Streams, I/O

streams

Sequential input and output uses objects called streams.

Stream is an abstract representation of an input or output device that is a source of, or destination for data.

There are two kinds of streams:

- **character streams (text streams)**

  used for input from **text files and human-readable output to text files**, printers, and so on, using 16-bit Unicode characters

- **byte streams (binary streams)**

  used for compact and efficient **input and output of primitive data** (int, double, …) as well as objects and arrays, in **machine-readable form**

There are separate classes for handling character streams and byte streams:

- for character input and output - `Readers` and `Writers`
- byte input and output - `InputStreams` and `OutputStreams`.

Streams make program code independent of the device involved. With character streams, your program reads and writes Unicode characters, but the file will contain characters in the equivalent character encoding used by the local computer.

streams

java.io	Input Streams	Output Streams
Character Streams	*Reader* BufferedReader LineNumberReader *FilterReader* PushBackReader InputStreamReader FileReader PipedReader CharArrayReader StringReader	*Writer* BufferedWriter *FilterWriter* OutputStreamWriter FileWriter PipedWriter PrintWriter CharArrayWriter StringWriter
Byte Streams	*InputStream* ByteArrayInputStream FileInputStream FilterInputStream BufferedInputStream DataInputStream PushBackInputStream ObjectInputStream PipedInputStream SequenceInputStream StringBufferInputStream	*OutputStream* ByteArrayOutputStream FileOutputStream FilterOutputStream BufferedOutputStream DataOutputStream PrintStream ObjectOutputStream PipedOutputStream
	RandomAccessFile	

java.io classes for Input and Output

Class (Package **java.io**)	Description
File	a pathname either to a file or to a directory
OutputStream	the base class for **byte** stream output operations
InputStream	the base class for **byte** stream input operations
Writer	the base class for **character** stream output operations
Reader	the base class for **character** stream input operations
RandomAccessFile	support for random access to a file

Output Streams

Defining a File – `File` class

File represents a pathname to a physical file or directory (not a stream !)

```
File myDir=new File("F:/jdk1.3/src/java/io");
File myFile=new File("F:/jdk1.3/src/java/io/File.java");
```

```
File myDir=new File("F:/jdk1.3/src/java/io");
File myFile=new File(myDir,"File.java");
```

```
File myFile=new File("F:/jdk1.3/src/java/io","File.java");
```

Testing and checking `File` objects

All operations that involve accessing the files can throw a SecurityException if access is not authorized and security manager exists on local computer(in an applet for instance).

Method	Description
`exists()`	true if the file or directory referred to by the File object exists
`isDirectory()`	true if the File object refers to a directory
`isFile()`	true if the File object refers to a file
`isHidden()`	true if the File object refers to a hidden file
`isAbsolute()`	true if the File object refers to an absolut path name
`canRead()`	true if you are permitted to read the file
`canWrite()`	true if you are permitted to write the file
`equals()`	true for the same paths

Accessing `File` objects

Method	Description
`getName()`	returns String object containing the last name in the path
`getPath()`	returns String object containing path
`getAbsolutePath()`	returns String object containing absolute path
`getParent()`	original path without the last name
`list()`	returns String array containing the names of the members of the directory (null if curent object is file)
`listFiles()`	returns an array of File objects coresponding to the files and directories in that directory
`length()`	returns long value the is the length in bytes for the file represented by the current file object
`getName()`	returns String object containing the last name in the path (file or directory name)
`lastModified()`	Returns a value of type long representing time of last file or directory modification (number of miliseconds from 01.01.1970)
`toString()`	returns String representation of File object
`hashCode()`	Returns hash code value for the current File object

