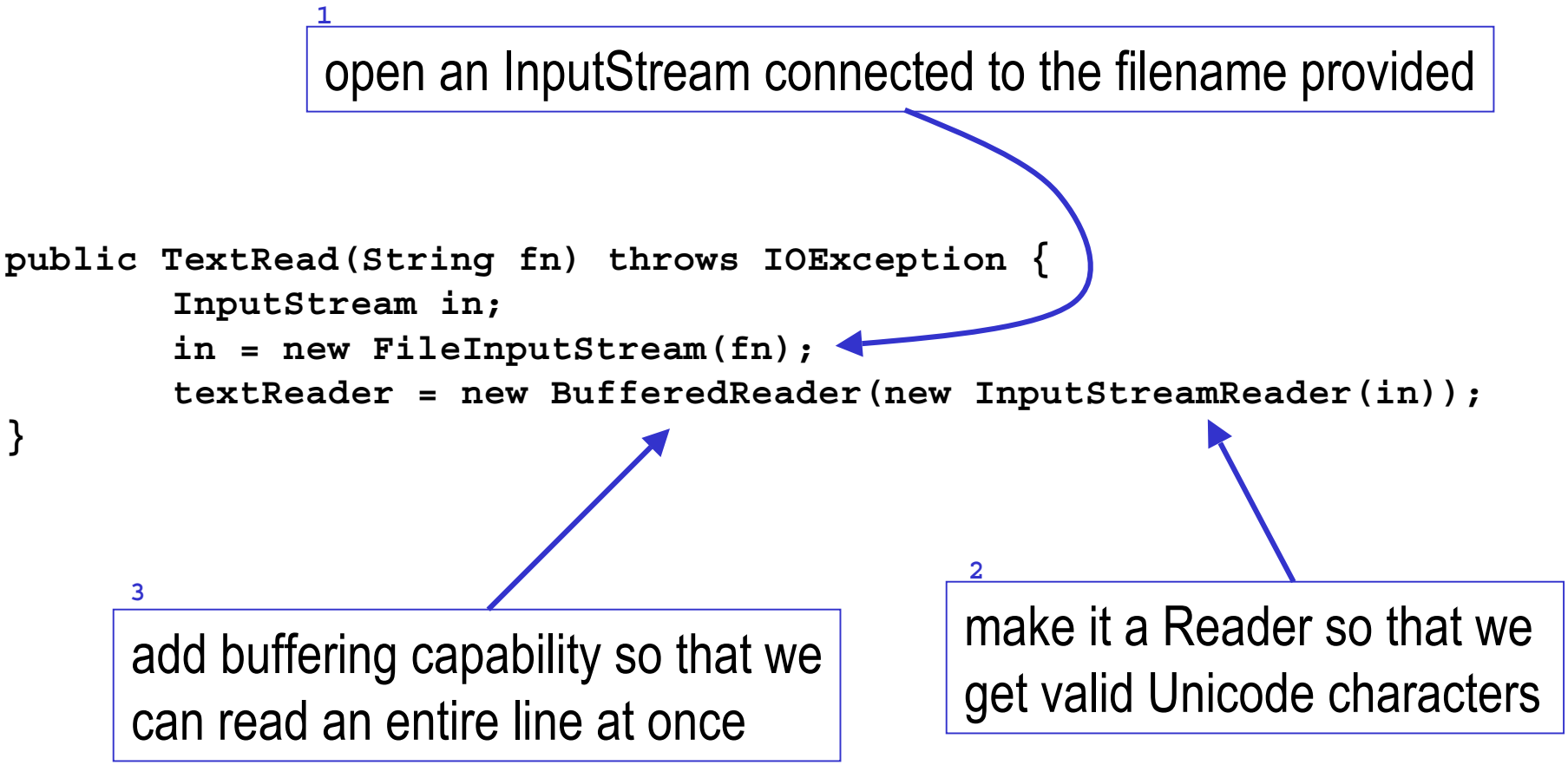
- Create new FileReader and connect it to a file
  - » FileReader is a convenience class for reading character files. The constructors of this class assume that the default character encoding and the default byte-buffer size are appropriate. To specify these values yourself, construct an InputStreamReader on a FileInputStream.

```
FileReader(File file)
    Creates a new FileReader, given the File to read from.

FileReader(FileDescriptor fd)
    Creates a new FileReader, given the FileDescriptor to read from.

FileReader(String fileName)
    Creates a new FileReader, given the name of the file to read from.
```

