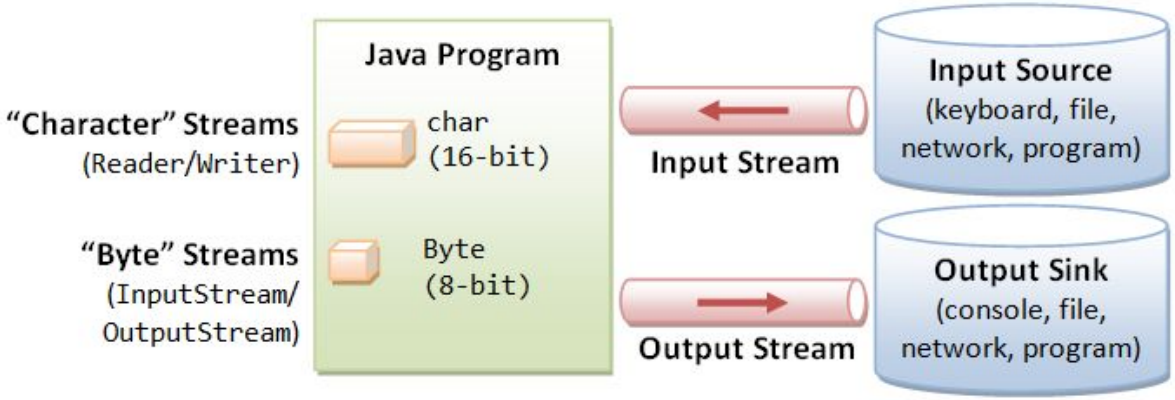
JAVA IO

Session II

Stream I/O

- Inputs and outputs are handled by the so-called streams
- A stream is a sequential and contiguous one-way flow of data
- The Java program receives data from a source by opening an input stream, and sends data to a sink by opening an output stream
- Two streams - an input stream and an output stream

Continued



Internal Data Formats:

- Text (char): UCS-2
- int, float, double, etc.

External Data Formats:

- Text in various encodings (US-ASCII, ISO-8859-1, UCS-2, UTF-8, UTF-16, UTF-16BE, UTF16-LE, etc.)
- Binary (raw bytes)

Byte Streams

- Byte streams are used to read/write raw bytes serially from/to an external device
- All the byte streams are derived from the abstract superclasses InputStream and OutputStream

Reading from an InputStream

```
public abstract int read() throws IOException
```

- returns the input byte read as an `int` in the range of 0 to 255, or
- returns -1 if "end of stream" condition is detected, or

```
public int read(byte[] bytes, int offset, int length) throws IOException// Read "length" number of bytes,

store in bytes array starting from offset of index.


public int read(byte[] bytes) throws IOException    // Same as read(bytes, 0, bytes.length)
```

