Java Programming 1

CHAPTER 10

Input/Output (I/O)

Contents

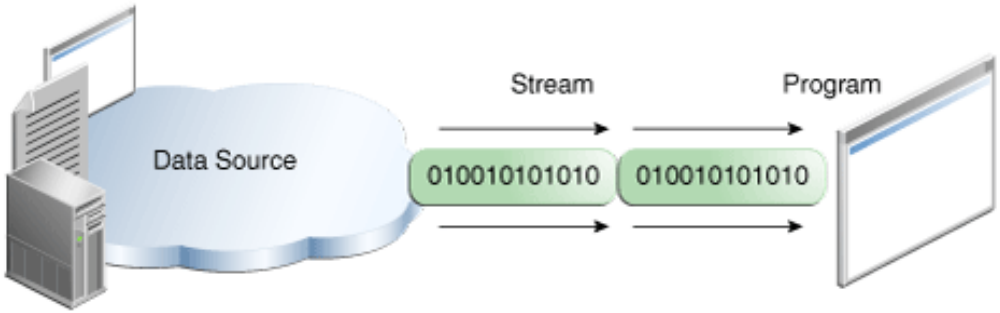
- Input/Output (I/O) Overview
- Streams
  - Byte streams
  - Character streams
  - Buffered streams
  - Data streams
- Summary

Streams around us

- Water moves in one direction
- We cannot move water back (in opposite direction)

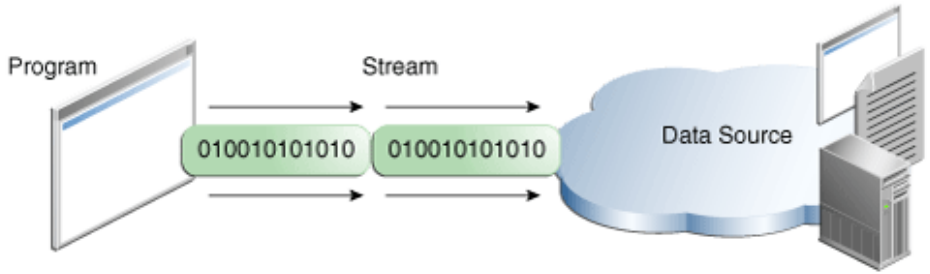
Input streams in programming languages

- A program takes (receives / read) data from the input steam chunk by chunk (portion by portion)
  - from the beginning to the end: in one direction
- The program cannot jump ahead skipping several chunks of data
- The program cannot return back the chunks which have been already read.



Output streams in programming languages

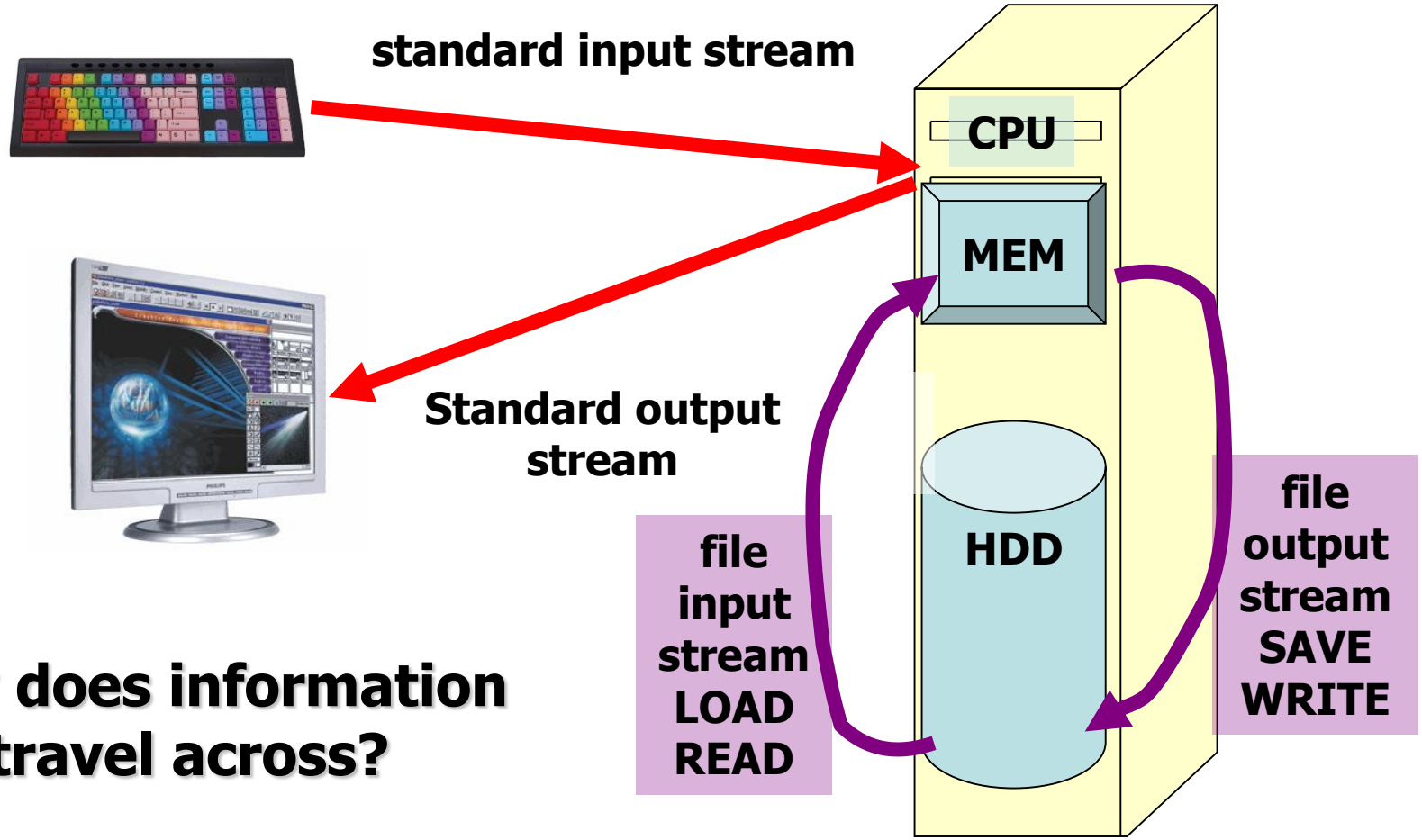
- A program moves (sends / writes) data into the output steam chunk by chunk
  - from the beginning to the end: in one direction
- The program cannot jump ahead skipping several chunks of data
- The program cannot return back the chunks which have been already written.