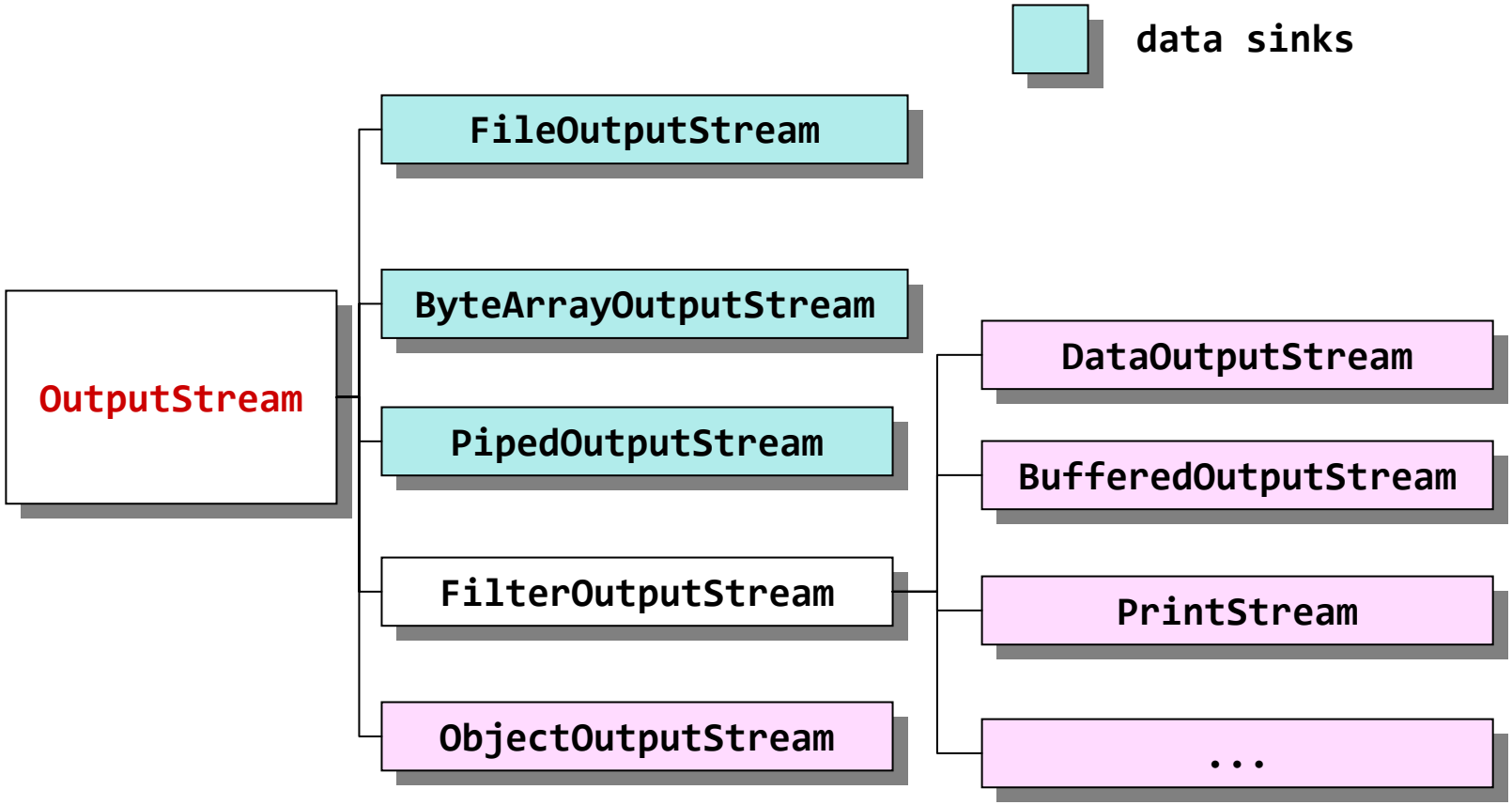
Byte Output Streams

```
public abstract class OutputStream
```

Method	Description
`write(int b)`	(`abstract`) write the low order byte of the argument, b, to the output stream
`write(byte[] b)`	write the array of bytes
`write(byte[] b),int offset, int length)`	writes length bytes from the array b, starting with the element b[offset]
`flush()`	forces any buffered output data to be written to the output stream
`close()`	close the output stream

Byte Output Streams

```
public abstract class OutputStream
```



FileOutputStream class

```
public class FileOutputStream extends OutputStream
```

- služi upisu podataka u fajl
- obratite pažnju na kostruktore, ne postoji default-ni i svaki zahteva fajl u nekom "obliku"

Constructor	Description
`FileOutputStream(String filename)`	Creates an output stream, existing contents will be overwritten, IOexception will be thrown if the file cannot be opened for writing
`FileOutputStream(String filename,boolean append)`	same, excepting data written to the file will be appended if append is true
`FileOutputStream(File file)`	creates a file output stream for the file represented by the object file,…
`FileOutputStream(FileDescriptor desc)`	FileDescriptor object represents an existing connection to file ,…

FileOutputStream class

+ write,flush,close (inherited) and getFD()

FileDescriptor

- FileDescriptor class– represents current connection to the physical file
- can be used to create another byte input stream object (without checking for the file existence)
- method `getFD()` returns an object of type FileDescriptor
- three `public static members` of FileDescriptor class: `in`, `out`, `err` (standard system input, s.s. output, standard error stream)

Primer 1.

```
try {
  FileOutputStream file1 = new FileOutputStream("myFile.txt");
}
catch(IOException e){
  System.out.println(aFile + " not found");
}
```

nije dobra praksa

FileOutputStream class

Primer 2.

```
String filename = "myFile.txt";
File aFile = new File(filename);
try
{
  if(!aFile.exists())
  {
    // open for overwriting
    FileOutputStream file1 = new FileOutputStream(aFile);
    System.out.println("myFile.txt output stream created");
    file1.write(65);
  }
  else
    System.out.println("myFile.txt already exists.");
}
catch(IOException e)
{
  System.out.println("File "+aFile + " not found");
}
```

