# Streams and Files

The objectives of this chapter are:

- To understand the principles of I/O streams and where to use them
- To understand the options and limitations of I/O streams
- To become comfortable with the mechanisms for accessing the file system

# What are Streams?

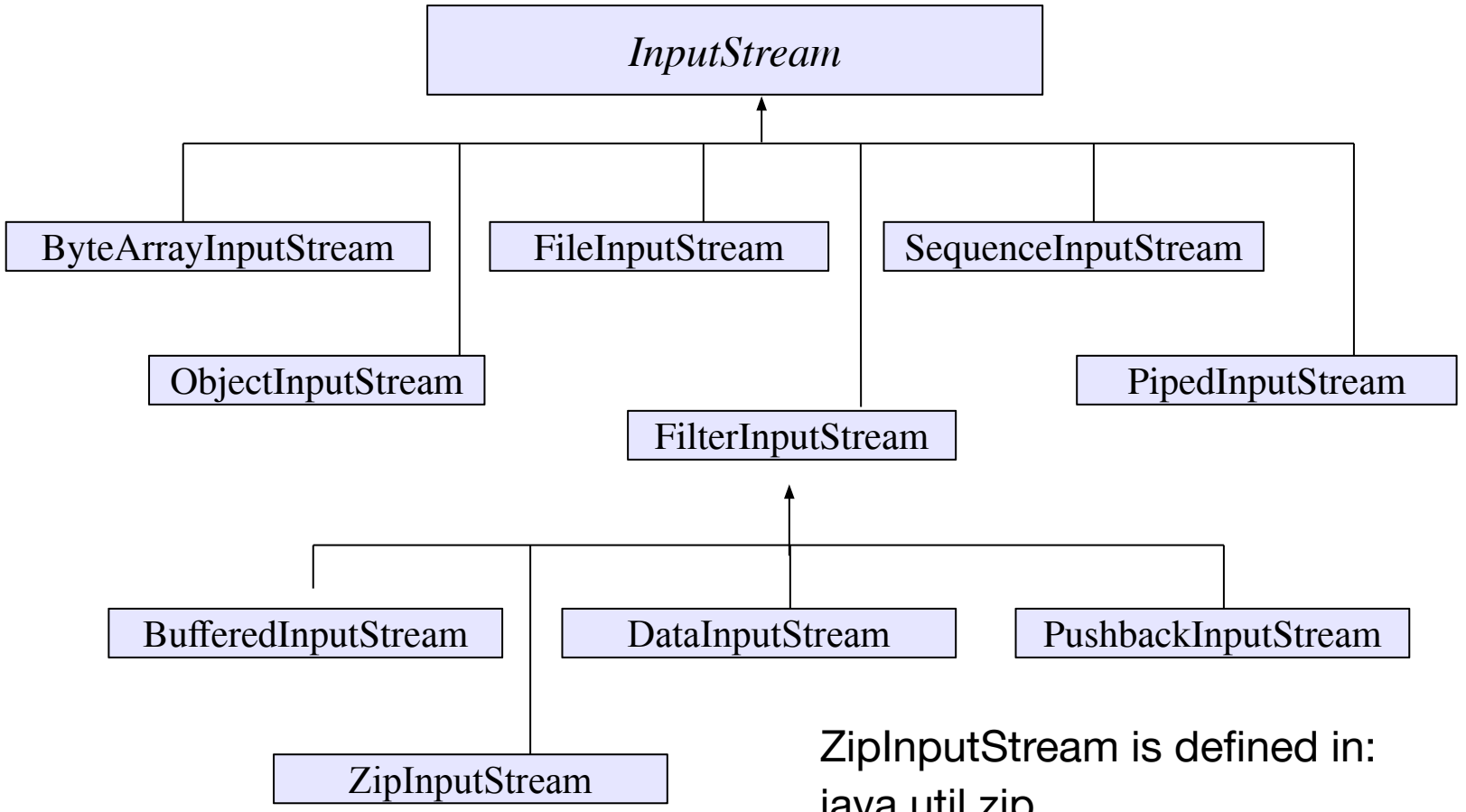
- The I/O System in Java is based on Streams
  - Input Streams are data sources
    - Programmers read data from input streams
  - Output Streams are data sinks
    - Programmers write data to output streams
- Java has two main types of Streams
  - Byte Oriented
    - Each datum is a byte
    - uses InputStream class hierarchy & OutputStream class hierarchy
  - Character-based I/O streams
    - each datum is a Unicode character
    - uses Reader class hierarchy & Writer class hierarchy