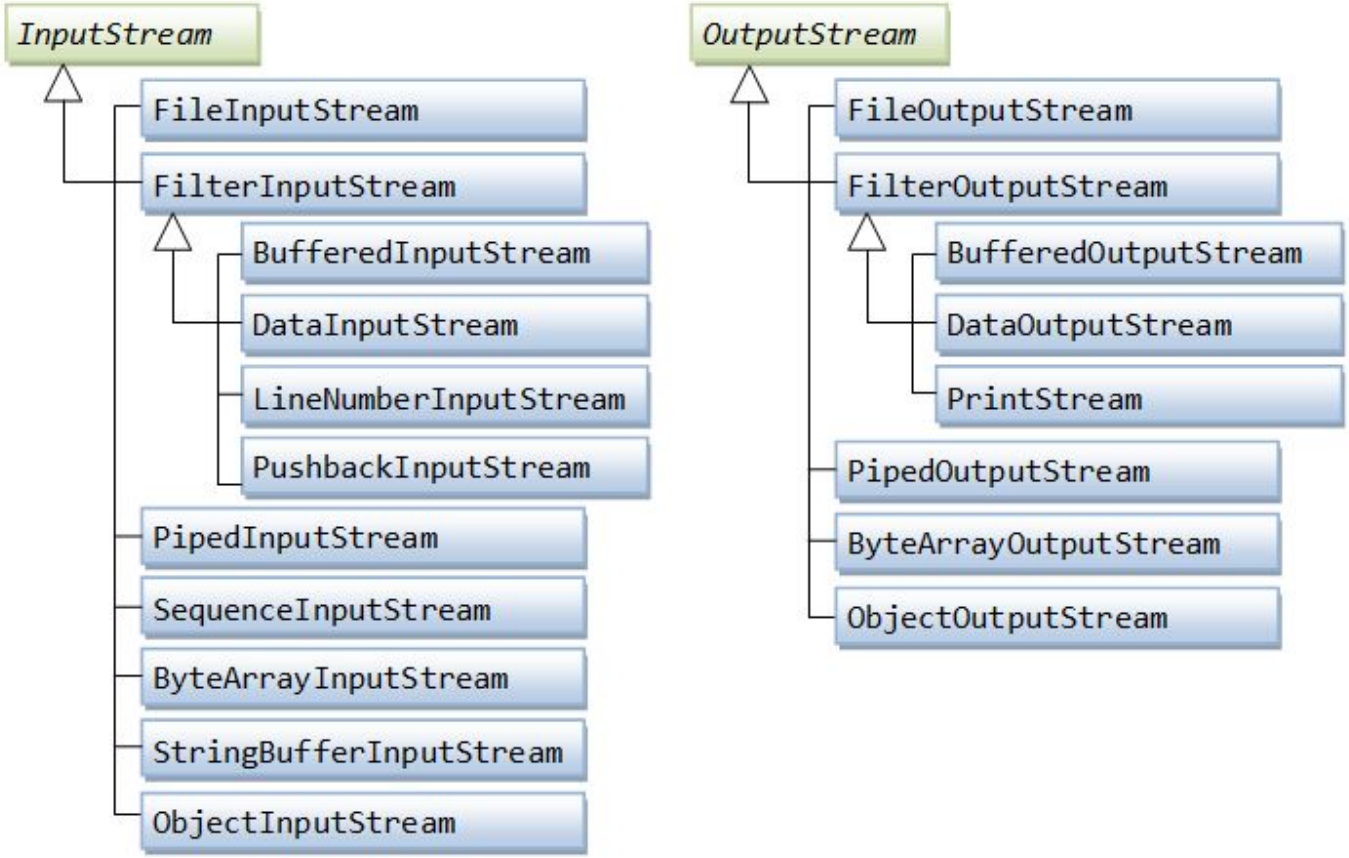
Writing to an OutputStream

```
public void abstract void write(int unsignedByte) throws IOException => to write a data-byte to the output sink


public void write(byte[] bytes, int offset, int length) throws IOException
// Write "length" number of bytes, from the bytes array starting from offset of index.
public void write(byte[] bytes) throws IOException
 // Same as write(bytes, 0, bytes.length)
```

