Streams

Hyang-Won Lee
Department of Computer Science & Engineering
Konkuk University
leehw@konkuk.ac.kr
https://sites.google.com/view/leehwko/

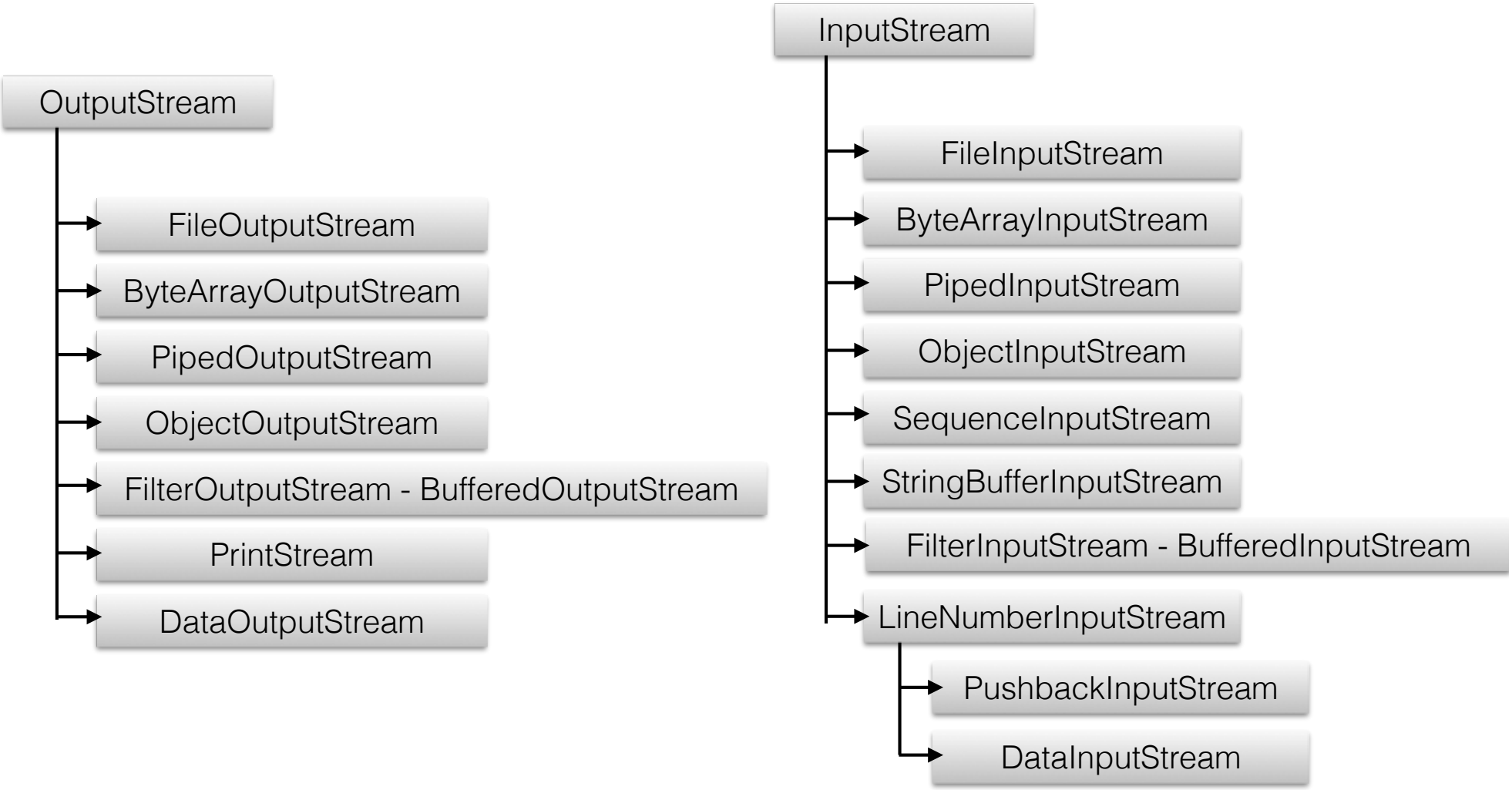
Network Programs

- What network programs do
  - move bytes from one host to another
  - sequence of input/output (I/O) operations
- I/O operations
  - write to or read from I/O devices
- I/O devices
  - files
  - keyboard, monitor, mouse
  - memory
  - network interface card (NIC)
- Sending data to a client ≈ writing data to a file
- Reading data a server sent ≈ reading data from a file

Streams

- I/O operations in Java are built on streams
- Stream
  - sequence of data (byte)
- Input streams read data
- Output streams write data
- Different streams classes read/write different sources of data
  - java.io.FileInputStream reads data from a file
  - java.io.FileOutputStream writes data to a file
- All output streams use the same methods to write data
- All input streams use the same methods to read data

Hierarchy



- Streams can be chained for encryption, compression, (format) conversion

Synchronous vs. Nonblocking

- Streams are synchronous
  - when a program asks a stream to read or write, it waits for data to be read or written before it does anything else (blocking)
- Nonblocking I/O
  - doesn't wait if data not ready to be read or written

Special I/O Streams

- public static PrintStream out //standard output, i.e., screen
  - System.out.println("message");
- public static InputStream in //standard input, i.e., keyboard
  - int d = System.in.read(); // read one byte from keyboard (triggered by return/enter key)
- public static PrintStream err //standard error output
  - System.err.println("error message");
- Standard I/O streams
  - most frequently used streams
  - static member in java.lang.System

Output Streams

OutputStream Class

- Basic output class

```
public abstract class OutputStream
```

- Methods to write data

```
public abstract void write(int b) throws IOException
public void write(byte[] data) throws IOException
public void write(byte[] data, int offset, int length)
    throws IOException
public void flush() throws IOException
public void close() throws IOException
```

- Subclasses use these methods
  - FileOutputStream use these methods to write data to a file
  - ByteArrayOutputStream these methods to write data to byte array

can instantiate like “OutputStream out = new OutputStream();”??