# Byte Oriented Streams

- There are many different types of Byte-Oriented Streams
  - Represented by different classes within the java.io.package
  - All byte-oriented streams are subclasses of a common Stream class
    - Input Streams are subclasses of the abstract class java.io.InputStream
    - Output Streams are subclasses of the abstract class java.io.OutputStream
  - All byte-oriented streams inherit basic methods from their respective superclasses
    - Some define new methods pertinent to the type of data they provide.
- Byte-oriented streams are closely related to the I/O streams provided by other programming languages like C, C++, and pascal.
- Because they are byte-oriented they are suitable for reading binary and ASCII data.
  - Byte-oriented streams do not work well with unicode text.
  - Use character oriented streams for unicode.

# Byte-Oriented Input Stream Classes

- **The following is the byte-oriented input stream class hierarchy:**




ZipInputStream is defined in:
java.util.zip

# InputStream Methods

- Reading
  - read() methods will block until data is available to be read
  - two of the three read() methods return the number of bytes read
    - -1 is returned if the Stream has ended
  - throws IOException if an I/O error occurs. This is a checked exception
- There are 3 main read methods:

```
int read()
```

  - Reads a single character.  Returns it as integer

```
int read(byte[] buffer)
```

  - Reads bytes and places them into buffer (max = size of buffer)
  - returns the number of bytes read

```
int read(byte[] buffer, int offset, int length)
```

  - Reads up to length bytes and places them into buffer
  - First byte read is stored in buffer[offset]
  - returns the number of bytes read

# InputStream Methods

- available() method returns the number of bytes which can be read without blocking
- skip() method skips over a number of bytes in the input stream
- close() method will close the input stream and release any system resources
- input streams optionally support repositioning the stream

  can mark the stream at a certain point and 'rewind' the stream to that point later.

- methods that support repositioning are:

| | |
|---|---|
| markSupported() | returns true if repositioning is supported |
| mark() | places a mark in the stream |
| reset() | 'rewinds' the stream to a previously set mark |

# Creating an InputStream

- InputStream is an abstract class
  - Programmers can only instantiate subclasses.

- ByteArrayInputStream:
  - Constructor is provided with a byte array.
  - This byte array contains all the bytes provided by this stream
  - Useful if the programmer wishes to provide access to a byte array using the stream interface.

- FileInputStream:
  - Constructor takes a filename, File object or FileDescriptor Object.
  - Opens a stream to a file.
- FilterInputStream:
  - Provides a basis for filtered input streams
  - Filtered streams are covered later in the chapter.