I/O Overview

- I/O refers to input to and output from programs
- Input can be from keyboard or a file
- Output can be to display (screen) or a file
- Advantages of file I/O
  - permanent copy
  - output from one program can be input to another
  - input can be automated (rather than entered manually)

I/O Overview



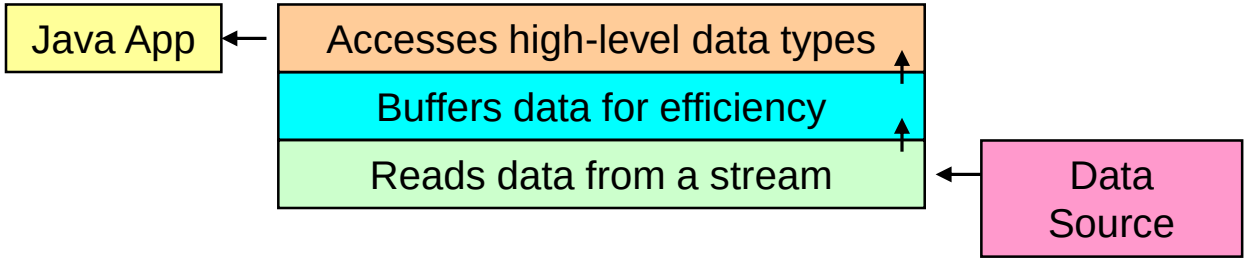
How does information travel across?

Streams

I/O Overview

- Java I/O handling can look complex because it gives great flexibility

- A typical code fragment which opens a file for reading will use multiple layers of functionality



Streams

- ***Stream***: an object that either delivers data to its destination (screen, file, etc.) or that takes data from a source (keyboard, file, etc.)
  - it acts as a buffer between the data source and destination

- A stream connects a program to an I/O object
  - `System.out` connects a program to the screen
  - `System.in` connects a program to the keyboard

Using Streams

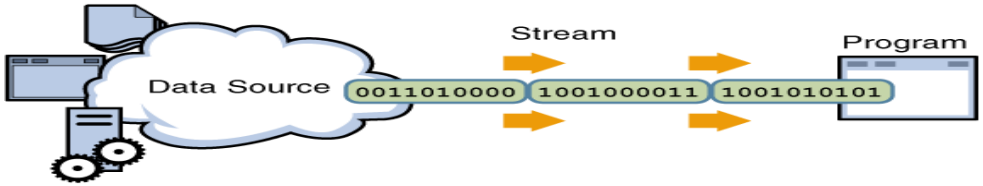
```
import java.io.*;
```

- *Open* the stream
- *Use* the stream (read, write, or both)
- *Close* the stream

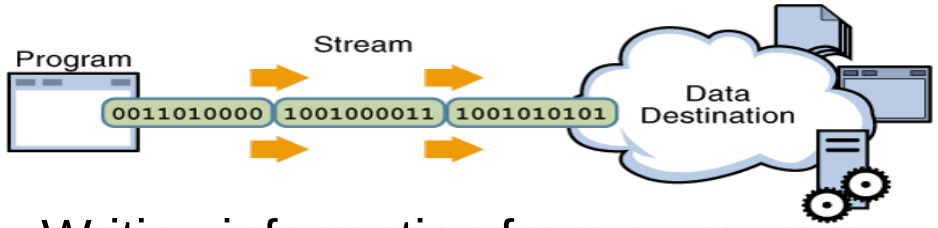
Streams

- ***Input stream***: a stream that provides input to a program
  - Example: `System.in` is an input stream

Reading information into a program



- ***Output stream***: a stream that accepts output from a program
  - Example: `System.out` is an output stream



Writing information from a program

Streams

- A stream is an expensive resource
  - It requires large amount of memory to be allocated
- There is a limit on the number of streams that you can have open at one time
  - You should not have more than one stream open on the same file
- You must close a stream before you can open it again
- *Always close your streams when you finish working with them!*

Streams

Java has two main types of Streams:

Byte Streams (Binary Stream)	Character streams (Text Stream)
Operates on 8 bit (1 byte) data.	Operates on 16-bit (2 byte) unicode characters.
Uses classes: Inputstreams and Outputstreams	Uses classes: Readers and Writers

Streams

Streams in JAVA are Objects, of course !

- Having:

  - 2 types of streams (text / binary) and
  - 2 directions (input / output)

- results in 4 base-classes dealing with I/O:

1. Reader: text-input
2. Writer: text-output
3. InputStream: byte-input
4. OutputStream: byte-output

Streams

InputStream

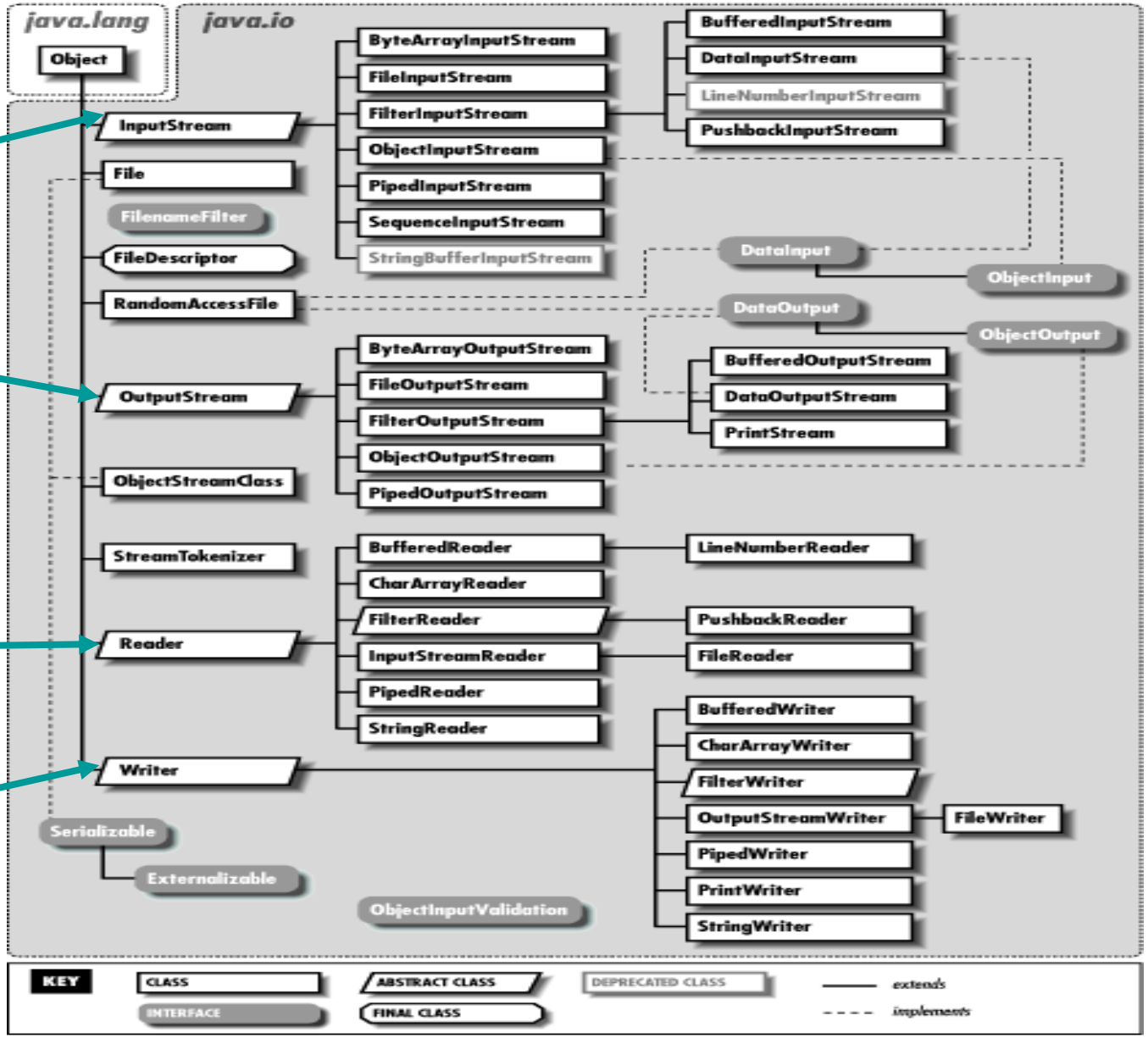
OutputStream

binary

Reader

Writer

text



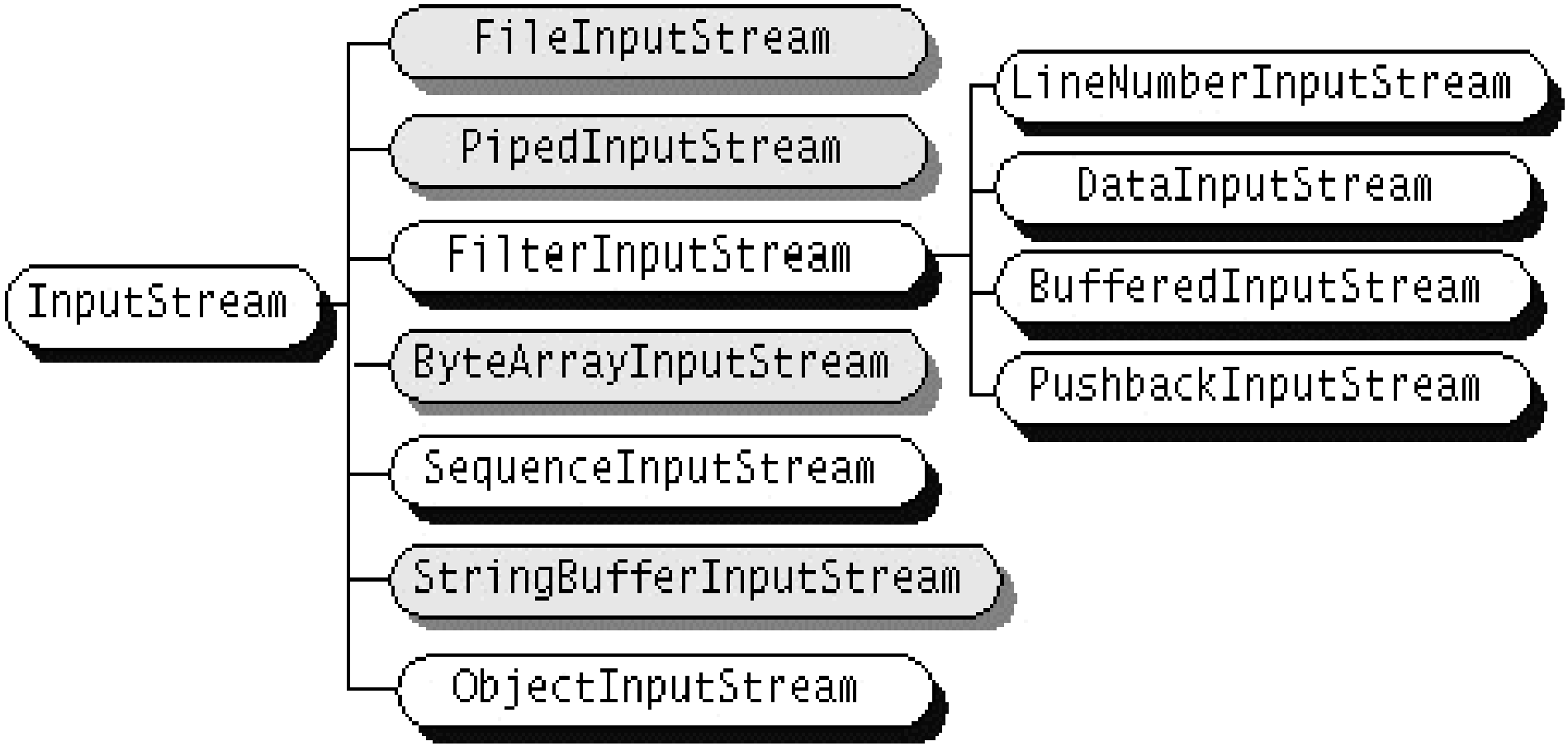
Byte Streams (Binary Streams)

- Byte streams:
  - **InputStream** – abstract superclass for all byte input streams.
  - **OutputStream** – abstract superclass for all byte output streams.