prepare to read a file



BufferedReader constructor from Sun

```
/**
* Create a buffering character-input stream that uses an input buffer of
* the specified size.
*
* @param  in   A Reader
* @param  sz   Input-buffer size
* @exception  IllegalArgumentException  If sz is <= 0
*/
public BufferedReader(Reader in, int sz) {
    super(in);
    if (sz <= 0)
        throw new IllegalArgumentException("Buffer size <= 0");
    this.in = in;
    cb = new char[sz];
    nextChar = nChars = 0;
}
```

the buffer is allocated here as an array of characters

readline()

- Read one line from a BufferedReader
  - » returns a String containing the contents of the line, not including any line-termination characters, or null if the end of the stream has been reached

```
/**
* Read one line from the text file and return it as a String to the caller.
* Note that the line might be null (at end of file), empty (0 characters) or
* blank (all whitespace).  Of course, it might also be a non-blank String with
* some useful characters in it.
* @return a String containing the next line or null if
* we are at the end of the file
*/
private String getNextLine() throws IOException {
        return textReader.readLine();
}
```

Detecting end of file

- End of file is expected when using readline()
  - » you will eventually read all the characters in a file
- So the method returns `null` if we are end of file
  - » you must check for null after every readline() call

```
String myLine = tr.getNextLine();
while (myLine != null) {
        System.out.println(">> "+myLine);
        myLine = tr.getNextLine();
}
```

close when done

- After reading through the file, you should close the stream, since an open file takes up system resources and prevents other programs from using the file

```
/**
* Close the stream.
*/
public void close() throws IOException {
        textReader.close();
}
```

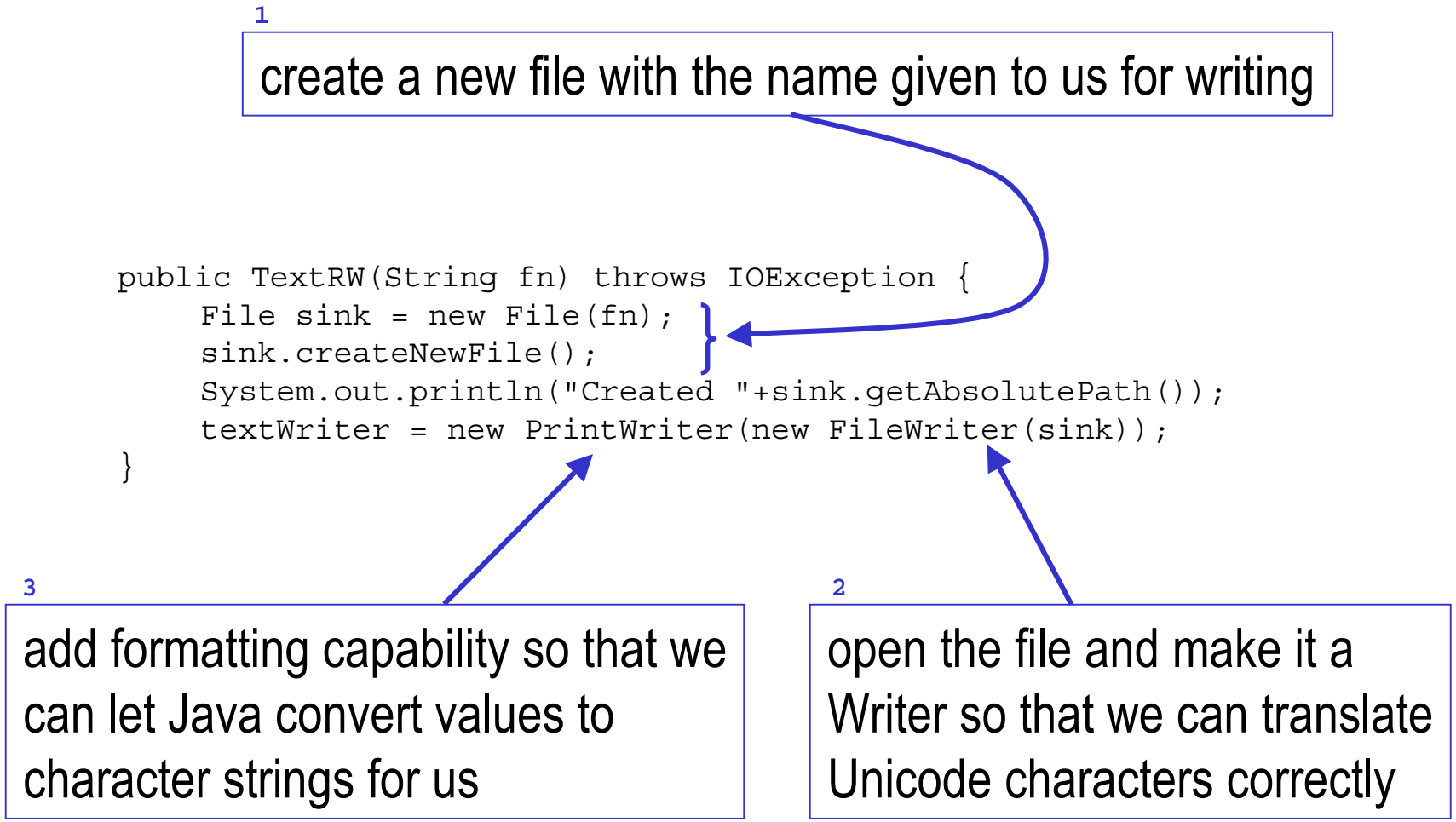
"bytes to a file as text"