# Creating an InputStream

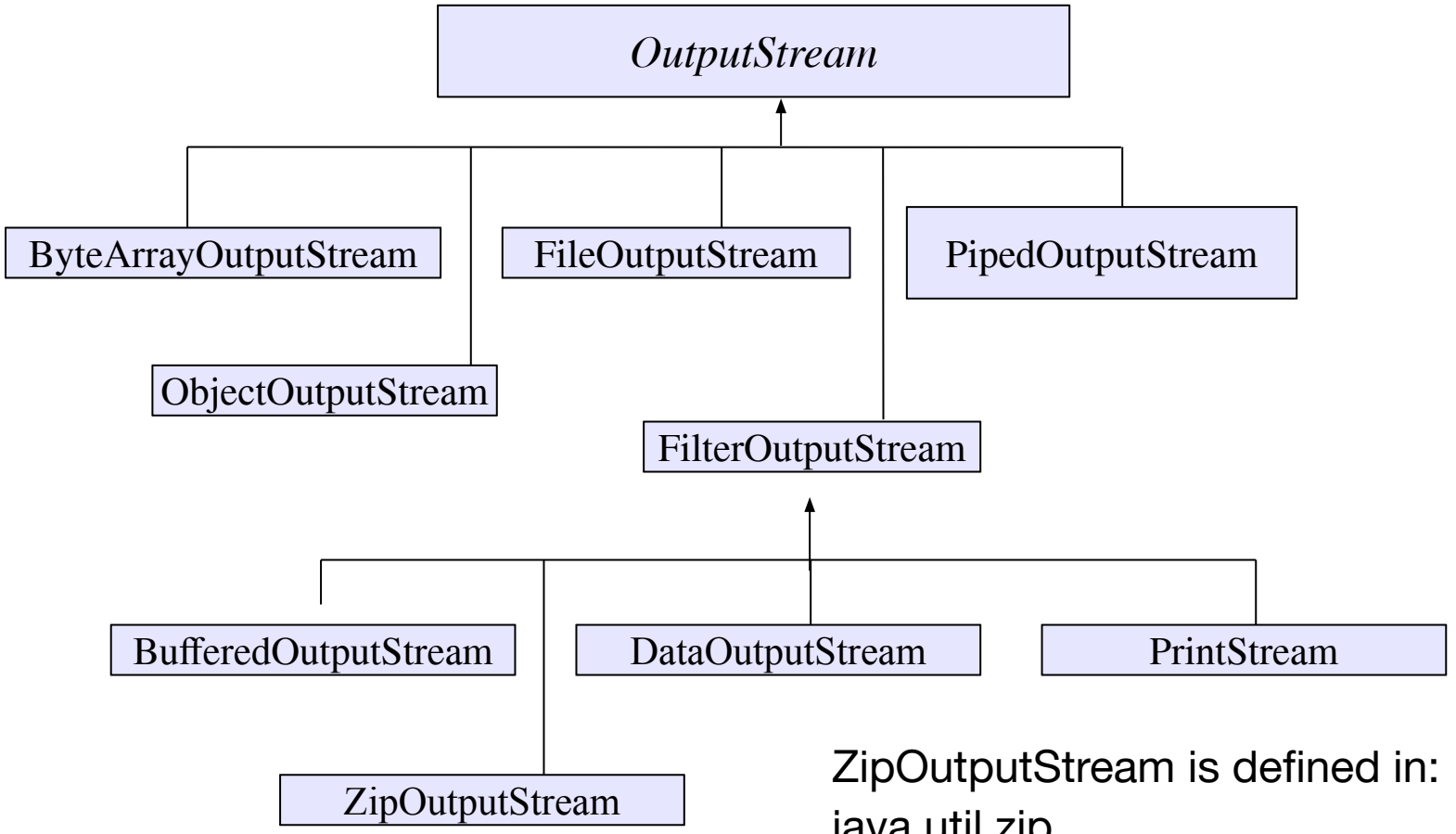
- ObjectInputStream
  - Created from another input stream (such as FileInputStream)
  - Reads bytes from the stream (which represent serialized Objects) and converts them back into Objects
  - More on Serialization later in the Chapter.

- PipedInputStream:
  - Connects to an Instance of PipedOutputStream
  - A pipe represents a one-way stream through which 2 threads may communicate
    - Thread1 writes to a PipedOutputStream
    - Thread2 reads from the PipedInputStream

- SequenceInputStream:
  - Constructor takes multiple InputStreams
  - Allows reading. When one stream ends, it continues reading from next stream in the list

# Byte-Oriented Output Stream Classes

- **The following is the byte-oriented input stream class hierarchy:**




ZipOutputStream is defined in:
java.util.zip

# OutputStream Methods

- Writing*:*
  - write() methods write data to the stream. Written data is buffered.
  - Use flush() to flush any buffered data from the stream.
  - throws IOException if an I/O error occurs. This is a checked exception
- There are 3 main write methods:

`void write(int data)`

- Writes a single character
- Note: even though data is an integer, data must be set such that:
  - 0 <= data <= 255

`void write(byte[] buffer)`

- Writes all the bytes contained in buffer to the stream

`void write(byte[] buffer, int offset, int length)`

- Writes length bytes to stream starting from buffer[offset]

# OutputStream Methods

- flush()
  - To improve performance, almost all output protocols buffer output.
  - Data written to a stream is not actually sent until buffering thresholds are met.
  - Invoking flush() causes the OutputStream to clear its internal buffers.
- close()
  - Closes stream and releases any system resources.

# Creating an OutputStream

- OutputStream is an abstract class.
  - Programmers instantiate one of its subclasses
- ByteArrayOutputStream:
  - Any bytes written to this stream will be stored in a byte array
  - The resulting byte array can be retrieved using toByteArray() method.
- FileOutputStream:
  - Constructor takes a filename, File object, or FileDescriptor object.
  - Any bytes written to this stream will be written to the underlying file.
  - Has one constructor which allows for appending to file:
    `FileOutputStream(String filename, boolean append)`
- FilterOutputStream:
  - Provides a basis for Output Filter Streams.
  - Will be covered later in chapter.