FileOutputStream class

Primer 3.

```
String filename = "myFile.txt"
File aFile = new File(filename);
try
{
  aFile.createNewFile();   //Create new file if aFile doesn't
                           //exist
  // Open to append
  FileOutputStream file1= new FileOutputStream(aFile,true);

}
catch()
{
  System.out.println(e);
}
```

za ILIJU

http://www.comweb.nl/java/Console/Console.html

iskali ste protumacite ;), ali sad

ByteArrayOutputStream class

Implements an output stream in which the data is written into a byte array.
The buffer automatically grows as data is written to it.
The data can be retrieved using toByteArray() and toString().
byte array operations are very fast

```
ByteArrayOutputStream baos = new ByteArrayOutputStream (  ) ;
PrintStream ps = new PrintStream ( baos ) ;

// write some output
for  ( int i = 0; i  <  1000; i++ )  {
      ps.println ( i + " ABCDEFGHIJKLMNOPQRSTUVWXYZ" ) ;
};
System.out.println(baos.toString());
```

ByteArrayOutputStream class

Field Summary

protected	byte[] buf	The buffer where data is stored.
protected	int count	The number of valid bytes in the buffer.

Constructor Summary

ByteArrayOutputStream()
ByteArrayOutputStream(int size)

Method Summary

void	close()	no effect
void	reset()	Resets the count field to zero
int	size()	Returns the current size of the buffer.
byte[]	toByteArray()	
String	toString()	
void	write(byte[] b, int off, int len)	
void	write(int b)	
void	writeTo(OutputStream out)	Writes the complete contents of this byte array output stream to the specified output stream argument, as if by calling the output stream's write method using out.write(buf, 0, count).

PipedOutputStream class

Used in conjunction with a piped input stream that receives the data written to the output stream
Two independent program threads can communicate with each other by using piped streams

Constructor Summary

```
PipedOutputStream()
PipedOutputStream(PipedInputStream snk)
```

Method Summary

```
void    close()
void    connect(PipedInputStream snk)
void    flush()
void    write(byte[] b, int off, int len)
void    write(int b)
```

PipedOutputStream class

```
public class TestPipedStreams {

public static void main(String[] args) {
 try {
  PipedInputStream pins = new PipedInputStream();
  PipedOutputStream pouts = new PipedOutputStream(pins);

   byte[] outArray = {'H', 'E', 'L', 'L', 'O'};
   pouts.write(outArray, 0, 5);
   System.out.println("Wrote "+new String(outArray)+" to pouts");

   byte[] inArray = new byte[5];
   pins.read(inArray, 0, 5);
   System.out.println("Read "+new String(inArray)+" from pins\n");
   }
  catch (Exception e) { e.printStackTrace(); }
```

FilterOutputStream class

- superclass of all classes that filter output streams
- These streams sit on top of an already existing output stream (the underlying output stream) which it uses as its basic sink of data, but possibly transforming the data along the way or providing additional functionality.
- You must first create an object of type
  - `FileOutputStream`,
  - `PipedOutputStream` or
  - `ByteArrayStream` (sinks!),

  and then use this object in the constructor for the filter output stream class.

Field Summary

`protected OutputStream out`

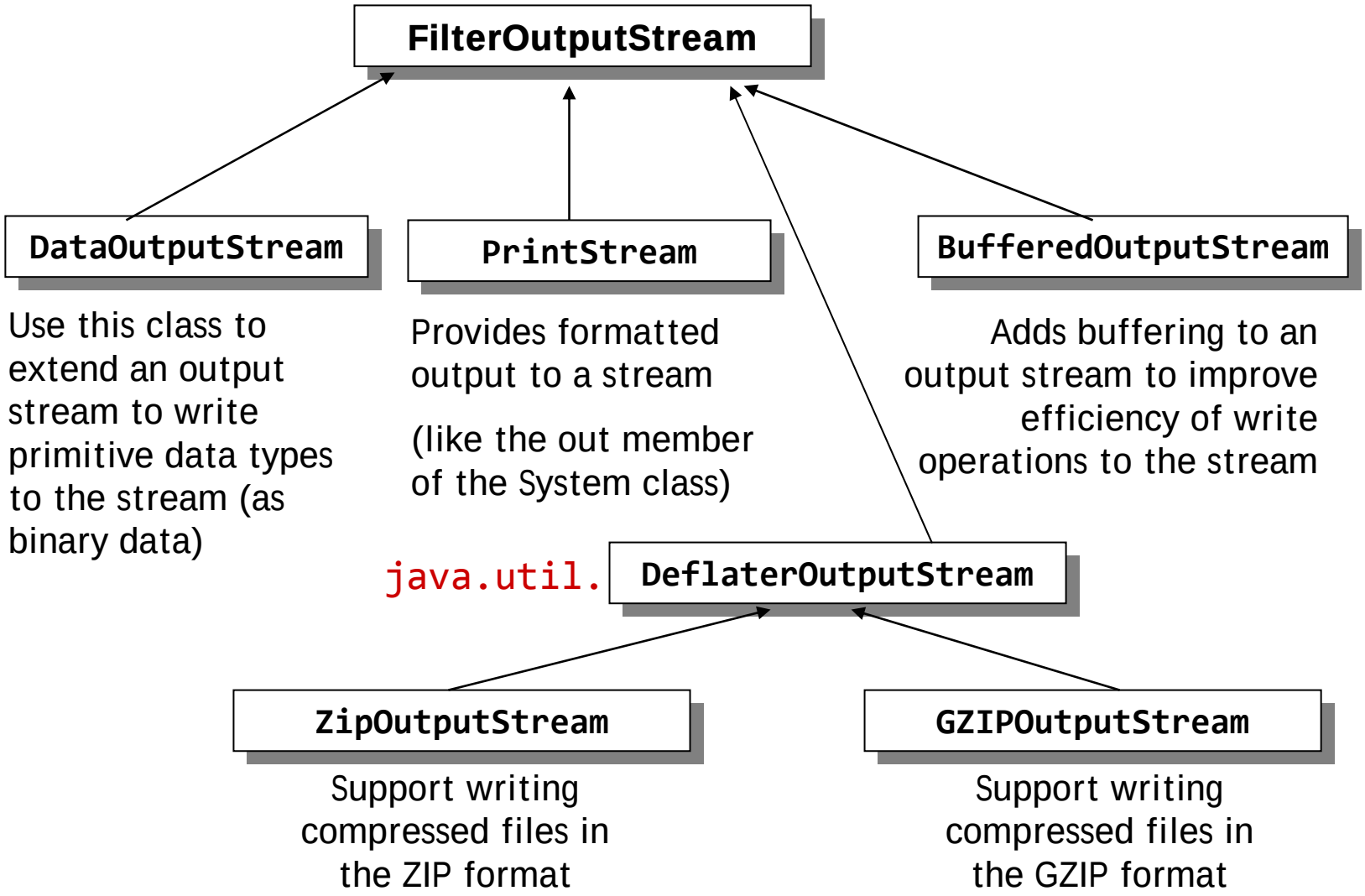
The underlying output stream to be filtered.