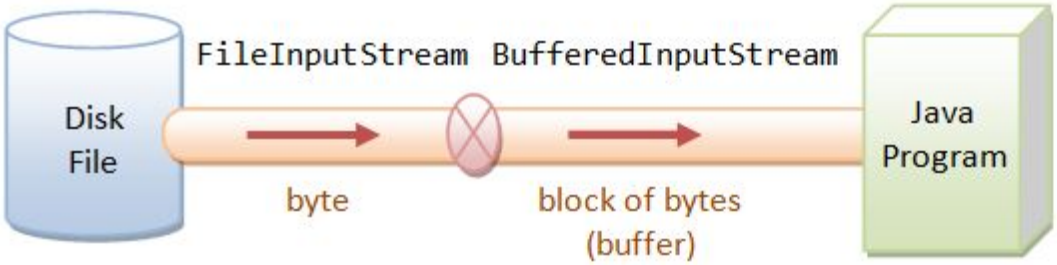
Opening & Closing I/O Streams

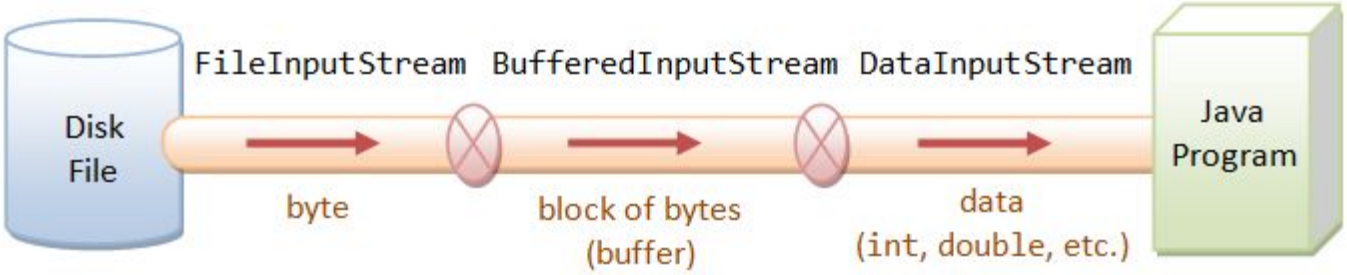
```
FileInputStream in = null;

try {

   in = new FileInputStream(...); // Open
stream

} catch (IOException ex) {
ex.printStackTrace();} finally { // always
close the I/O streams

   try {  if (in != null) in.close();

   } catch (IOException ex) {
ex.printStackTrace();   }

  }
```

```
try (FileInputStream in = new
FileInputStream(...)) {

   ......

   ......

} catch (IOException ex) {

   ex.printStackTrace();

  }     //  Automatically  closes  all  opened
  resource in try (...).
```


InputStream
FileInputStream
FilterInputStream
BufferedInputStream
DataInputStream
LineNumberInputStream
PushbackInputStream
PipedInputStream
SequenceInputStream
ByteArrayInputStream
StringBufferInputStream
ObjectInputStream
OutputStream
FileOutputStream
FilterOutputStream
BufferedOutputStream
DataOutputStream
PrintStream
PipedOutputStream
ByteArrayOutputStream
ObjectOutputStream

Chained I/O Streams





Continued

```
FileInputStream fileIn = new FileInputStream("in.dat");

BufferedInputStream bufferIn = new BufferedInputStream(fileIn);

DataInputStream dataIn = new DataInputStream(bufferIn);

// or

DataInputStream in = new DataInputStream(

                        new BufferedInputStream(

                            new FileInputStream("in.dat")));
```

JDK 1.7

```
try (FileInputStream  in  = new FileInputStream(inFileStr);
            FileOutputStream out = new FileOutputStream(outFileStr)) {
           int byteRead;
          // Read a raw byte, returns an int of 0 to 255.
          while ((byteRead = in.read()) != -1) {
             // Write the least-significant byte of int, drop the upper 3 bytes
             out.write(byteRead);
           }
                } catch (IOException ex) {
          ex.printStackTrace();
       }
```

Formatted Data-Streams: DataInputStream & DataOutputStream

```
DataInputStream in = new DataInputStream(new BufferedInputStream(new FileInputStream("in.dat")));

// 8 Primitives

public final int readInt() throws IOExcpetion;       // Read 4 bytes and convert into int

public final double readDoube() throws IOExcpetion;  // Read 8 bytes and convert into double

public final byte readByte() throws IOExcpetion;

public final char readChar() throws IOExcpetion;

public final short readShort() throws IOExcpetion;

public final long readLong() throws IOExcpetion;

public final boolean readBoolean() throws IOExcpetion;    // Read 1 byte. Convert to false if zero

public final float readFloat() throws IOExcpetion;
```

```
public final int readUnsignedByte() throws IOExcpetion;   // Read 1 byte in [0, 255] upcast to int

public final int readUnsignedShort() throws IOExcpetion;  // Read 2 bytes in [0, 65535], same as
char, upcast to int

public final void readFully(byte[] b, int off, int len) throws IOException;

public final void readFully(byte[] b) throws IOException;

 // Strings

public final String readLine() throws IOException;

     // Read a line (until newline), convert each byte into a char - no unicode support.

public final String readUTF() throws IOException;

    // read a UTF-encoded string with first two bytes indicating its UTF bytes length
```