Byte Streams: InputStream

- The **InputStream** abstract class is the superclass of all classes representing an input stream of bytes.
- Applications that define a subclass of **`InputStream`** must always provide a method that returns the next byte of input.
- All methods may throw an **`IOException.`**

Byte Streams: InputStream



FileInputStream: Example

```
import java.io.*;
public class ReadStringsFromFile {
  public static void main(String[] args) {
    File file = new File("ReadStringsFromFile.java"); //create  a file object
    int ch; // variable to store the current byte
    try  {
      FileInputStream fin = new FileInputStream(file);     // 1
      while( (ch = fin.read()) != -1)                      // 2
              System.out.print((char) ch);                 // convert to char and display it
      fin.close();                                         // 3
    }
    catch(FileNotFoundException e) {
      System.out.println("File " + file.getAbsolutePath() +
                         " could not be found on filesystem");
    }
    catch(IOException ioe) {
      System.out.println("Exception while reading the file" + ioe);
    }
    System.out.println("\n Program is finished.");
  }
}    // Output: lines of the ReadStringsFromFile.java file
```

Comments on the Previous Slide

- Line marked with // 1
  - Creates new FileInputStream object. Constructor of FileInputStream throws FileNotFoundException if the agrument *file* does not exist.
- Line marked with // 2
  - To read bytes from stream use, int read() method of FileInputStream class. This method reads a byte from stream. This method returns next byte of data from file or -1 if the end of the file is reached. Read method throws IOException in case of any IO errors.
- Line marked with // 3
  - To close the FileInputStream, use void close() method of FileInputStream class. close method also throws IOException.

Methods of InputStream (1 of 2)

int **read** ( )	Reads the next byte of data from this input stream.
int **read** (byte [ ] b)	Read up to b.length bytes of data from this input stream into an array of byres.
int **read** (byte [ ] b, int off, int len)	Reads up to len bytes of data from this input stream into array of bytes.
void **close** ( )	Closes this input stream and releases any system resources associated with the stream.
int **available** ( )	Returns the number of bytes that can be read from this input stream without blocking.

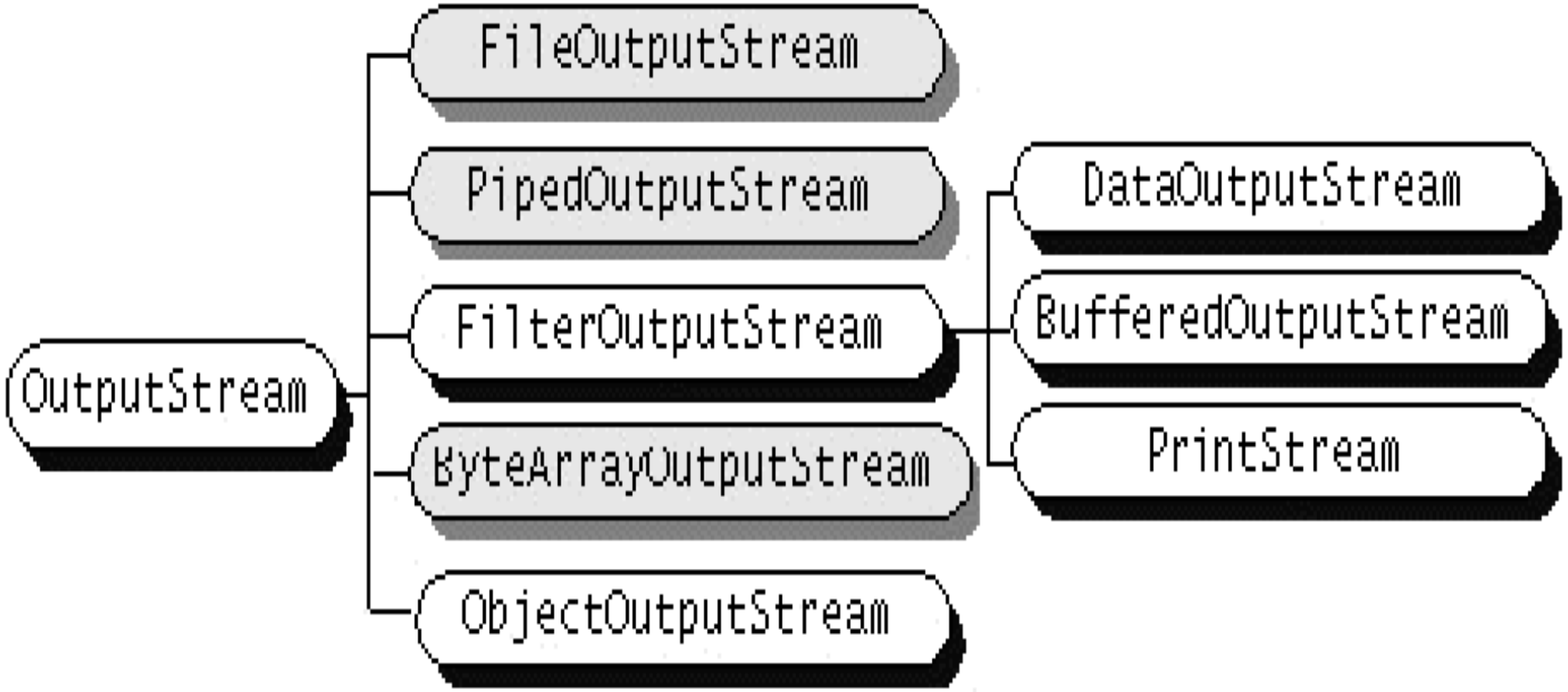
Methods of InputStream (2 of 2)

void **mark** (int)	Marks the current position in this input stream.
void **reset** ( )	Repositions this stream to the position at the time the mark method was last called for this input stream.
long **skip** (long n)	Skips over and discards n bytes of data from this input stream

Byte Streams: OutputStream

- The **OutputStream** is an abstract class.
- The **OutputStream** is the superclass of all classes representing an output stream of bytes.
- Subclasses of **OutputStream** must have at least a method that writes one byte of output .
- All methods may throw an **IOException.**

Byte Streams: OutputStream



FileOutputStream: Example

```
import java.io.*;
public class WriteStringsToFile {
   public static void main(String args[]) {
       String newline = System.getProperty("line.separator");
            // Windows uses "\n", Mac uses "\r", and Linux uses "\r\n" for newline.
        byte b3[] = newline.getBytes();
        try {
            FileOutputStream fout = new FileOutputStream("Example_2.txt");
            String s1 = "This is line 1";
            String s2 = "This is line 2";
            byte b1[] = s1.getBytes();
            byte b2[] = s2.getBytes();
            fout.write(b1);
            fout.write(b3);
            fout.write(b2);
            fout.close();
            System.out.println("File is created");
        } catch (Exception e) {  System.out.println(e);
        }
   }
}
```

output: Two lines in the file Example_2.txt

Methods of OutputStream

write (int b)	Writes the specified byte to this output stream
write (byte [ ] b)	Writes **b. length** bytes from the specified byte array to this output stream
write (byte b [ ] , int off, int len	Writes **len** bytes from the specified byte array starting at offset **off** to this output stream.
close ( )	Closes this output stream and releases any system resources associated with this stream.
flush ( )	Flushes this output stream and forces any buffered output bytes to be written out.