Constructor Summary

`FilterOutputStream(OutputStream out)`

Creates an output stream filter built on top of the specified underlying output stream.

FilterOutputStream class



PrintStream class

- Up to now we have made extensive use of `println()` method from the PrintStream class in our examples
- PrintStream has largely been made obsolete by the PrinterWriter class (Java 1.1)
- It has problems with Unicode conversions
- You can still use `System.out.println()` for text output to screen without any problems

DataOutputStream class

- Provides methods to write any of the basic types of data or a String to a byte stream

Field Summary

`protected int written`	The number of bytes written.
`out`	inherited

Constructor Summary

`DataOutputStream(OutputStream out)`

Method Summary

```
void flush()
int  size()
void write(byte[] b,int off,int len)
void write(int b)
void writeBoolean(boolean v)
void writeByte(int v)
void writeBytes(String s)
void writeChar(int v)
void writeChars(String s)
void writeDouble(double v)
void writeFloat(float v)
void writeInt(int v)
void writeLong(long v)
void writeShort(int v)
void writeUTF(String str)
```

DataOutputStream class

```
import java.io.*;

public class TestDataStream
{
  public static void main(String[] args){
    String myStr = new String("Garbage in, garbage out");
    String dirName = "JunkData";   // Directory name
    try {
      File dir = new File(dirName);  // File object for directory
      if(!dir.exists()) dir.mkdir();
          else  if(!dir.isDirectory()) {
                  System.err.println(dirName + " is not a directory");
                  return;
          }
      File aFile = new File(dir, "data.txt");
      aFile.createNewFile();          // Now create a new file if necessary
      DataOutputStream myStream = new DataOutputStream(new
                  FileOutputStream(aFile));
      myStream.writeChars(myStr);     // Write the string to the file
     System.out.println(myStream.size());
    }
    catch(IOException e) {
      System.out.println("IO exception thrown: " + e);}
  }
}
```

BufferedOutput class

- reduces number of actual output operations to the file
- written to the output when the buffer is full or with calling the `flush()` method or when the stream is closed
- default buffer size is 512 bytes

Field Summary

`protected byte[] buf`	The internal buffer where data is stored.
`protected int count`	The number of valid bytes in the buffer.
`out`	inherited from class `java.io.FilterOutputStream`

Constructor Summary

```
BufferedOutputStream(OutputStream out)
BufferedOutputStream(OutputStream out, int size)
```

Method Summary

```
void    flush()
void    write(byte[] b, int off, int len)
void    write(int b)
```

BufferedOutput class

```
import java.io.*;
public class TryPrimesOutput {
  public static void main(String[] args) {
    try {
      String dirName = "JunkData";
       String fileName = "Primes.bin";
      File myPrimeDir = new File(dirName);
      if(!myPrimeDir.exists())  myPrimeDir.mkdir();
         else  if(!myPrimeDir.isDirectory()) {
                  System.err.println(dirName+" is not a directory");
                  return;}
      File primesFile = new File(myPrimeDir, fileName);
      primesFile.createNewFile();
       BufferedOutputStream b=new BufferedOutputStream(new
                              FileOutputStream(primesFile));
      DataOutputStream primesStream = new DataOutputStream(b);
      long[] primes1 = new long[] {2,3,5,17,11,13};
       for(int i=0; i < primes1.length; i++) primesStream.writeLong(primes1[i]);
      // b.flush();
      primesStream.close();
       System.out.println("File size = " + primesStream.size());
    }
  catch(IOException e) { System.out.println("IOException " + e + " occurred");}
  }
}
```

java.util.zip.ZipInputStream

- This class implements an output stream filter for writing files in the ZIP file format. Includes support for both compressed and uncompressed entries.