CoIN LAB

write(int b)

- Take an integer and write a single unsigned byte in [0,255] to output stream
  - if b not in [0,255], (b mod 256) is written ≡ least significant byte is written
- Declared abstract
  - subclasses implement this method to handle particular media
- Example

```
public class OutputStreamWriteTest {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        for (int i = 0; i<200; i++) {
            System.out.write(i);
            System.out.print(i);
            System.out.flush();
        }
    }
}
```

- read a least significant byte
- print ASCII of the byte value

Example

enerateCharactersSingleByte.java | GenerateCharactersHomeworkSpeedComparison.java | *OutputStreamWriteTest.java

```java
import java.io.IOException;
import java.io.OutputStream;

public class GenerateCharactersSingleByte {
    public static void main(String[] args) {
        try {
            generateCharacters(System.out);
        } catch (IOException ex) {
        }
    }

    public static void generateCharacters(OutputStream out) throws IOException {
        int firstPrintableCharacter = 33; //printable ASCII characters start from 33
        int numberOfPrintableCharacters = 94;
        int numberOfCharactersPerLine = 72;
        int start = firstPrintableCharacter;
        int count = 0;

        while (count < 1000) {
            for (int i = start; i < start + numberOfCharactersPerLine; i++) {
                out.write((byte) ((i - firstPrintableCharacter) % numberOfPrintableCharacters + firstPrintableCharacter));
            }

            out.write((byte) '\r'); //carriage return
            out.write((byte) '\n'); //line feed

            start = ((start+1) - firstPrintableCharacter) % numberOfPrintableCharacters + firstPrintableCharacter;

            count++;
        }
    }
}
```

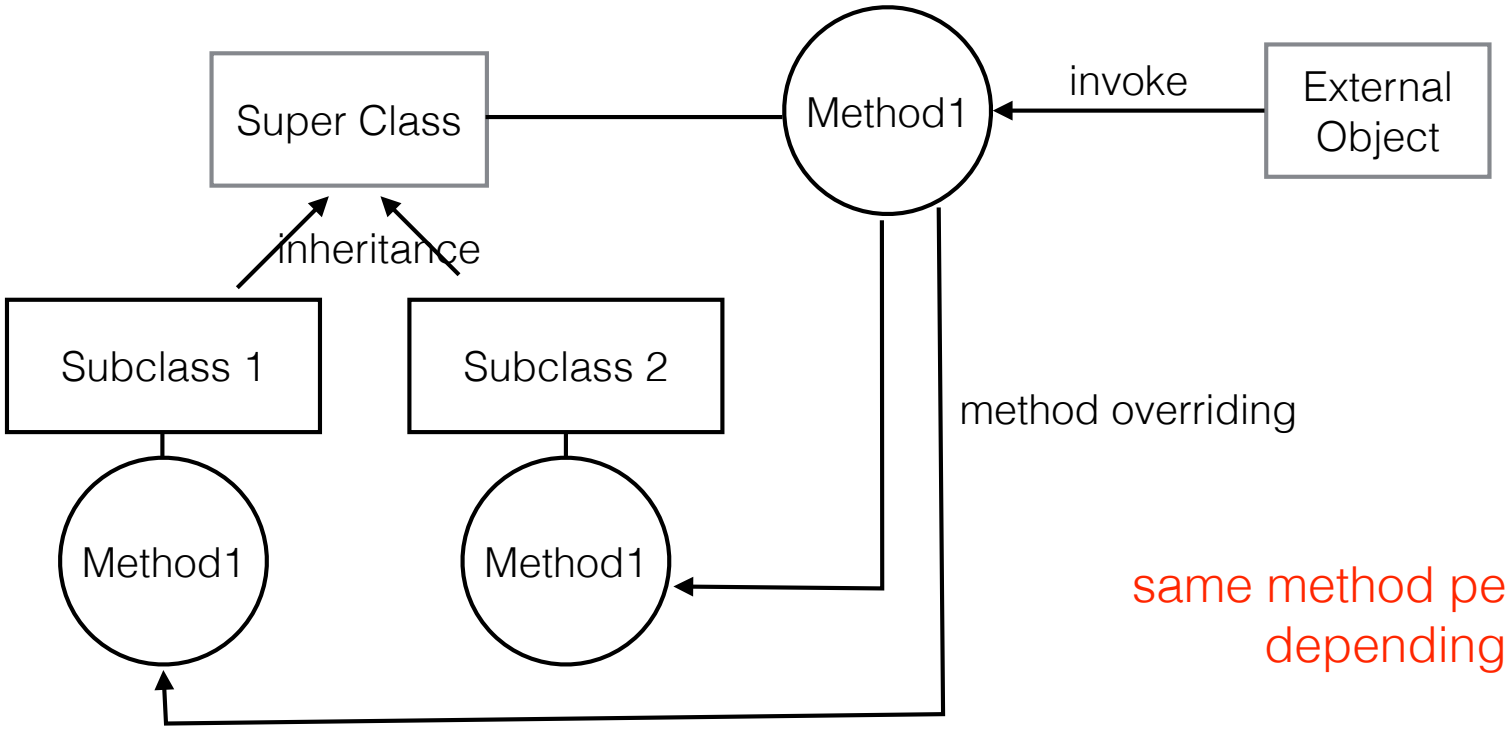
polymorphism

Problems | Javadoc | Declaration | Console

```
<terminated> GenerateCharactersSingleByte [Java Application] /Library/Java/JavaVirtualMachines/jd
YZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~!"#$%&'()*+,-./0123456789:;<=>?@AB
Z[\]^_`abcdefghijklmnopqrstuvwxyz{|}~!"#$%&'()*+,-./0123456789:;<=>?@ABC
[\]^_`abcdefghijklmnopqrstuvwxyz{|}~!"#$%&'()*+,-./0123456789:;<=>?@ABCD
\]^_`abcdefghijklmnopqrstuvwxyz{|}~!"#$%&'()*+,-./0123456789:;<=>?@ABCDE
```

Polymorphism

- Definition
  - (dictionary) several different forms
  - (Java) ability of an object to take on many forms



same method performs different tasks depending on subclasses

Example

```java
public class InheritanceTest {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        FirstChild fc = new FirstChild();
        System.out.println(fc.read());

        SecondChild sc = new SecondChild();
        System.out.println(sc.read());

        ThirdChild tc1 = new ThirdChild(fc);
        System.out.println(tc1.read());

        ThirdChild tc2 = new ThirdChild(sc);
        System.out.println(tc2.read());
    }
}

class Parent {
    public String read() {
        return "Parent 입니다.";
    }
}
```

```java
class FirstChild extends Parent {
    public String read() {
        return super.read()+": firstChild";
    }
}

class SecondChild extends Parent {
    public String read() {
        return super.read()+": secondChild";
    }
}

class ThirdChild extends Parent {
    Parent p;

    public ThirdChild(Parent p) {
        this.p = p;
    }

    public String read() {
        return p.read() + ": thirdChild";
    }
}
```