Byte Streams: Example

URL for this code
http://docs.oracle.com/javase/tutorial/essential/io/examples/CopyBytes.java

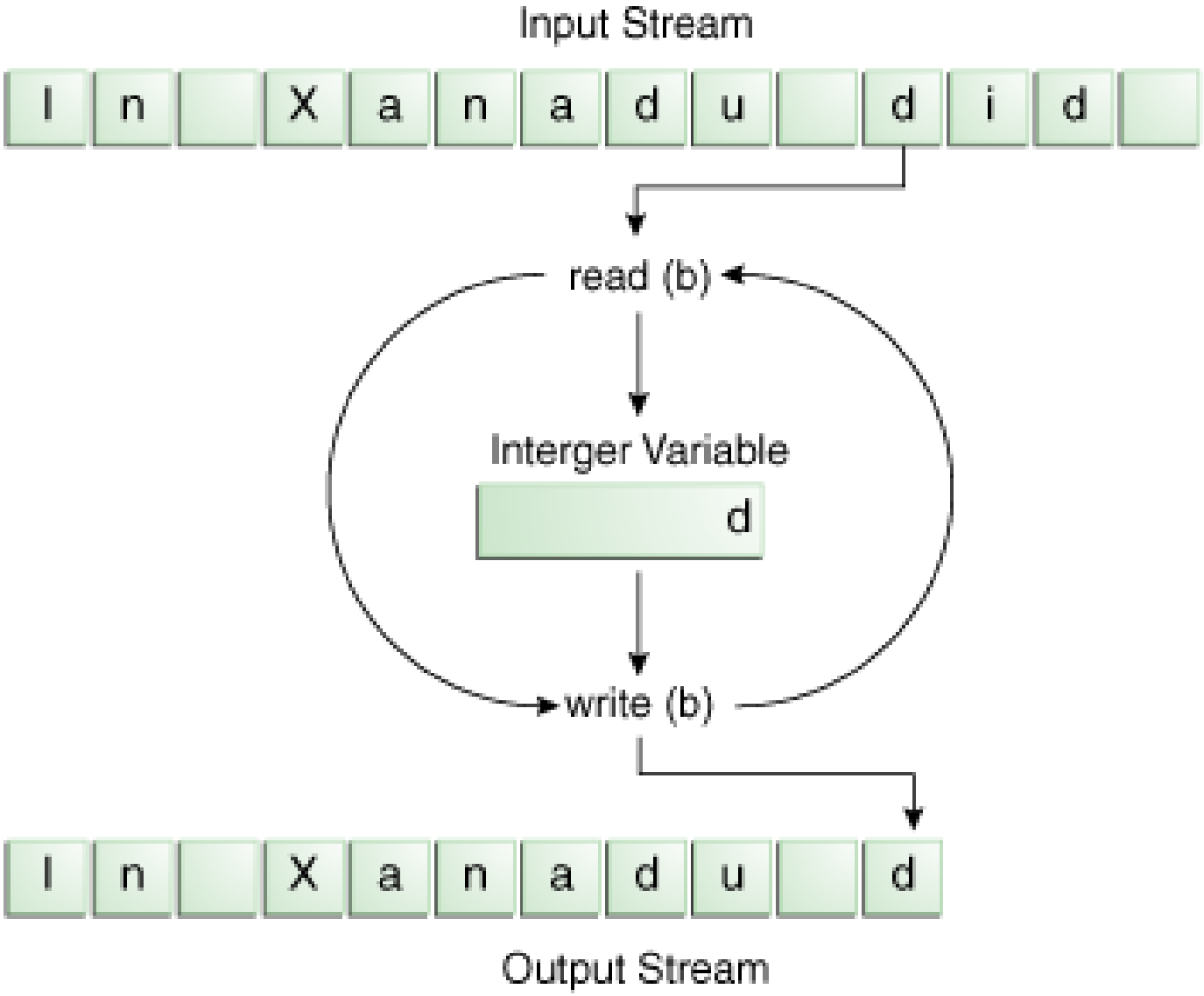
```
import java.io.*;
 public class CopyBytes {
    public static void main(String[] args) throws IOException {
        FileInputStream in = null;
        FileOutputStream out = null;
        try {
            in = new FileInputStream("xanadu.txt");
            out = new FileOutputStream("outagain.txt");
            int b;
            while ((b = in.read()) != -1) {
                out.write(b);
            }
        } finally {
            if (in != null) {
                in.close();
            }
            if (out != null) {
                out.close();
            }
        }
    }
}
```

FileInputStream (byte stream) object from the input file

Reads a byte of data from this input stream.

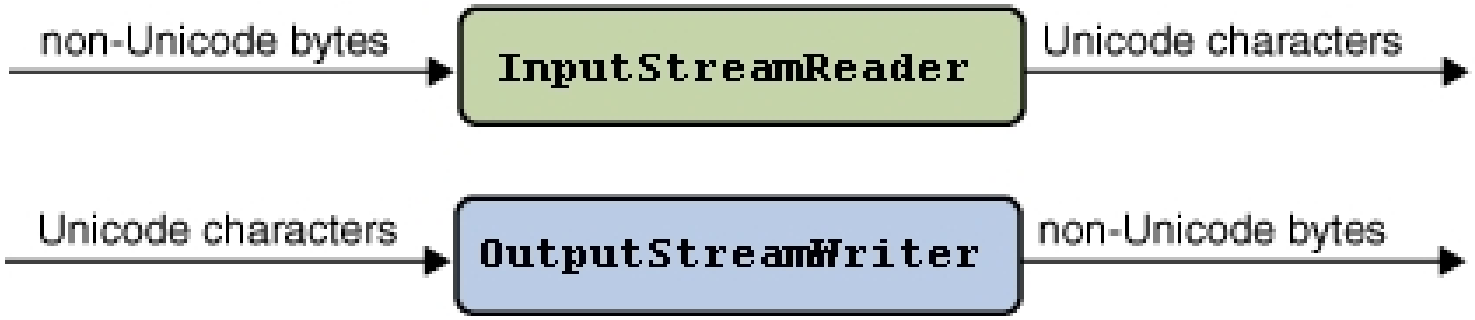
Input Example (xanadu.txt)
In Xanadu did Kubla Khan
A stately pleasure-dome decree:
Where Alph, the sacred river, ran
Through caverns measureless to man
Down to a sunless sea.