Fields `buf, def, out` inherited

Constructor Summary `ZipOutputStream(OutputStream out)`

Method Summary

`void close()` Closes the ZIP output stream and the stream being filtered.

`void closeEntry()` Closes the current ZIP entry and positions the stream for writing the next entry.

`void finish()` Finishes writing the contents of the ZIP output stream without closing the underlying stream.

`void putNextEntry(ZipEntry e)` Begins writing a new ZIP file entry and positions the stream to the start of the entry data.

`void setComment(String comment)`

`void setLevel(int level)` Sets the compression level for subsequent entries which are DEFLATED.

`void setMethod(int method)` Sets the default compression method for subsequent entries.

`void write(byte[] b, int off, int len)` Writes an array of bytes to the current ZIP entry data.

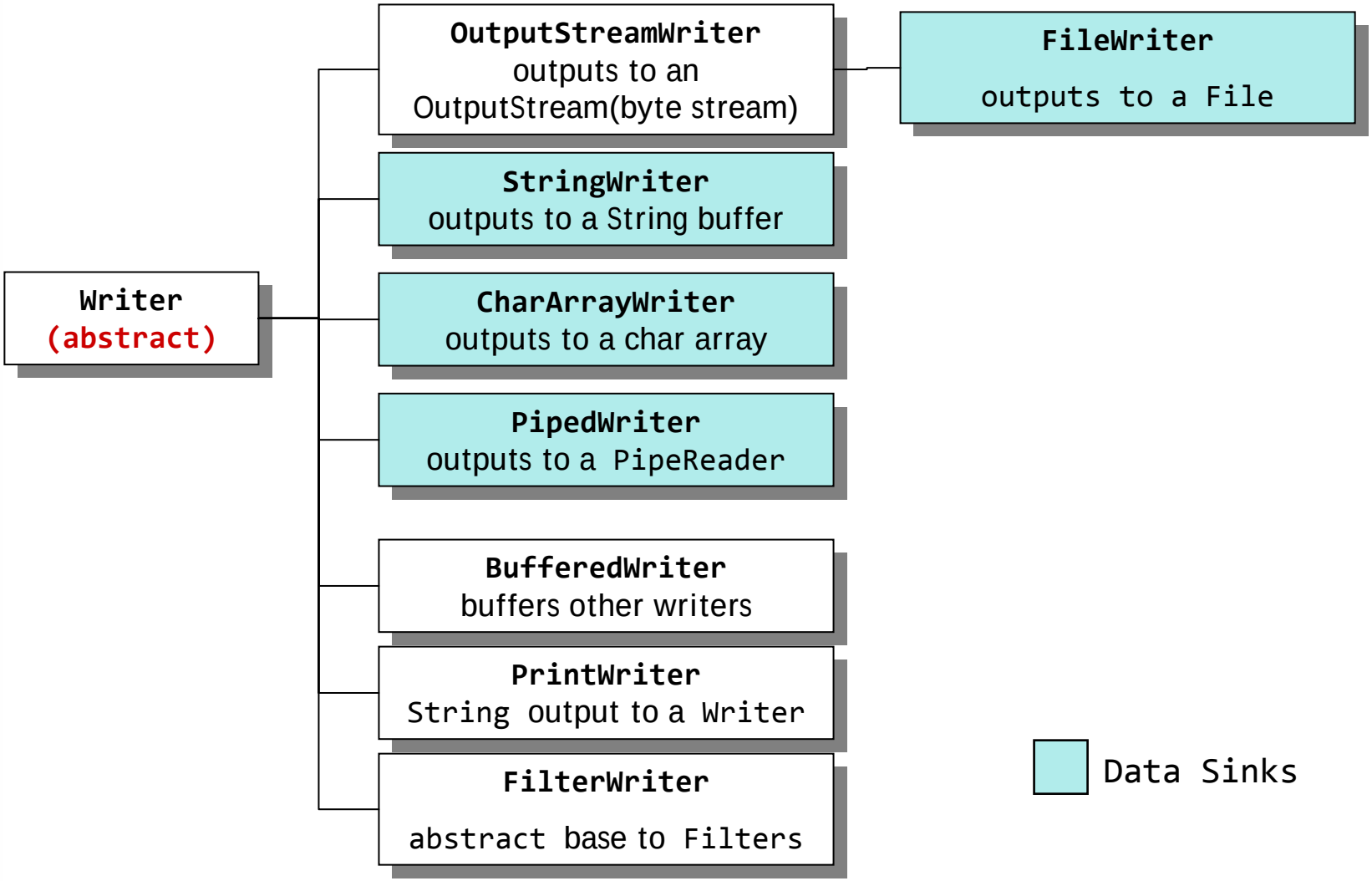
java.util.zip.ZipInputStream

```
String[] filenames = new String[]{"filename1", "filename2"};
byte[] buf = new byte[1024]; // Create a buffer for reading the files
 try {
        // Create the ZIP file
   String outFilename = "outfile.zip";
   ZipOutputStream out = new ZipOutputStream(new
  FileOutputStream(outFilename));

        // Compress the files
   for (int i=0; i<filenames.length; i++) {
            FileInputStream in = new FileInputStream(filenames[i]);

            // Add ZIP entry to output stream.
            out.putNextEntry(new ZipEntry(filenames[i]));
            // Transfer bytes from the file to the ZIP file
            int len;
            while ((len = in.read(buf)) > 0) { out.write(buf, 0, len); }

            // Complete the entry
            out.closeEntry();
            in.close();
        }
         // Complete the ZIP file
        out.close();
    } catch (IOException e) {
    }
```