Problems | @ Javadoc | Declaration | Console

```
<terminated> InheritanceTest [Java Application] /Library/Java/J
Parent 입니다.: firstChild
Parent 입니다.: secondChild
Parent 입니다.: firstChild: thirdChild
Parent 입니다.: secondChild: thirdChild
```

tc1

p=FC

FC.read() invoked

tc2

p=SC

SC.read() invoked

Example

```java
class Parent2 {
    int i = 7;
    public int get() {
        return i;
    }
}

class Child2 extends Parent2 {
    int i = 5;
    public int get() {
        return i;
    }
}

public class ChildTest {
    public static void print(Parent2 p) {
        System.out.println(p.i);
        System.out.println(p.get());
    }


    public static void main(String[] args) {
        // TODO Auto-generated method stub
        Parent2 p = new Parent2();
        System.out.println("--------- 1 ---------");
        System.out.println(p.i);
        System.out.println(p.get());

        Child2 c = new Child2();
        System.out.println("--------- 2 ---------");
        System.out.println(c.i);
        System.out.println(c.get());

        Parent2 p2 = new Child2();
        System.out.println("--------- 3 ---------");
        System.out.println(p2.i);
        System.out.println(p2.get());

        System.out.println("--------- 4 ---------");
        print(c);
        print(p2);

    }
}
```

C2 instantiated and referenced by P2

Problems | @ Javadoc | Declaration | Console

```
<terminated> ChildTest [Java Application] /Library/Java/JavaVi
--------- 1 ---------
7
7
--------- 2 ---------
5
5
--------- 3 ---------
7
5
--------- 4 ---------
7
5
7
5
```

field in superclass
method in subclass

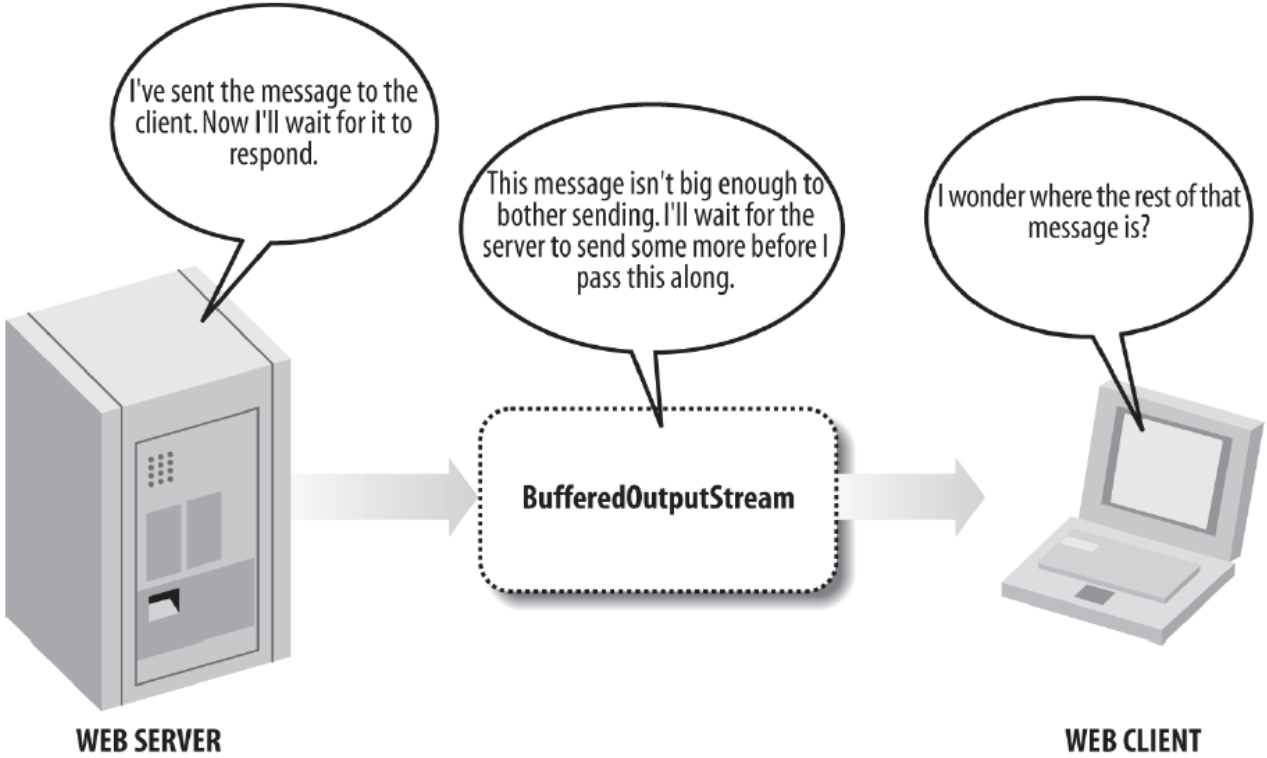
polymorphism

- only fields in superclass can be used
- method overridden in subclass can be used

write(byte[] data)

- Motivation
  - consider sending each byte of data separately using TCP
  - header of TCP/IP: at least 40byte
  - overhead: 4000% (40byte of overhead + 1byte of data)
  - solution: accumulate bytes in memory and send when a certain number of bytes have been accumulated
- Methods
  - write(byte[] data) write data to output stream
  - write(byte[] data, int offset, int length) write data[offset,…,offset+length-1] to output stream
  - much faster than writing one byte at a time

flush()



- flush() should be called to write data if not enough data (e.g., 300byte in 1024-byte buffer) have been received but want to send

close()

- Allow OS to release any resources associated to stream
- example

```
try {
  OutputStream out = new FileOutputStream("numbers.dat");
  // Write to the stream...
  out.close( );
}
catch (IOException ex) {
  System.err.println(ex);
}
```

what is the potential risk of this code?

Two Alternatives

```java
OutputStream out = null;
try {
  out = new FileOutputStream("/tmp/data.txt");
  // work with the output stream...
} catch (IOException ex) {
  System.err.println(ex.getMessage());
} finally {
  if (out != null) {
    try {
      out.close();
    } catch (IOException ex) {
      // ignore
    }
  }
}
```

```java
try (OutputStream out = new FileOutputStream("/tmp/data.txt")) {
  // work with the output stream...
} catch (IOException ex) {
  System.err.println(ex.getMessage());
}
```

Java automatically invokes close() on any AutoCloseable objects declared inside the argument list of try block

In-Class Lab

- Type and run examples
  - OutputStreamWriteTest
  - GenerateCharacterSingleByte
  - InheritanceTest
  - ChildTest
- Exercise
  - ??