Comments on the example

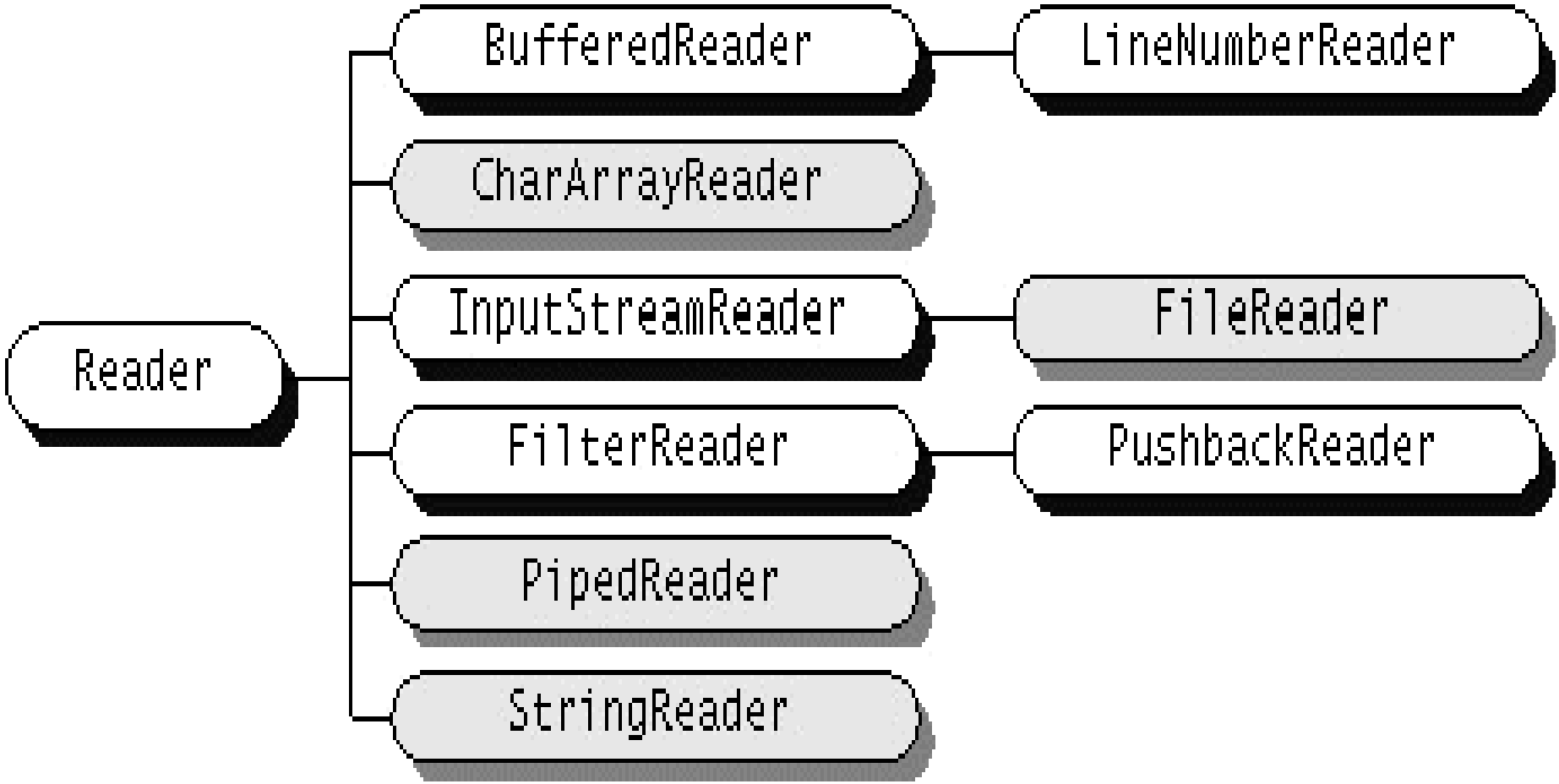


Character Streams

- Character streams:
  - **Reader** – abstract superclass for reading character streams.
  - **Writer** – abstract superclass for writing to character streams.
- Reader and Writer are the superclasses for all character streams.
- Character streams support the Unicode format.