Character Output Streams

- Writing characters
  - automatic conversion of Unicode characters to the character coding used by the local computer (binary streams use two bytes to write one character)
  - write text to a file or string representation of your data values
- All the classes that provide character operations are derived from the abstract class `Writer`

Character Output Streams



Writer

Abstract class for writing to character streams.

Field Summary

`protected Object lock` used to synchronize operations on this stream

Constructor Summary

`protected Writer()` Creates a new character-stream writer whose critical sections will synchronize on the writer itself.

`protected Writer(Object lock)` Creates a new character-stream writer whose critical sections will synchronize on the given object.

Method Summary

`Writer append(char c)`

`Writer append(CharSequence csq)`

`Writer append(CharSequence csq, int start, int end)`

`abstract void close()` Closes the stream, flushing it first.

`abstract void flush()` Flushes the stream.

`void write(char[] cbuf)` Writes an array of characters.

`abstract void write(char[] cbuf, int off, int len)`

`void write(int c)`

`void write(String str)`

`void write(String str, int off, int len)`

StringWriter

Class for writing to string buffer.

Constructor Summary

`StringWriter()` default initial string-buffer size
`StringWriter(int initialSize)`

Method Summary

`... inherited`
`StringBuffer getBuffer()`
`String toString()` Return the buffer's current value as a string.

```
StringWriter sw=new StringWriter();
sw.write("It's the end");
sw.write(" of the beginning of Java");
System.out.println(sw.toString());
```

CharArrayWriter

Class for writing to char array.

Constructor Summary

`StringWriter()` default initial string-buffer size
`StringWriter(int initialSize)`

Method

`... inherited`

`writeTo(Writer out)` Writes the contents of the buffer to another character stream.

`toCharArray()` Returns the copy of the data in the buffer as type char[]

`reset()` Reset the current buffer in the stream object

`size()` returns the size of current buffer as type int

`String toString()` Return the buffer's current value as a string

```
CharArrayWriter sw=new CharArrayWriter();
sw.write('a');
sw.write("bcd",0,3);
char[] car={'e','f'};
sw.write(new char[] {'e','f'},0,2);
System.out.println(sw.toString());
```

Connecting Character Stream to a Byte Stream

- An `OutputStreamWriter` is a bridge from character streams to any of byte streams
- Characters written to it are translated into bytes according to a specified character encoding. The encoding that it uses may be specified by name, or the platform's default encoding may be accepted.

```
Writer out = new BufferedWriter(new OutputStreamWriter(System.out));
```

Constructor Summary

```
OutputStreamWriter(OutputStream out)
OutputStreamWriter(OutputStream out, Charset cs)
OutputStreamWriter(OutputStream out, CharsetEncoder enc)
OutputStreamWriter(OutputStream out, String charsetName)
```

Using a character stream to write a file

- An `OutputStreamWriter` is a bridge from character streams to any of byte streams
- Characters written to it are translated into bytes according to a specified character encoding. The encoding that it uses may be specified by name, or the platform's default encoding may be accepted.

```
Writer out = new BufferedWriter(new OutputStreamWriter(System.out));
```

Field Summary

`lock` inherited from `class java.io.Writer`

Constructor Summary

```
FileWriter(File file)
FileWriter(File file, boolean append)
FileWriter(FileDescriptor fd)
FileWriter(String fileName)
FileWriter(String fileName, boolean append)
```

Using a character stream to write a file

```
import java.io.*;
class FileWrite
{
   public static void main(String args[])
  {
      try{
    // Create file
    FileWriter fstream = new FileWriter("out.txt");
        BufferedWriter out = new BufferedWriter(fstream);
    out.write("Hello Java");
    //Close the output stream
    out.close();
    }catch (Exception e){//Catch exception if any
      System.err.println("Error: " + e.getMessage());
    }
  }
}
```