- Create new PrintWriter and connect it to a file using a FileWriter
  - » PrintWriter provides the text formatting capabilities
  - » FileWriter provides the connection between the PrintWriter and the actual file

```
PrintWriter(Writer out)
    Create a new PrintWriter, without automatic line flushing.

FileWriter(String fileName)
    Constructs a FileWriter object given a file name.
```

prepare to write a file



println(...)

- Print formatted representations of objects and primitive type to a text-output stream
  - » does not contain methods for writing raw bytes, for which a program should use unencoded byte streams

```
/**
* Write one line on the output file.
* @param line the line of text to write out
*/
public void writeOneLine(String s) {
        textWriter.println(s);
}
```

close when done

- After writing the file, you should close the stream
  - » the last data that you have written may not actually have gotten all the way out to the disk - closing makes sure that the data is flushed to disk
  - » an open file takes up system resources and prevents other programs from using the file

```
/**
* Close the stream.
*/
public void close() throws IOException {
    textWriter.close();
}
```

The File class

- Manages an entry in a directory (a pathname)
- Several constructors are available
  - » File(String pathname)

    pathname string
  - » File(String parent, String child)

    parent pathname string and a child pathname string.
  - » File(File parent, String child)

    parent abstract pathname and a child pathname string.
- The File() constructors create a pathname object in memory, NOT a new file on disk

File class examples

```
File f = new File("c:\autoexec.bat");

File app = new File("c:\apps\JPadPro","JPadPro.exe");

File jppDir = new File("c:\apps\JPadPro");
File jppApp = new File(jppDir, "JPadPro.exe");
```

- Creating a new File object just creates a tool for managing files, it does not create a new file on disk!
  - » Creating a new Dog object did not create a new dog running around the room ...

File class methods

- Create, rename, delete a file
  - » createNewFile(), createTempFile(), renameTo(), delete()
- Determine whether a file exists and access limitations
  - » exists(), canRead(), canWrite()
- Get file info
  - » getParent(), getCanonicalPath(), length(), lastModified()
- Create and get directory info
  - » mkdirs(), list(), listFiles(), getParent()
- Etc, etc