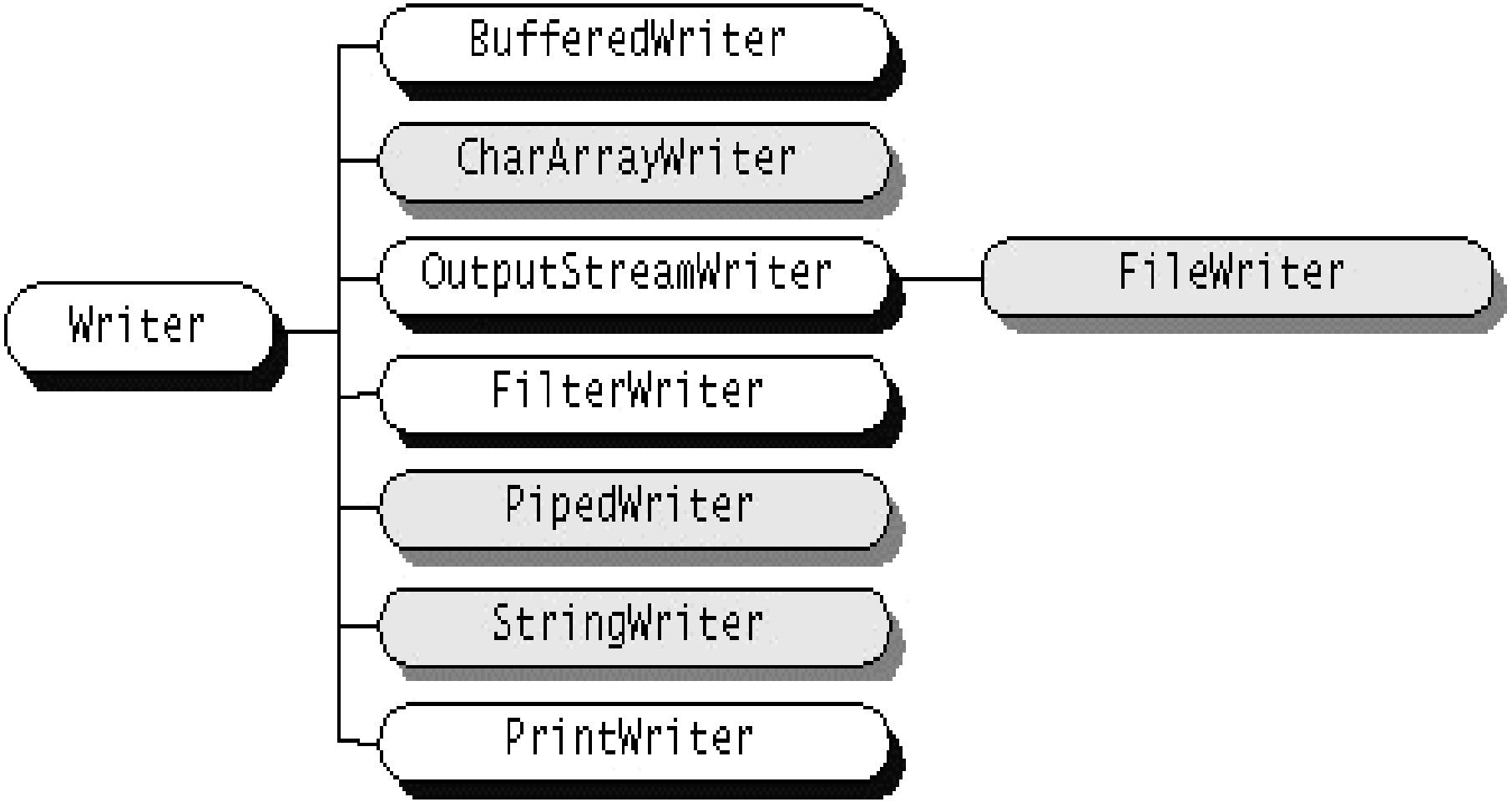
Character Streams: Reader class



Methods of the Reader class

int	**read** ( )	Read a single character.
int	read(char[ ] cbuf	Read characters into an array.
abstract int	**read**(char[ ] cbuf, int off, int len)	Read characters into a portion of an array.
abstract void	**close**( )	Close the stream**.**
void	**mark**(int readAheadLimit)	Mark the present position in the stream.
boolean	**ready**( )	Tell whether this stream is ready to be read.
void	**reset**( )	Reset the stream.
long	**skip**(long n)	Skip characters

Character Streams: Writer class



Methods of the Writer class

void	**write**(int c)	Write a single character.
void	**write**(String str)	Write a string.
void	**write**(String str, int off, int len)	Write a portion of a string
void	**write**(char[ ]cbuf)	Write an array of characters
abstract void	**write**(char[ ] cbuf, int off, int len)	Write a portion of an array of characters.
abstract void	**close** ( )	Close the stream, flushing it first
abstract void	**flush** ( )	Flush the stream

Character Streams: Writing a text file

```
import java.io.*;
public class IOTest {
        public static void main(String[] args) {
                try {
                   FileWriter out = new FileWriter("test.txt");
                   BufferedWriter b = new BufferedWriter(out);
                   PrintWriter p =new PrintWriter(b);
                   for (int i=1; i<6; i++)
                           p.println("I'm sentence " + i + " in a text file.");
                   p.close();
                } catch( Exception e) { }
        }
}
```

Output of this program is a file “test.txt” consisting of 5 lines:

I'm a sentence 1 in a text file.

. . .

I'm a sentence 5 in a text file.

Comments on the Example

- The statement

  FileWriter out = new FileWriter (“test.txt”);

  creates a stream object and associate it with a disk-file

- The statement

  BufferedWriter b = new BufferedWriter (out);

  gives the stream object the desired functionality

- The statements below writes data to the stream:

  PrintWriter p = new PrintWriter (b);

  p.println("I’m a sentence "+ i+ " in a text-file");

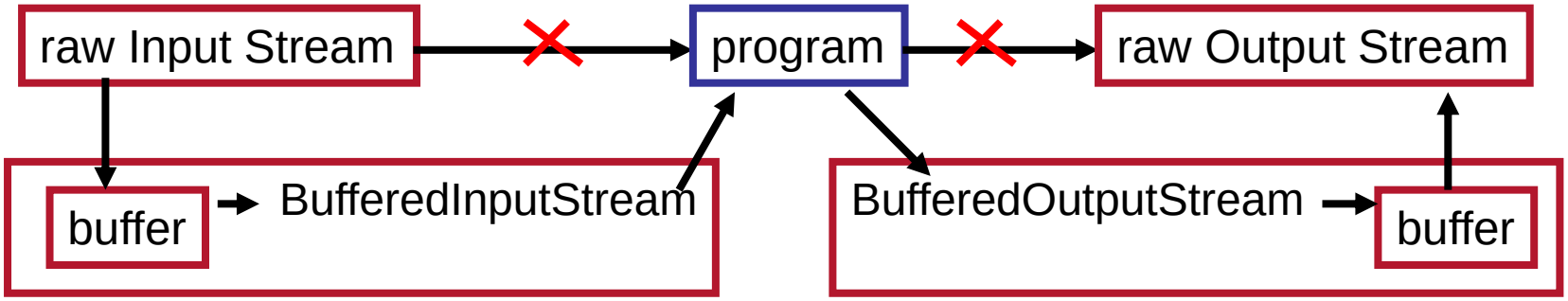
- The statement below closes the stream

  p.close();

Buffered Streams

- Buffered Streams
  - Buffer data while reading or writing, thereby reducing the number of accesses required on the original data source.
  - Typically more efficient than similar non-buffered streams; suggest always wrap a buffer stream around any stream such as a file stream whose read/write operations are costly.
  - BufferedReader f = new BufferedReader(new FileReader("data.txt"));
  - BufferedOutputStream b =

    new BufferedOutputStream (new FileOutputStream("dataout"));