Exercise

- Write the following code that
  - does the same as GenerateCharacterSingleByte
  - uses one of the write() methods to write one line at a time (not a byte at a time)
  - class name: GenerateCharacterByteArray
- Homework
  - Compare the speed of GenerateCharacterSingleByte and GenerateCharacterByteArray

Input Streams

InputStream Class

- Basic input class

```
public abstract class InputStream
```

- Methods to read data

```
public abstract int read() throws IOException
public int read(byte[] input) throws IOException
public int read(byte[] input, int offset, int length) throws IOException
public long skip(long n) throws IOException
public int available() throws IOException
public void close() throws IOException
```

- Subclasses use these methods to read

Polymorphism

- readSomething(InputStream in) vs. readSomething(FileInputStream in)

read()

- Read a single byte from input stream's source and returns it as an int from 0 to 255
  - int returned to indicate EOF
- End of stream is signified by returning -1
- Wait and block execution of any code that follows, until a byte of data is available and ready to be read
- Declared abstract
  - subclasses must implement this method to handle their particular medium

Example

```java
int inChar=0;
int inChar2=0;
System.out.println("Enter a Character:");
try {
     this is the original
    int i = 0;
    inChar = System.in.read();
    System.out.write(inChar);
```

Problems @ Javadoc Declaration Console

```
<terminated> SystemInReadTest [Java Application] /Library/Jav
Enter a Character:
abc
a
```

read(byte[] input)

- Two methods

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

attempt to read len bytes of data into b (b[off,…,off+len-1])

- read(byte[] input)
  - attempt to fill input and return the number of bytes read

```
byte[] input  = new byte[1024];
int bytesRead = in.read(input);
```

attempt to read 1024 bytes from InputStream in into input

  - block until input data available, EOF reached, or exception thrown
  - return -1 if no byte available

Example

- Read predetermined number of bytes

```
int bytesRead   = 0;
int bytesToRead = 1024;
byte[] input    = new byte[bytesToRead];
while (bytesRead < bytesToRead) {
  bytesRead += in.read(input, bytesRead, bytesToRead - bytesRead);
}
```

  - What is wrong with this code?

Example

```java
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;

public class SystemInReadOutWriteTest {

    public static void main(String[] args) {
        InputStream in = System.in;
        OutputStream out = System.out;

        try {
            int input = in.read();
            System.out.println(input);
            out.write(input);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

- Type 17 and press enter
- Anything wrong with the above code?

available()

- Return how many bytes can be read without blocking
- Example

```
try {
  byte[] b = new byte[System.in.available( )];
  System.in.read(b);
}
catch (IOException ex) {
  System.err.println("Couldn't read from System.in!");
}
```

skip(long n)

- Skip n bytes
- Return the number of bytes actually skipped
  - -1 if end of stream is encountered
- Often faster than reading and discarding bytes
  - example
    - input stream attached to a file
    - skipping changes the position in the file
    - reading & discarding copy bytes from disk into memory
- example