# Creating an OutputStream

- ObjectOutputStream
  - Created from another output stream (such as FileOutputStream)
  - Programmers serialize objects to the stream using the writeObject() method
  - More on Serialization later in the Chapter.

- PipedOutputStream:
  - Connects to an Instance of PipedInputStream
  - A pipe represents a one-way stream through which 2 threads may communicate
    - Thread1 writes to a PipedOutputStream
    - Thread2 reads from the PipedInputStream

# Example - Copy a File

```
import java.io.*;

public class CopyFile
{
  public void copyFile(String inputFilename, String outputFilename)
  {
    try
    {
        FileInputStream fpin = new FileInputStream(inputFilename);
        FileOutputStream fpout = new FileOutputStream(outputfilename);
        byte buffer = new byte[8192];
        int length = 0;
        while ((length = fpin.read(buffer, 0, buffer.length)) > 0)
        {
            fpout.write(buffer, 0, length);
        }
        fpout.flush();
        fpout.close();
        fpin.close();
    }
    catch (IOException x)
    {
        System.out.println("Error:" + x);
    }
  }
}
```

# Limitations of Byte Oriented Streams

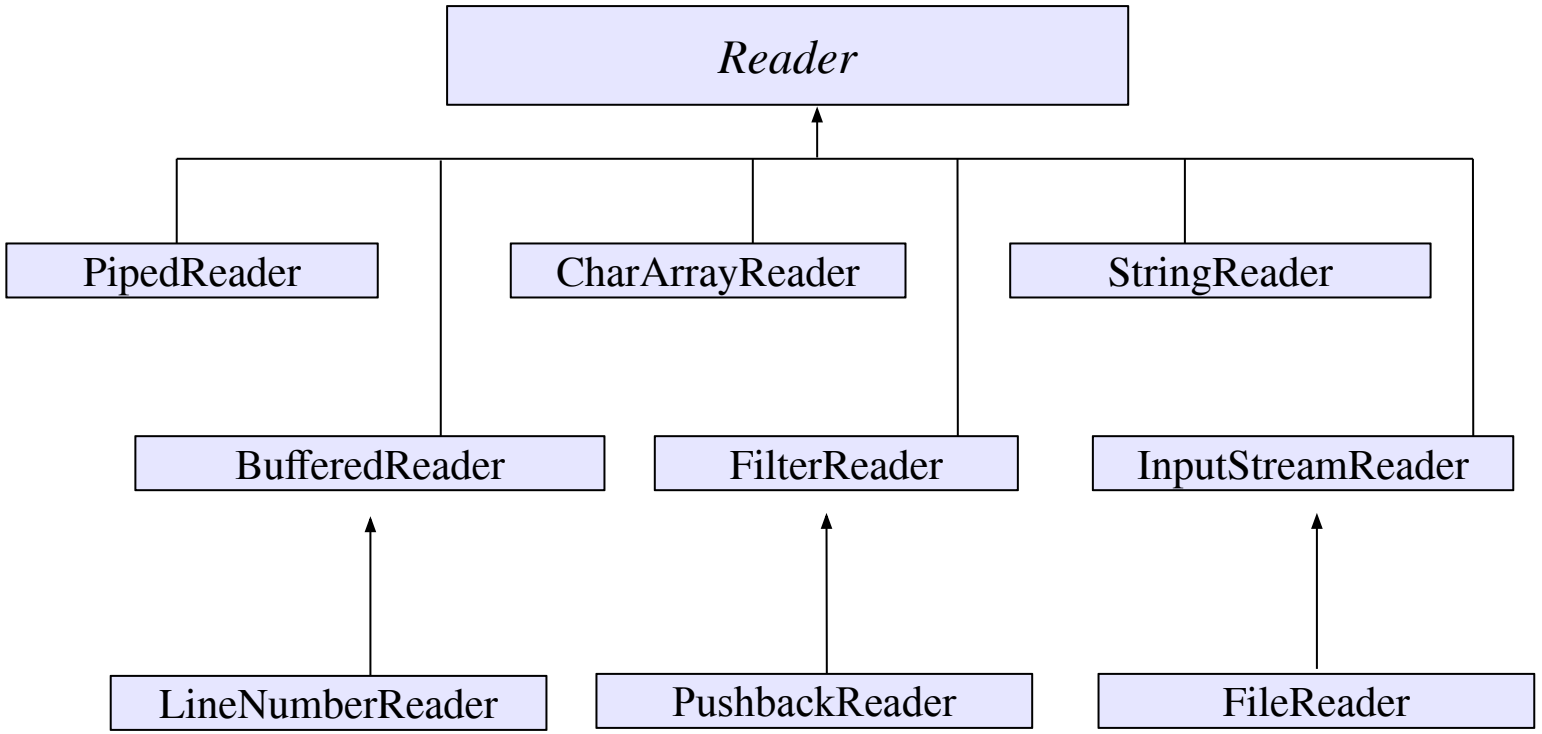
- Byte oriented streams are attractive to programmers familiar with C, C++ or who have UNIX experience
  - They are identical to what these programmers are used to
- Because they are byte-oriented, they are inflexible when dealing with multi-byte characters
  - Byte oriented streams only directly support ASCII
    - International fonts would require extra work for the programmer
- Character based streams
  - Abstract classes are Reader and Writer
  - Can be used in conjunction with byte-oriented streams
  - Useful when reading and writing text (character data)
  - Readers and Writers support a wide variety of character encodings including multi-byte encodings like Unicode.