PrintWriter

- Prints formatted representations of objects to a text-output stream. This class implements all of the print methods found in PrintStream. It does not contain methods for writing raw bytes.
- if automatic flushing is enabled it will be done only when one of the println, printf, or format methods is invoked

Field Summary

```
protected  Writer out
lock                          inherited from class java.io.Writer
```

Constructor Summary

```
PrintWriter(File file)
PrintWriter(File file, String csn)
PrintWriter(OutputStream out)
PrintWriter(OutputStream out, boolean autoFlush)
PrintWriter(String fileName)
PrintWriter(String fileName, String csn)
PrintWriter(Writer out)  writer should be OutputStreamWriter object
PrintWriter(Writer out, boolean autoFlush)
```

PrintWriter

- `print()` and `println()` methods
- overloaded to accept different argument types(char,int,...,Object)
- PrintWriter methods doesn't throw exceptions (use checkError() method to check for output errors)

Primeri

```
import java.io.*;
class WriteCharacters {
  public static void main(String[] args) {
    try {
      String dirName = "JunkData";
      String fileName = "Proverbs.txt";
      File output = new File(dirName, fileName);
      output.createNewFile();
       if(!output.isFile()) {
        System.out.println("Creating " + output.getPath() + " failed.");
        return;}
      // only this FileWriter constructor allows data appending
      BufferedWriter out = new BufferedWriter(
                              new FileWriter(output.getPath(), true));
      String[] sayings={ "Only the mediocre are always at their best.",
                         "A little knowledge is a dangerous thing.",
                          "Who knows most says least.",};
     for(int i = 0; i < sayings.length; i++) {
        out.write(sayings[i].length() + sayings[i]);}
      out.close();
    }
    catch(IOException e) {
      System.out.println("Error writing the file " + e);
    }
  }
}
```

Input Streams

Byte input streams

- Byte input streams are defined using sub-classes of the abstract class `InputStream`
- Java supports markable streams (position can be marked calling a special method

`Input Stream class`

Constructor Summary

`InputStream()`

Method Summary

`int available()`

`void close()`

`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.