```
long bytesSkipped = 0;
long bytesToSkip = 80;
while (bytesSkipped < bytesToSkip) {
  long n = in.skip(bytesToSkip - bytesSkipped);
  if (n == -1) break;
  bytesSkipped += n;
}
```

Filter Classes

Filter Classes

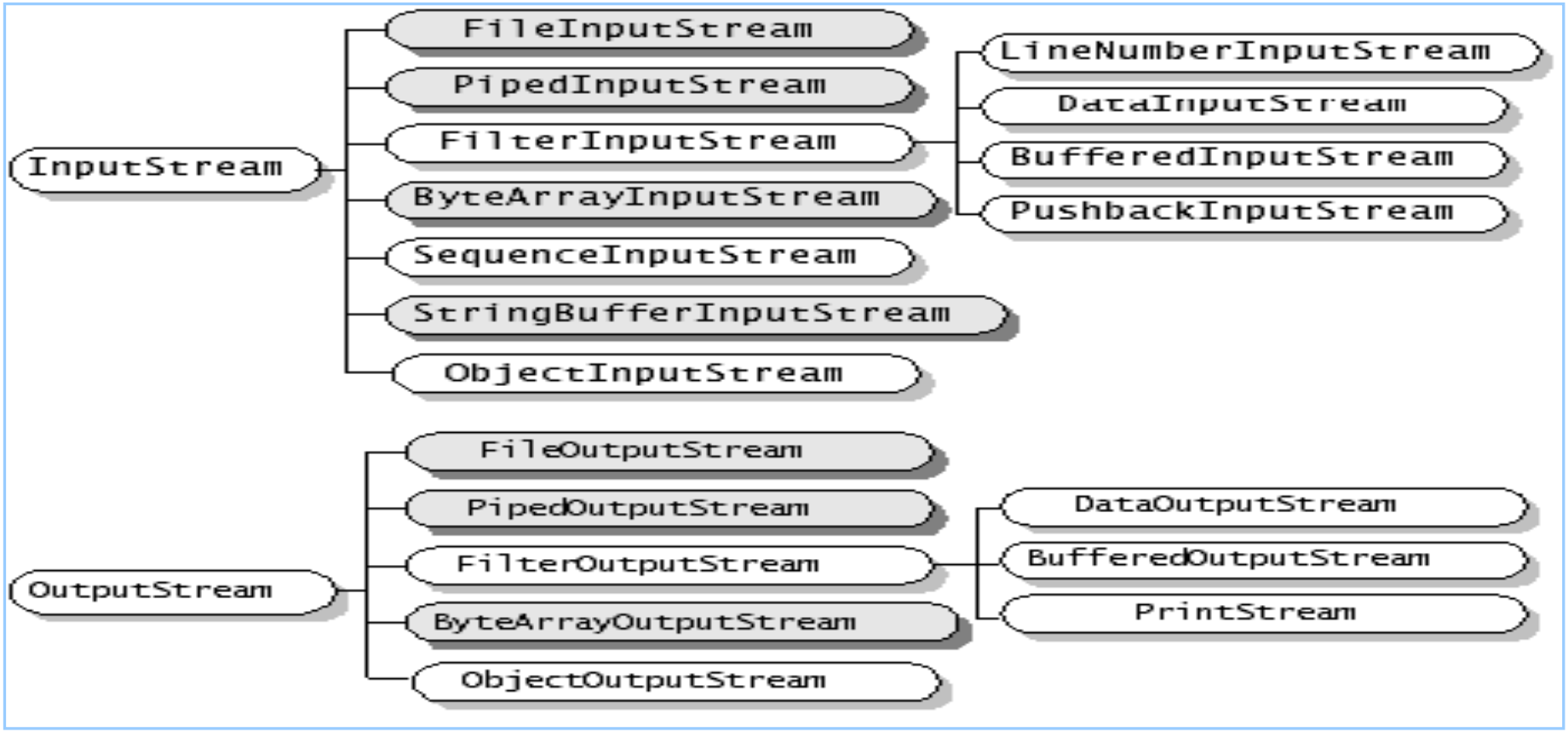
- InputStream and OutputStream are fairly raw classes
  - read and write bytes oblivious of data types
- Network protocols use different types of data
  - HTTP header: 7-bit ASCII
  - FTP: ZIP format
  - with only InputStream and OutputStream, developers need to write a routine to interpret and convert formats
- Java provides a number of filter classes to attach to raw streams to translate raw bytes to and from other formats
  - three kinds of filter classes: byte streams, readers and writers

character streams

bytes

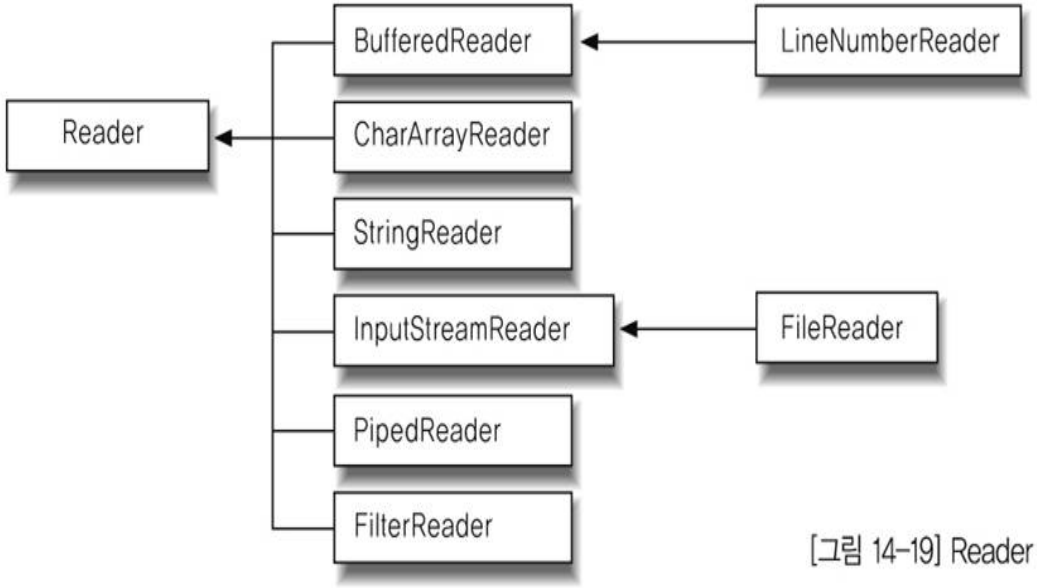
handle text in a variety of encodings such as UTF-8, ISO 8859-1

Byte Streams

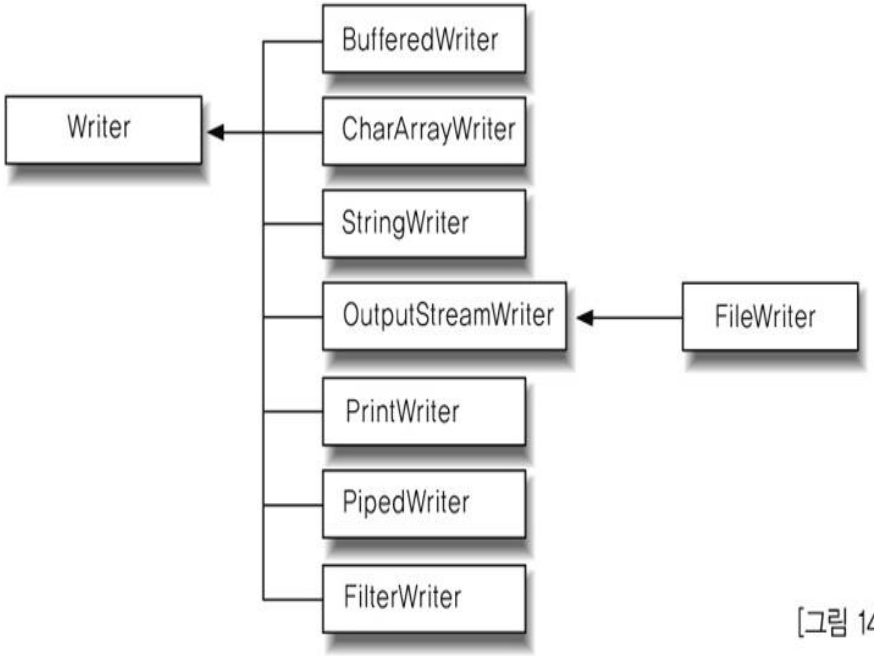


◉

Character Streams

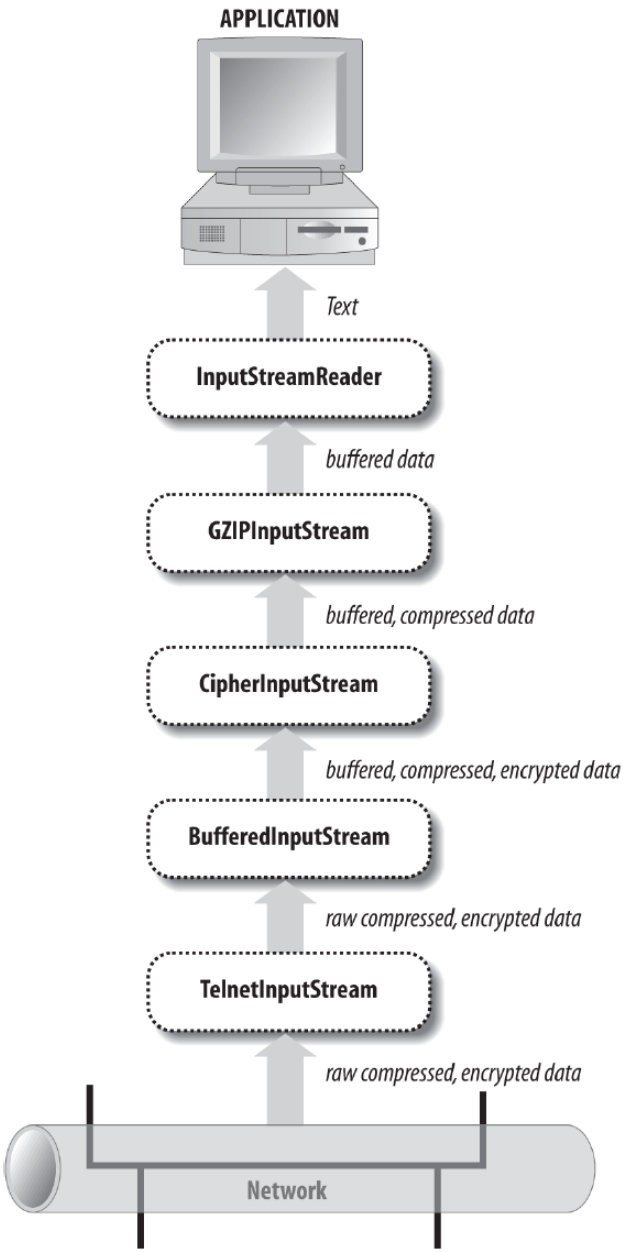


[그림 14-19] Reader 클래스의 상속도



[그림 14-18] Writer 클래스의 상속도

Chaining Filters



Filters are chained

File Streams

File Streams

- FileInputStream and FileOutputStream
  - read bytes from data in a file, and write bytes of data in a file
- Constructors