# Character-Oriented Streams

- There are many different types of Character-Oriented Streams
  - Represented by different classes within the java.io.package
  - All character-oriented streams are subclasses of an abstract class
    - Writers are subclasses of the abstract class java.io.Writer
    - Readers are subclasses of the abstract class java.io.Reader
  - All character-oriented streams inherit basic methods from their respective superclasses
    - Some define new methods pertinent to the type of data they provide.
- Character oriented streams can be used in conjunction with byte-oriented streams:
  - Use InputStreamReader to "convert" an InputStream to a Reader
  - Use OutputStreamWriter to "convert" an OutputStream to a Writer

# Character-Oriented Reader Classes

- **The following is the byte-oriented input stream class hierarchy:**



# Reader Methods

- Reading
  - read() methods will block until data is available to be read
  - two of the three read() methods return the number of bytes read
    - -1 is returned if the Stream has ended
  - throws IOException if an I/O error occurs. This is a checked exception
- There are 3 main read methods:

```
int read()
```

  - Reads a single character.  Returns it as integer

```
int read(char[] buffer)
```

  - Reads bytes and places them into buffer (max = size of buffer)
  - returns the number of bytes read

```
int read(char[] buffer, int offset, int length)
```

  - Reads up to length bytes and places them into buffer
  - First byte read is stored in buffer[offset]
  - returns the number of bytes read

# Reader Methods

- close() method closes the stream
- mark(int readAheadLimit) marks the current location
  - Parameter specifies the number of characters which can be read before the marks becomes invalid
- ready() returns true if there is data to be read from the stream
  - returns true if the stream is guaranteed not to block upon next read.
- reset() returns the stream to its previously marked location
- skip(long n) skips over n bytes

# Creating a Reader Object

- Reader is abstract.  Programmers instantiate one of its subclasses.
- BufferedReader
  - Reads text from the character input stream
  - Provides buffering to provide efficient reading of characters, arrays and lines
- CharArrayReader
  - Similar to ByteArrayInputStream
  - Constructor takes a character array.  The character array provides the characters for the stream.
- FilterReader
  - An abstract class for filtering character streams
  - Filtering will be discussed later in the chapter