Character Streams: Reading a text file

```
import java.io.*;
public class FileReaderExample {
    public static void main(String[] args) throws IOException {
        BufferedReader a = null;
        try {
              a = new BufferedReader(new FileReader("test.txt"));
              String s;
              while((s = a.readLine())!= null)
                              System.out.println(s);
         } catch (Exception e) {  System.out.println(e);
         } finally  { if (a != null)
                              a.close();
         }
    }
}
```

Output of the program:

Printed on the screen lines from the file created on Slide 34

Selecting tokens in a file

```
import java.io.*;
import java.util.Scanner;
public class ScanFile {
    public static void main(String[] args)
                        throws IOException {
        Scanner s = null;
        try {
            s = new Scanner(new
                BufferedReader(new
                FileReader("test.txt")));
            while (s.hasNext()) {
                System.out.println(s.next());
            }
        } finally {
            if (s != null) {
                s.close();
            }
        }
    }
}
```

- Scanner is useful for breaking down formatted input into tokens and dealing with individual tokens according to their data types.
- ***test.txt*** is a file created by program on Slide 34.
  - java ScanFile

I'm

a

sentence

1

In

. . . . .

Data Streams

- **Motivation**: The methods of `InputStream and OutputStream` are not convenient for working with data such as int, float, string.
  - Our examples on Slides 19, 25, and 27 illustrate this issue.
  - Why did we talk about byte streams? All other streams are built on the byte streams!
- **Solution**: Data streams
  - **DataOutputStream** lets an application to write primitive Java data types to an output stream.
    - —DataOutputStream is a subclass of OutputStream.
  - **DataInputStream** can then be used to read the data back in.
    - —DataInputStream is a subclass of InputStream.

Methods of DataOutputStream class

writeInt (int) writeLong (long) writeByte (int) WriteShort (int) writeDouble (double) writeFloat (float) writeChar (int) writeBoolean (boolean)	Writes an int to the underlying output stream as four bytes, high byte first.
WriteUTF (String)	Writes a Unicode string to the underlying output stream in a machine-independent manner.
Flush ( )	Flushes this data output stream.
Size ( )	Returns the number of bytes written to this data output stream.

DataOutputStream also contains all the methods of **OutputStream**.

Methods of DataInputStream class

int readInt ( ) long readLong ( ) byte readByte ( ) short readShort ( ) float readFloat ( ) double readDouble ( ) char readChar ( ) boolean readBoolean ( )	Reads a signed 32-bit integer from this data input stream.
String readUTF ( )	Reads a Unicode string from this data input stream.
String readLine ( )	Reads the next line of text from this data input stream (Deprecated).

- **DataInputStream** has also all the methods of **InputStream**.
- Most of these methods may throw **IOException** an **EOFException**.

Data Stream: how to create

```
import java.io.*;
 public class CreateDataStream {
    static final String dataFile = "invoicedata";
    static final double[] prices = { 19.99, 9.99, 15.99  };
    static final int[] units = { 12, 8, 13  };
    static final String[] descs = { "Java T-shirt“, "Java Mug“,  "Duke Juggling Dolls"};
    public static void main(String[] args) throws IOException {
        DataOutputStream out = null;
        try {
            out = new DataOutputStream(new
                  BufferedOutputStream(new FileOutputStream(dataFile)));
            for (int i = 0; i < prices.length; i ++) {
                out.writeDouble(prices[i]);
                out.writeInt(units[i]);
                out.writeUTF(descs[i]);
            }
        } finally {
              out.close(); }
    }
}                                  // output of the program is in the invoicedata file
```

Comments on the Previous Slide

- The program creates the ***invoicedata*** file.
- If you use
  - cat invoicedata
- You will see
  - @3ýp£×= Java T-shirt@#úáG® { Java Mug@/úáG® { Duke Juggling Dolls
- Data in the file are in the same way as they are in the memory of computer.
  - There is only a transformation to a platform independent form: If you create a file on a Mac computer, you may read later it on a Windows machine.

Data Stream: How to read

```
import java.io.*;
public class ReadDataStream {
    static final String dataFile = "invoicedata";
    public static void main(String[] args) throws IOException {
        DataInputStream in = null;
        double total = 0.0; // for the total price of the items in the file
        try {
            in = new DataInputStream(new BufferedInputStream(new FileInputStream(dataFile)));
            double price;
            int unit;
            String desc;
            try {  while (true) {
                        price = in.readDouble();
                        unit = in.readInt();
                       desc = in.readUTF();
                       System.out.format("You ordered %d units of %s at $%.2f%n", unit, desc, price);
                       total += unit * price;
                      }
            } catch (EOFException e) { }
            System.out.format("The total price is $%.2f%n", total);
        }
        finally {  in.close();   }
    }
}
```

// output of the program see on the next slide

Comments on the previous slide

- Program reads the ***invoice*** data file. created by the *CreateDataStream* program.
- Output of the program is as follows:

```
java ReadDataStream
You ordered 12 units of Java T-shirts at $19.99
You ordered 8 units of Java Mug at $9.99
You ordered 12 units of Duke Juggling Dolls at  $15.99
The total price is $527.67
```

- Text in read is typed by the user.
- Text in blue printed by the program.

Streams differences

Data on external data medium	Stream	A piece of information to read/write at one I/O operation
Characters	**Character stream** automatically translates the internal Unicode format to and from the local character set. In Western locales, the local character set is usually an 8-bit superset of ASCII.	Character (2 bytes in Unicode)
Characters	**Data stream as a variant of Byte Stream** is used for reading and writing Java primitives, so you are dealing with data in its "natural" form; no conversion when data are read/written; there is only a transformation to a platform independent form: If a file is created on a Mac computer, it can be read on a Windows machine.	Value of the variable without conversion into characters

Command-line Arguments

```
public class Echo {
    public static void main (String[] args) {
        for (String s: args) {
            System.out.println(s);
        }
    }
}
```

Program run:

java Echo Drink Hot Java

Drink
Hot
Java

Input from Keyboard

- See an example: Lecture 12, slides 23 – 25.
- See an example: Lecture 13, slides 14 – 16.

Summary

- Streams provide uniform interface for managing I/O operations in Java irrespective of device types.
- Java supports classes for handling Input Steams and Output steams via java.io package.
- Exceptions supports handling of errors and their propagation during file operations.