```
public FileInputStream(String fileName) throws IOException
public FileInputStream(File file) throws FileNotFoundException
public FileInputStream(FileDescriptor fdObj)
```

```
public FileOutputStream(String filename) throws IOException
public FileOutputStream(File file) throws IOException
public FileOutputStream(FileDescriptor fd)
```

Example

```java
import java.io.*;

public class FileView {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        long start = System.currentTimeMillis();

        if (args.length != 1) {
            System.out.println("사용법: java FileView 파일이름");
            System.exit(0);
        }

        FileInputStream fis = null;

        try {
            fis = new FileInputStream(args[0]);
            int i = 0;
            while((i=fis.read()) != -1) {
                System.out.write(i);
            }
        } catch (Exception ex) {
            System.out.println(ex);
        } finally {
            try {
                fis.close();
            } catch (IOException e) {

            }
        }
        long end = System.currentTimeMillis();
        System.out.println("Run-time: " + (end-start));

    }

}
```

Improvement

- reading 1 byte at a time
  - OS reads adjacent 256 or 512 bytes
  - read 1000bytes => read 512 bytes 1000 times
- read 512 bytes at a time
  - read 1000bytes => read 512 bytes twice

Example

```java
import java.io.*;

public class FileStreamCopy {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        if (args.length != 2) {
            System.out.println("사용법: java FileView 파일이름1 파일이름2");
            System.exit(0);
        }

        FileInputStream fis = null;
        FileOutputStream fos = null;

        try {
            fis = new FileInputStream(args[0]);
            fos = new FileOutputStream(args[1]);
            int readcount = 0;
            byte[] buffer = new byte[512];

            while((readcount=fis.read(buffer)) != -1) {
                //System.out.write(i);
                fos.write(buffer,0,readcount);
            }
            System.out.println("복사가 완료되었습니다");
        } catch (Exception ex) {
            System.out.println(ex);
        } finally {
            try {
                fis.close();
            } catch (IOException e) {}
            try {
                fos.close();
            } catch (IOException e) {}
        }
    }
```

In-Class Lab

- Type and run examples
  - SystemInReadTest
  - SystemInReadOutWriteTest
  - FileView
  - FileStreamCopy
- Exercise
  - ??

Exercise

- Find and fix what is wrong with 'bytesRead' example
- Improve FileView example

Buffered Streams

Buffered Streams

- BufferedOutputStream and BufferedInputStream
  - use buffer before reads and writes
- Constructors

polymorphism

underlying stream (children of InputStream or OutputStream accepted)

```
public BufferedInputStream(InputStream in)
public BufferedInputStream(InputStream in, int bufferSize)
public BufferedOutputStream(OutputStream out)
public BufferedOutputStream(OutputStream out, int bufferSize)
```

  - bufferSize: buffer size
    - default input buffer size: 2048bytes
    - default output buffer size: 512bytes

BufferedOutputStream

- Store written data in a buffer (protected byte array field named buf) until the buffer is full or stream is flushed
- Write the data onto the underlying output stream all at once
- Useful for network connections
  - TCP/IP header: 40bytes
  - sending one byte at a time
    - send 1kbytes ⇒ 41kbytes will be sent
  - sending all at once
    - send 1kbytes ⇒ 1.04kbytes will be sent

BufferedInputStream

- Have a buffer (protected byte array field named buf)
- read() called ⇒ try to get requested data from buf
  - buf runs out of data ⇒ read from underlying source
    - read as much data as it can from source into buffer
- T(reading several hundreds of bytes) ~ T(reading a byte)
  - buffering can substantially improve performance

Example

- File copy using buffered stream

```
try {
    fis = new FileInputStream(args[0]);
    fos = new FileOutputStream(args[1]);

    BufferedInputStream bis = new BufferedInputStream(fis);
    BufferedOutputStream bos = new BufferedOutputStream(fos);

    int readcount = 0;
    byte[] buffer = new byte[512];

    while((readcount=bis.read(buffer)) != -1) {
        //System.out.write(i);
        bos.write(buffer,0,readcount);
    }
    System.out.println("복사가 완료되었습니다");
    bis.close();
    bos.close();
```

Data Streams

Data Streams

- DataInputStream and DataOutputStream
  - provide methods for reading and writing Java's data types (int, float, double, boolean, short, byte) and strings in a binary format
- Constructors
  -
```
public DataInputStream(InputStream in)
public DataOutputStream(OutputStream out)
```

DataOutputStream

- Methods (usual write(), flush(), close() omitted)

```
public final void writeBoolean(boolean b) throws IOException
public final void writeByte(int b) throws IOException
public final void writeShort(int s) throws IOException
public final void writeChar(int c) throws IOException
public final void writeInt(int i) throws IOException
public final void writeLong(long l) throws IOException
public final void writeFloat(float f) throws IOException
public final void writeDouble(double d) throws IOException
public final void writeChars(String s) throws IOException
public final void writeBytes(String s) throws IOException
public final void writeUTF(String s) throws IOException
```

1byte of 0 or 1
write low-order 1byte
write low-order 2byte
write low-order 2byte
4byte
8byte
IEEE 754 form 4byte
IEEE 754 form 8byte
2byte char seq.
2byte seq. (s[0]s[1]..)
write s in UTF-8 enc.

  - all data is written big-endian format (most significant byte in the lowest address)
  - chars are written as two unsigned bytes

DataInputStream

- Methods (usual read(), available(), skip(), close() omitted)

```
public final boolean readBoolean() throws IOException
public final byte readByte() throws IOException
public final char readChar() throws IOException
public final short readShort() throws IOException
public final int readInt() throws IOException
public final long readLong() throws IOException
public final float readFloat() throws IOException
public final double readDouble() throws IOException
public final String readUTF() throws IOException
```

  - read requested number of bytes repeatedly