Dataoutputstream

```
// 8 primitive types

public final void writeInt(int i) throws IOExcpetion;      // Write the int as 4 bytes

public final void writeFloat(float f) throws IOExcpetion;

public final void writeDoube(double d) throws IOExcpetion; // Write the double as 8 bytes

public final void writeByte(int b) throws IOExcpetion;     // least-significant byte

public final void writeShort(int s) throws IOExcpetion;    // two lower bytes

public final void writeLong(long l) throws IOExcpetion;

public final void writeBoolean(boolean b) throws IOExcpetion;

  public final void writeChar(int i) throws IOExcpetion;
```

```
// String
public final void writeBytes(String str) throws IOExcpetion;
    // least-significant byte of each char
public final void writeChars(String str) throws IOExcpetion;
    // Write String as UCS-2 16-bit char, Big-endian (big byte first)
public final void writeUTF(String str) throws IOException;
    // Write String as UTF, with first two bytes indicating UTF bytes length
public final void write(byte[] b, int off, int len) throws IOException
public final void write(byte[] b) throws IOException
  public final void write(int b) throws IOException     // Write the least-significant byte
```

Character-Based I/O & Character Streams

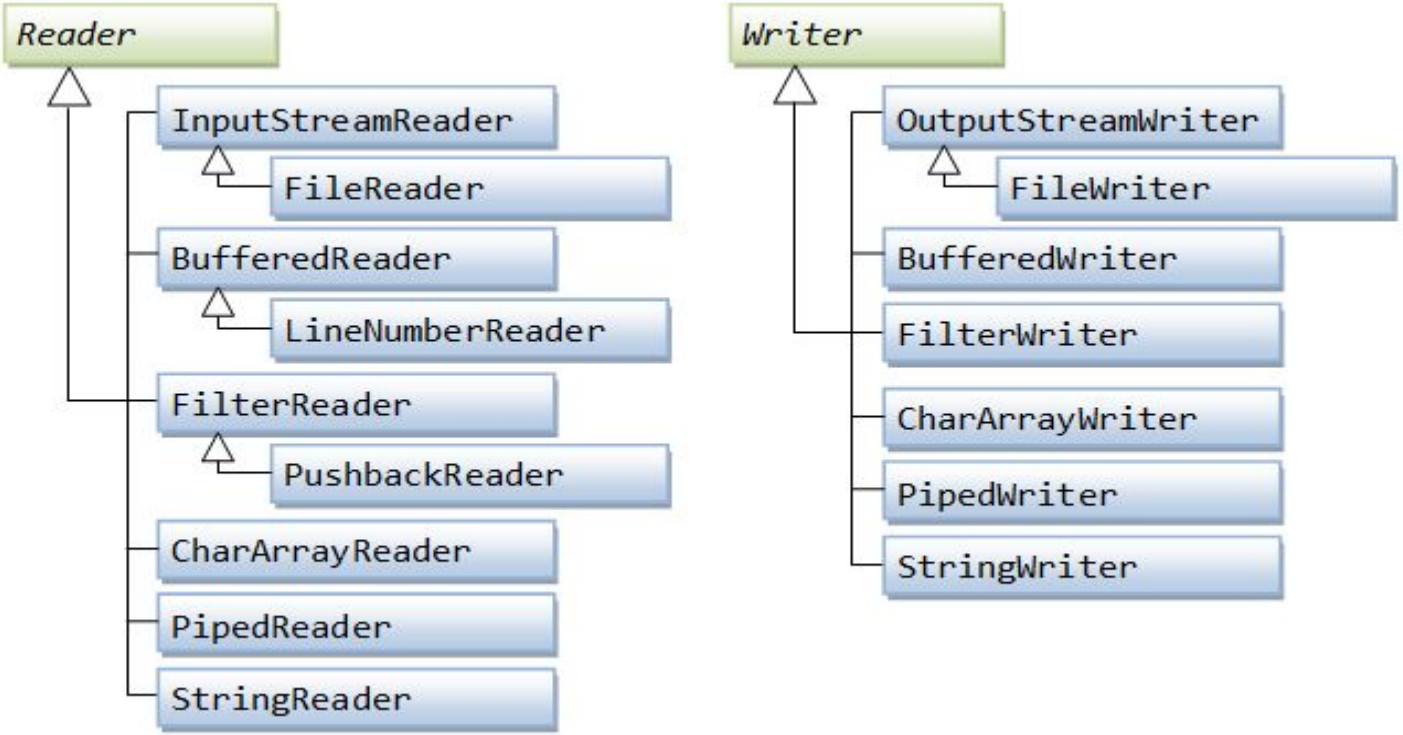
- Java internally stores characters (char type) in 16-bit UCS-2 character set
- Character-based I/O is almost identical to byte-based I/O. Instead of InputStream and OutputStream, we use Reader and Writer for character-based I/O