# Creating a Reader Object

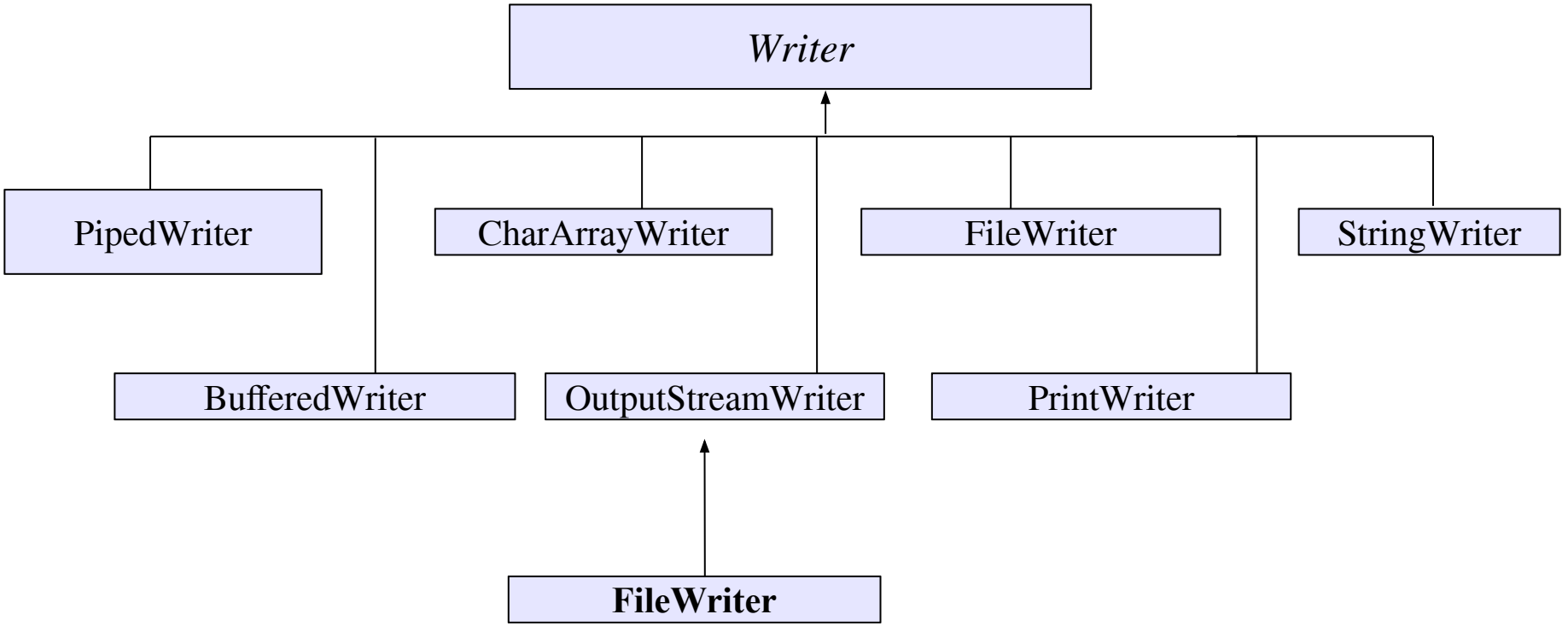
- InputStreamReader
  - This class acts as a bridge from byte streams to character streams
  - InputStreamReader takes an InputStream parameter to its constructor
  - The InputStreamReader reads bytes from the InputStream and translates them into characters according to the specified encoding.
- PipedReader
  - Similar to PipedInputStream
  - Connects to an Instance of PipedWriter
  - A pipe represents a one-way stream through which 2 threads may communicate
    - Thread1 writes to a PipedWriter
    - Thread2 reads from the PipedReader
- StringReader
  - Provides a character stream where the data is obtained from a String

# Creating a Reader Object

- LineNumberReader (subclass of BufferedReader)
  - A stream which keeps track of how many lines there have been
  - A line is terminated with a linefeed, carriage return or a carriage return followed immediately by a linefeed.
- PushbackReader (subclass of FilterReader)
  - A stream which allows characters to be pushed back into the stream after being read
  - The number of characters which can be pushed back is specified when instantiated.  Default = 1
- FileReader (subclass of InputStreamReader)
  - A convenience class to provide a character based stream from file.
  - Alternatively, open the file using a FileInputStream and then pass that stream to an InputStreamReader instance.

# Character-Oriented Writer Classes

- **The following is the byte-oriented input stream class hierarchy:**



# Writer Methods

- There are 5 main write methods:

`void write(int c)`
  - Writes a single character.

`void write(char[] buffer)`
  - Writes an array of characters

`void write(char[] buffer, int offset, int length)`
  - Writes a portion of an array of characters
  - First character written is starts at buffer[offset]
  - length indicates how many characters to write.

`void write(String aString)`
  - Writes aString to the stream

`void write(String aString, int offset, int length)`
  - Writes a portion of a String to the stream
  - First character written is starts at aString.charAt(offset)
  - length indicates how many characters to write.

# Creating a Writer Object

- Writer is abstract.  Programmers instantiate one of its subclasses.
- BufferedWriter
  - Writes text to the character stream
  - Provides buffering to provide efficient writing of characters, arrays and lines
- CharArrayWriter
  - Similar to ByteArrayOutputStream
  - Characters written to the stream are stored in a buffer.
  - The buffer can be retrieved by calling toCharArray() or toString()
- FilterWriter
  - An abstract class for writing filtered character streams
  - Filtering will be discussed later in the chapter

# Creating a Writer Object

- OutputStreamWriter
  - This class acts as a bridge from character streams to byte streams
  - OutputStreamWriter takes an OutputStream parameter to its constructor
  - Characters written to the OutputStreamWriter are translated to bytes (based on the encoding) and written to the underlying OuputStream.
- PipedWriter
  - Similar to PipedOutputStream
  - Connects to an Instance of PipedReader
  - A pipe represents a one-way stream through which 2 threads may communicate
    - Thread1 writes to a PipedWriter
    - Thread2 reads from the PipedReader
- StringWriter
  - Characters written to this stream are collected in a StringBuffer.
  - The StringBuffer can be used to construct a String.