```
public final void readFully(byte[] input) throws IOException
public final void readFully(byte[] input, int offset, int length)
    throws IOException
```

    - can be useful for reading, for example, HTTP header (know how many bytes to read in advance)

Example

```java
import java.io.*;

public class DataOutputStreamTest {

    public static void main(String[] args) throws IOException {
        // TODO Auto-generated method stub
        FileOutputStream fos = null;
        DataOutputStream dos = null;

        boolean addTab = false;

        fos = new FileOutputStream("data.bin");
//      fos = new FileOutputStream(FileDescriptor.out);
        dos = new DataOutputStream(fos);
        dos.writeBoolean(false);
        if (addTab) dos.writeChar('\n');
        //System.out.println((byte)1);
        dos.writeByte((byte)125);
        if (addTab) dos.writeChar('\n');
        //System.out.println((byte)5);
        dos.writeInt(10);
        if (addTab) dos.writeChar('\n');
        //System.out.println((byte)1000000);
        dos.writeDouble(200.5);
        if (addTab) dos.writeChar('\n');

        dos.writeUTF("hello world");

        System.out.println("저장하였습니다");

        fos.close();
        dos.close();
```

Example

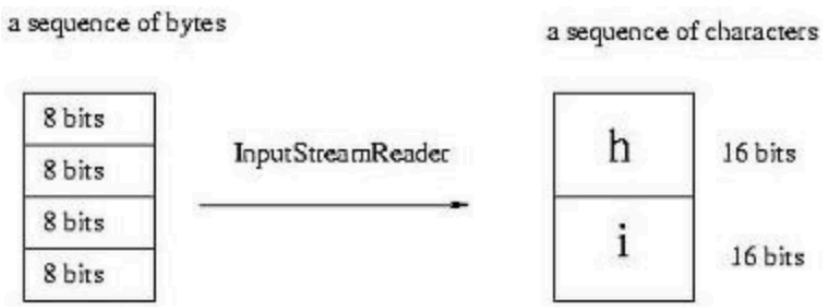
```
        FileInputStream fis = null;
        DataInputStream dis = null;

        fis = new FileInputStream("data.bin");
        dis = new DataInputStream(fis);

//      readcount = fis.read(buffer);
//      System.out.println(buffer);

        boolean boolVar = dis.readBoolean();
        if (addTab) dis.readChar();
        byte byteVar = dis.readByte();
        if (addTab) dis.readChar();
        int intVar = dis.readInt();
        if (addTab) dis.readChar();
        double doubleVar = dis.readDouble();
        if (addTab) dis.readChar();
        String stringVar = dis.readUTF();

        System.out.println(boolVar);
        System.out.println(byteVar);
        System.out.println(intVar);
        System.out.println(doubleVar);
        System.out.println(stringVar);

        fis.close();
        dis.close();


    }

}
```

Character Streams

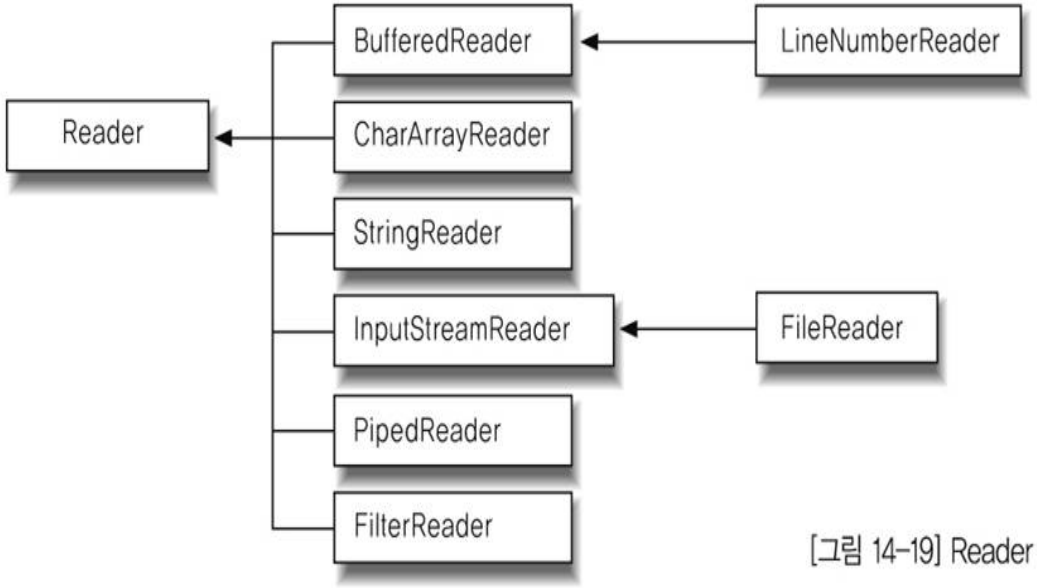
Readers and Writers

Motivation

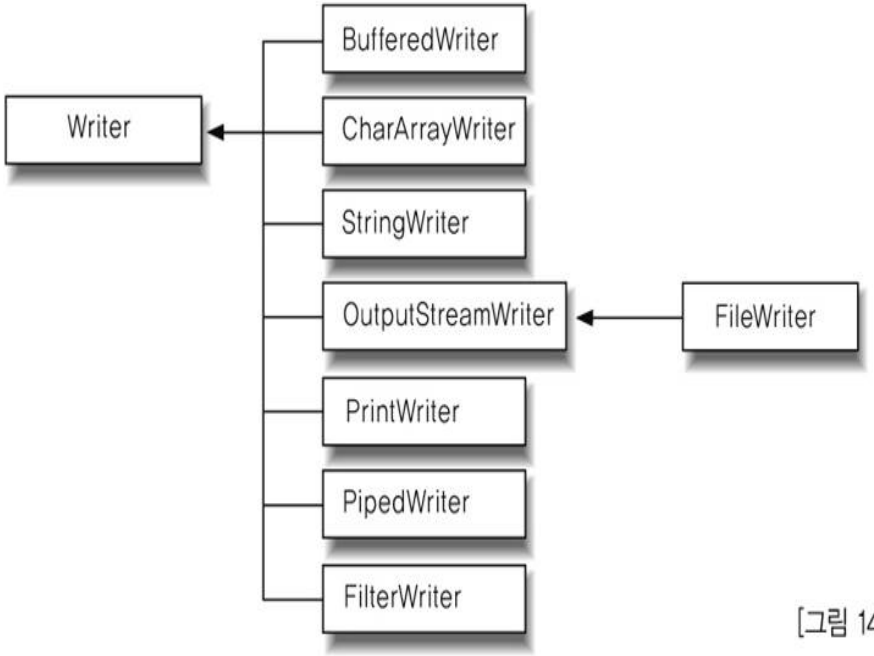
- InputStream and OutputStream read and write data as primitive bytes
- Different OSs can have different encodings
  - file written in one OS using byte stream can be unreadable in another OS if the two OSs have different encodings
- Character streams: Reader and Writer
  - Java platform stores characters using Unicode (internal format)
  - Reader/Writer automatically translate the internal format to and from local character set



Character Streams Hierarchy



[그림 14-19] Reader 클래스의 상속도



[그림 14-18] Writer 클래스의 상속도

Writers

- Mirror of OutputStream
  - declared abstract (never used directly, but used polymorphically)
- Methods