Reader

```
public abstract int read() throws IOException

public int read(char[] chars, int offset, int length) throws
IOException

public int read(char[] chars) throws IOException
```

Writer

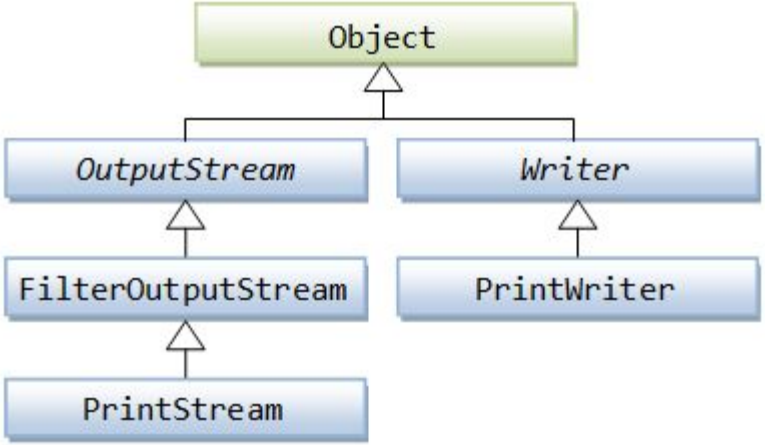
```
public void abstract void write(int aChar) throws IOException

public void write(char[] chars, int offset, int length) throws
IOException

public void write(char[] chars) throws IOException
```


Reader
InputStreamReader
FileReader
BufferedReader
LineNumberReader
FilterReader
PushbackReader
CharArrayReader
PipedReader
StringReader
Writer
OutputStreamWriter
FileWriter
BufferedWriter
FilterWriter
CharArrayWriter
PipedWriter
StringWriter

InputStreamReader and OutputStreamWriter

```
public InputStreamReader(InputStream in)    // Use default charset

public InputStreamReader(InputStream in, String charsetName) throws UnsupportedEncodingException

public InputStreamReader(InputStream in, Charset cs)
```

```
try (BufferedInputStream in = new BufferedInputStream( //
Buffered for efficiency
                 new FileInputStream(csStrs[i] +
outFileExt))) {


}
```



Object Serialization and Object Streams

- Object serialization is the process of representing a "particular state of an object" in a serialized bit-stream, so that the bit stream can be written out to an external device (such as a disk file or network)
- The bit-stream can later be re-constructed to recover the state of that object
- Object serialization is necessary to save a state of an object into a disk file for persistence or sent the object across the network for applications such as Web Services, Distributed-object applications, and Remote Method Invocation (RMI)
- java.io.Serializable or java.io.Externalizable interface

ObjectInputStream & ObjectOutputStream

```
public final Object readObject() throws IOException, ClassNotFoundException;

public final void writeObject(Object obj) throws IOException;
```

```
ObjectOutputStream out =
   new ObjectOutputStream(
      new BufferedOutputStream(
         new FileOutputStream("object.ser")));
out.writeObject("The current Date and Time is ");
// write a String object
out.writeObject(new Date());
// write a Date object
out.flush();
 out.close();
```

```
ObjectInputStream in =
   new ObjectInputStream(
      new BufferedInputStream(
         new FileInputStream("object.ser")));
String str = (String)in.readObject();
Date d = (Date)in.readObject(new Date()); //
downcast
 in.close();
```

Serialization

transient & static

- `static` fields are not serialized, as it belongs to the class instead of the particular instance to be serialized.
- To prevent certain fields from being serialized, mark them using the keyword `transient`. This could cut down the amount of data traffic.
- The `writeObject()` method writes out the class of the object, the class signature, and values of non-`static` and non-`transient` fields.

```
class MySerializedObject implements Serializable {
   private int number;

   public MySerializedObject(int number) {
      this.number = number;
   }

   public int getNumber() {
      return number;
   }
 }
```