# Creating a Writer Object

- PrintWriter
  - Provides print() and println() methods for standard output
  - both print() and println() are overloaded to take a variety of types
  - When println is used, the stream will output the appropriate sequence (either linefeed, carriage return or carriage return/linefeed) for the current platform
  - System.out and System.err are PrintWriters

- FileWriter (subclass of OutputStreamWriter)
  - A convenience class for writing characters to file
  - FileWriters assume that the default character encoding is acceptable
  - Alternatively, open the file using a FileOutputStream and then pass that stream to an OutputStreamWriter instance.

# Filter Streams

- What are filter streams?
  - Filter streams are similar to filters in Unix
  - The basic idea is that while the data is being read (or written) the data is modified by a filter or series of filters.
  - How the data is modified is depends on which filters are used.
  - Filters can be chained together.

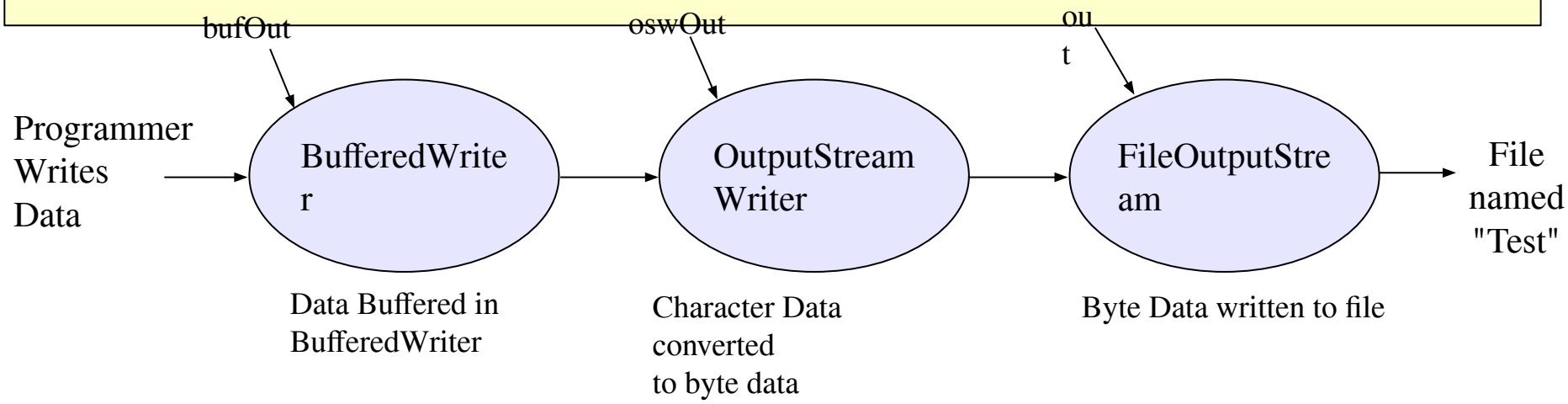
- Example:
  - A programmer creates a FileOuputStream
  - OutputStreams are byte-oriented, but the programmer wishes to use character-oriented streams instead.
    - The programmer knows that the OutputStreamWriter class can convert between character oriented streams and byte oriented streams
    - The programmer creates an OuputStreamWriter and passes the FileOutputStream reference to it
  - The programmer wishes to improve performance using a BufferedWriter.
    - The programmer creates a BufferedWriter and passes the OutputStreamWriter object to the constructor

# Filter Streams - Example

```
import java.io.*;

public class MyClass
{
   public void test()
   {
      try
      {
           FileOutputStream out = new FileOutputStream("Test");
           OutpuStreamWriter oswOut = new OutputStreamWriter(out);
           BufferedWriter bufOut = new BufferedWriter(oswOut);

           // programmer now uses bufOut
      }
      catch (IOException x)
      {
      }
   }
}
```



# FileWriter Revisited

- Remember FileWriter?
  - A convenience class for writing characters to file
  - FileWriters assume that the default character encoding and default buffer size are acceptable
  - Alternatively, open the file using a FileOutputStream and then pass that stream to an OutputStreamWriter instance.