```
protected Writer()
protected Writer(Object lock)
public abstract void write(char[] text, int offset, int length)
    throws IOException
public void write(int c) throws IOException
public void write(char[] text) throws IOException
public void write(String s) throws IOException
public void write(String s, int offset, int length) throws IOException
public abstract void flush() throws IOException
public abstract void close() throws IOException
```

the other four write() use this method

Examples

- Write a string

```
char[] network = {'N', 'e', 't', 'w', 'o', 'r', 'k'};
w.write(network, 0, network.length);
```

- Four other ways to do the same write

```
w.write(network);
for (int i = 0;  i < network.length;  i++) w.write(network[i]);
w.write("Network");
w.write("Network", 0, 7);
```

-

OutputStreamWriter

- Most important concrete subclass of Writer
  - receive characters from Java program
  - convert received characters into bytes according to specified encoding
  - write them onto underlying stream
- Constructor

```
public OutputStreamWriter(OutputStream out, String encoding)
    throws UnsupportedEncodingException
```

  - encoding unspecified ⇒ default encoding in the platform
- Getter

```
public String getEncoding()
```

Readers

- Mirror of InputStream
  - declared abstract (never used directly)
- Methods

```
protected Reader()
protected Reader(Object lock)
public abstract int read(char[] text, int offset, int length)
    throws IOException
public int read() throws IOException
public int read(char[] text) throws IOException
public long skip(long n) throws IOException
public boolean ready()
public boolean markSupported()
public void mark(int readAheadLimit) throws IOException
public void reset() throws IOException
public abstract void close() throws IOException
```

the other two read() use this method

InputStreamReader

- Most important concrete subclass of Reader
  - read bytes from underlying input stream such as FileInputStream and TelnetInputStream
  - convert these bytes into characters according to specified encoding and return them
- Constructor

```
public InputStreamReader(InputStream in)
public InputStreamReader(InputStream in, String encoding)
    throws UnsupportedEncodingException
```

  - encoding unspecified ⇒ default encoding in the platform

Example

```
public static String getMacCyrillicString(InputStream in)
    throws IOException {

  InputStreamReader r = new InputStreamReader(in, "MacCyrillic");
  StringBuilder sb = new StringBuilder();
  int c;
  while ((c = r.read()) != -1) sb.append((char) c);
  return sb.toString();
}
```

Example

```java
import java.io.*;

public class StreamReaderWriterTest {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        if (args.length != 1) {
            System.out.println("사용법: java StreamReaderTest 파일명");
            System.exit(0);
        }

        FileInputStream fis = null;
        InputStreamReader isr = null;
        OutputStreamWriter osw = null;

        try {
            fis = new FileInputStream(args[0]);
            isr = new InputStreamReader(fis);
            osw = new OutputStreamWriter(System.out);
            char[] buffer = new char[512];
            int readcount = 0;
            while((readcount = isr.read(buffer)) != -1) {
                osw.write(buffer,0,readcount);
            }
            fis.close();
            isr.close();
            osw.close();
        } catch (Exception ex) {
            System.out.println(ex);
        }

    }

}
```

read from file and print on screen

FileReader and FileWriter

extends InputStreamReader extends OutputStreamWriter

- Constructors

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

```
public FileReader(String fileName) throws FileNotFoundException
public FileReader(File file) throws FileNotFoundException
public FileReader(FileDescriptor fd)
```

- FileReader and FileWriter always use the local default encoding
  - not recommended
- Alternative
  - InputStreamReader + FileInputStream
  - OutputStreamWriter + FileOutputStream

Example

```java
import java.io.*;

public class FileCopy {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        if(args.length != 2) {
            System.out.println("사용법: java FileCopy 파일명1 파일명2");
            System.exit(0);
        }

        FileReader fr = null;
        FileWriter fw = null;
        try {
            fr = new FileReader(args[0]);
            fw = new FileWriter(args[1]);
            char[] buffer = new char[512];
            int readcount = 0;
            while((readcount = fr.read(buffer)) != -1) {
                fw.write(buffer,0,readcount);
            }
            System.out.println("파일을 복사하였습니다");

            fr.close();
            fw.close();
        } catch (Exception ex) {
            System.out.println(ex);
        }
    }
}
```

BufferedReader and BufferedWriter

- BufferedReader and BufferedWriter
  - character-based equivalents of byte-based BufferedInputStream and BufferedOutputStream
  - use internal array of chars
- When a program reads from BufferedReader
  - text is taken from the buffer rather than directly from input streams or text source
  - when buffer empties, it is filled again with as much text as possible, for current and future use
- When a program writes onto BufferedWriter
  - text is placed in the buffer
  - text is moved to underlying output stream only when buffer fills up or writer is explicitly flushed

Methods

- Constructors

```
public BufferedReader(Reader in, int bufferSize)
public BufferedReader(Reader in)
public BufferedWriter(Writer out)
public BufferedWriter(Writer out, int bufferSize)
```

- Read a single line of text

```
public String readLine() throws IOException
```

- Insert a platform-dependent line-separator string into output

```
public void newLine() throws IOException
```

  - should not be used for network protocols that require explicit line separator

Example

```java
public static String getMacCyrillicString(InputStream in)
    throws IOException {

  Reader r = new InputStreamReader(in, "MacCyrillic");
  r = new BufferedReader(r, 1024);
  StringBuilder sb = new StringBuilder();
  int c;
  while ((c = r.read()) != -1) sb.append((char) c);
  return sb.toString();
}
```

- compare with

```java
public static String getMacCyrillicString(InputStream in)
    throws IOException {

  InputStreamReader r = new InputStreamReader(in, "MacCyrillic");
  StringBuilder sb = new StringBuilder();
  int c;
  while ((c = r.read()) != -1) sb.append((char) c);
  return sb.toString();
}
```