`abstract int read()`

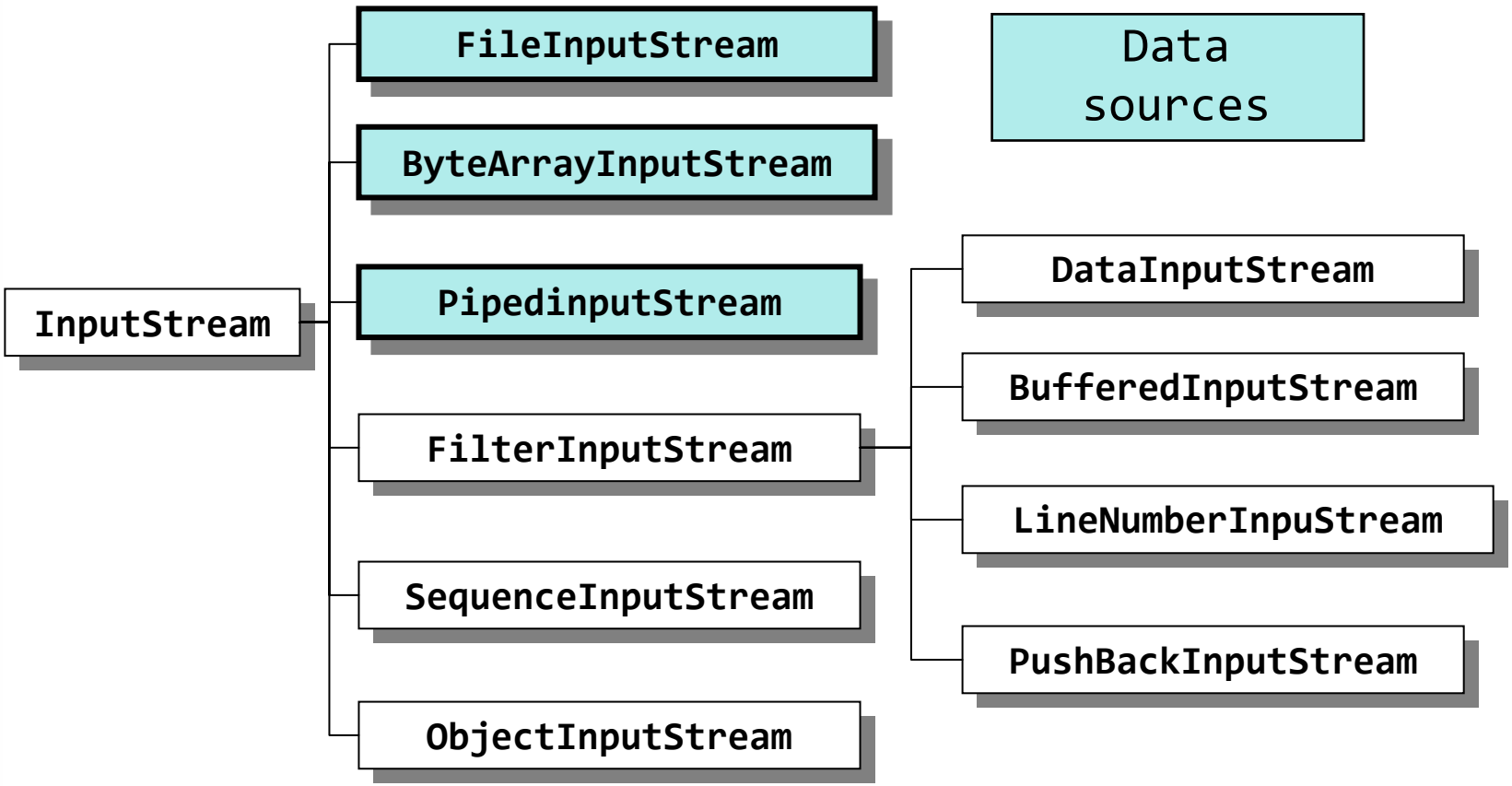
`int read(byte[] b)`

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

`void reset()`

`long skip(long n)`

Byte input streams



Byte input streams

- `FileInputStream`: low level facilities for reading data from file
- `PipedInputStream` and `ByteArrayInputStream`: complement to equivalent output classes
- `SequenceInputStream`: allows concatenating several input streams into a single stream
- `FilterInputStream`: class designed to be a base for deriving a number of filter stream classes to add more flexibility in the way data input is handled(other classes allow only byte manipulation)
- ObjectInputStream: reading objects

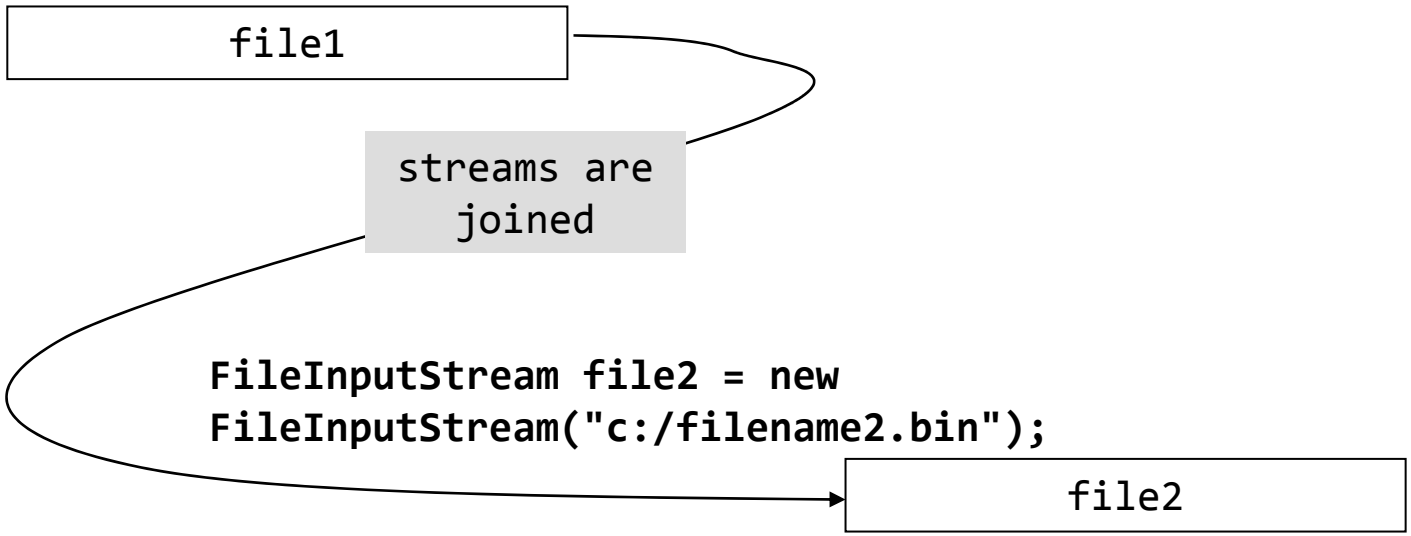
FileInputStream

- Throws `FileNotFoundException` if the file doesn't exist
- Only available are `read()` methods for byte or byte array reading (use filter input stream classes to add functionality)
- method `getFD()` returns FileDescriptor object

```
File myPrimes = new File("c:/JavaCourse/CourseJunkData/Primes.bin");
FileInputStream myStream= new FileInputStream(myPrimes);
```

SequenceInputStream

```
FileInputStream file1 = new FileInputStream("c:/filename1.bin");
```




```
SequenceInputStream comb = new SequenceInputStream(file1,file2);
```

- transition from one concatenated stream to the next is automatic
- when the end of one stream is reached, its close method will be called, and data will then be read from the next stream
- no means of detecting when this occurs