- FileWriter is a filter class.
  - When it is created, it constructs a FileOutputStream, an OutputStreamWriter (with the default encoding) and a BufferedWriter with the default buffer size.
  - It is a considered a convenience class because it goes through the process of setting up the filter chain using default encoding and buffer sizes.
  - If the default values are not acceptable, the programmer will have to set up their own filters as outlined in the previous example.

# FilterStreams Provided with the JSDK

- Standard Byte-oriented Filter Streams:
  - ObjectInputStream, ObjectOutputStream
  - BufferedInputStream, BufferedOutputStream
  - DataInputStream, DataOutputStream
  - PushbackInputStream
- Compression filter Streams
  - GZIPInputStream, GZIPOutputStream
  - ZipInputStream, ZipOutputStream
  - InflatorInputStream, DeflatorOutputStream
- Character-oriented Filter Streams:
  - PushbackReader
  - FileWriter

# Object Serialization

- When an object is instantiated, the system reserves enough memory to hold all of the object's instance variables
  - The space includes inherited instance variables.
- The object exists in memory.
  - Instance methods read and update the memory for a given object.
- The memory which represents an object can be written to an ObjectOutputStream.
  - Objects are serialized to an ObjectOutputStream
- Any other objects referred to by the Serialized object are also serialized to the stream
  - Unless they are marked as "transient"
- When an object is serialized, the stream checks to ensure that the object implements the java.io.Serializable interface.
  - If not, the Stream throws a NotSerializableException
  - The Serializable interface does not define any methods.

# Example - Serialize an Object

```
import java.io.*;

public class Test
{
  public void saveObject(String outputFilename, Object anObject)
  {
    try
    {
        FileOutputStream fpout = new FileOutputStream(outputFilename);
        ObjectOutputStream obOut = new ObjectOutputStream(fpout);
        obOut.writeObject(anObject);
        obOut.flush();
        obOut.close();
    }
    catch (IOException x)
    {
        System.out.println("Error:" + x);
    }
  }
}
```

# Example - Read in a Serialized Object

```
import java.io.*;

public class Test
{
  public Object readObject(String inputFilename)
  {
     try
     {
          FileInputStream fpin = new FileInputStream(inputFilename);
          ObjectInputStream obIn = new ObjectInputStream(fpin);
          Object anObject = obIn.readObject();
          obIn.close();
          return anObject;
     }
     catch (IOException x)
     {
          System.out.println("Error:" + x);
     }
  }
}
```

# Example - Serialize an Object and Compress

```
import java.io.*;
import java.util.zip.*;

public class Test
{
  public void saveObject(String outputFilename, Object anObject)
  {
    try
    {
        FileOutputStream fpout = new FileOutputStream(outputFilename);
        DeflaterOutputStream dOut = new DeflaterOutputStream(fpout);
        ObjectOutputStream obOut = new ObjectOutputStream(dOut);
        obOut.writeObject(anObject);
        obOut.flush();
        obOut.close();
    }
    catch (IOException x)
    {
        System.out.println("Error:" + x);
    }
  }
}
```

# Example - Read in a Compressed Serialized Object

```
import java.io.*;

public class Test
{
  public Object readObject(String inputFilename)
  {
    try
    {
        FileInputStream fpin = new FileInputStream(inputFilename);
        InflaterInputStream inflateIn = new InflaterInputStream(fpin);
        ObjectInputStream obIn = new ObjectInputStream(inflateIn);
        Object anObject = obIn.readObject();
        obIn.close();
        return anObject;
    }
    catch (IOException x)
    {
        System.out.println("Error:" + x);
    }
  }
}
```

# The File Class

- Java IO provides a class which is an abstract representation of a file or directory within the file system.
- The File class has 2 constructors:

```
File(String pathName)
File(File parent, String child)
```

- The File class provides several query methods:
  - canRead(), canWrite(), exists(), getAbsolutePath(), getName(), getParent(), getPath(), isAbsolute(), isDirectory(), isHidden(), lastModified(), length(), and list()
- The File class also provides several methods which act on the file system:
  - createTempFile(), delete(), deleteOnExit(), mkdir(), mkdirs(), renameTo(), setLastModified(), setReadOnly()