Lab 7

Java I/O

- Stream
  - The way a program connects I/O objects to send and receive data.
- Byte Stream
  - Inputs and outputs the binary data.
  - Ex) image, video
- Character Stream
  - Inputs and outputs the text data.
  - Ex) HTML document, txt file

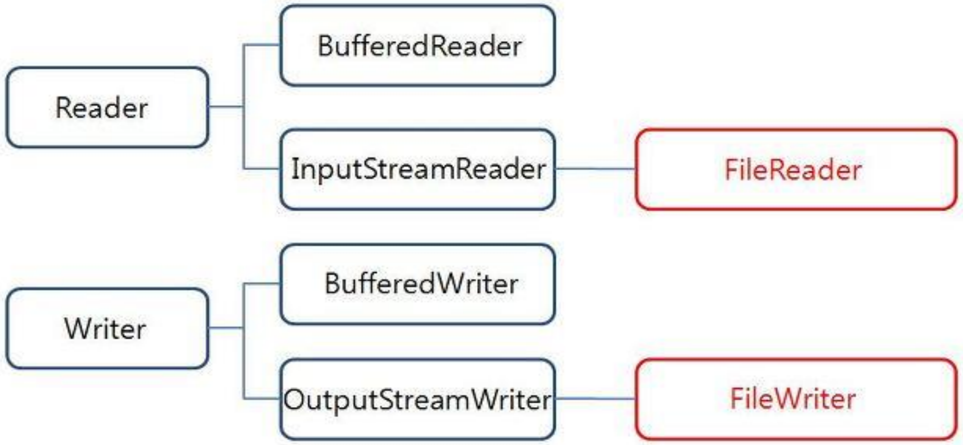
Sources & Sinks of Data

Binary	Character
• **InputStream**	• **Reader** converter: **InputStreamReader**
• **OutputStream**	• **Writer** converter: **OutputStreamWriter**
• **FileInputStream**	• **FileReader**
• **FileOutputStream**	• **FileWriter**
• **StringBufferInputStream**	• **StringReader**
• (no corresponding class)	• **StringWriter**
• **ByteArrayInputStream**	• **CharArrayReader**
• **ByteArrayOutputStream**	• **CharArrayWriter**
• **PipedInputStream**	• **PipedReader**
• **PipedOutputStream**	• **PipedWriter**

- Byte Stream
  - InputStream / OutputStream
    - Parents of byte-based input / output stream
  - ByteArrayInputStream / ByteArrayOutputStream
    - byte array(byte[])
  - FileInputStream / FileOutputStream
    - input and output stream for a file
- Character Stream
  - Reader / Writer
    - Parents of character-based input / output streadm
  - FileReader / FileWriter
    - Classes that input and output text-based files

- Sub Stream
  - The performance of the program follows the device with the lowest I/O.
  - For example, no matter how good CPU and memory performance is, if the input and output of the hard disk is slow, the performance of the program depends on the processing speed of the hard disk.
  - There is no complete solution to this, but instead of working directly with I/O sources, programs can improve execution to some extent by working with memory buffers in the middle.

- Character based Stream
  - FileReader / FileWriter
    - A class of I/O used to convert bytes stored in a file to Unicode characters and to convert Unicode characters to read or output to bytes in default character encoding and save them to a file.
    - File Reader and FileWriter are subclasses of InputStreamReader or OutputStreamWriter, respectively, which contain the ability to convert Unicode characters and bytes.
  - CharArrayReader / CharArrayWriter
  - PipedReader / PipedWriter
  - StringReader / StringWriter



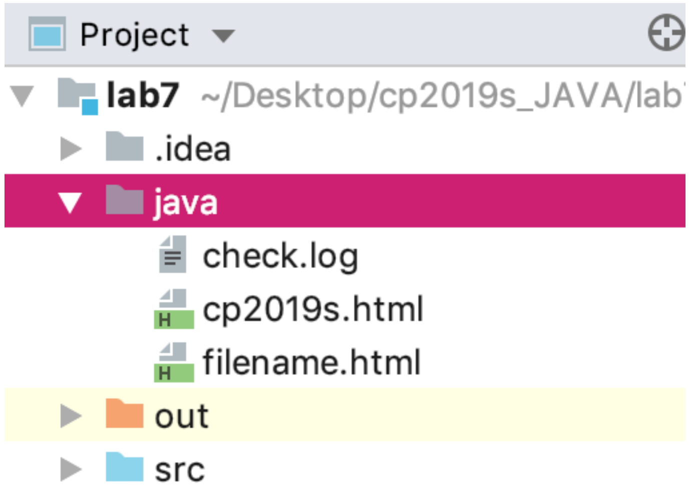
Getting bytes from a file

```
import java.io.*;
public class Read {
    public static void main(String[] args) {
    try {
        FileInputStream f = new FileInputStream("in.txt");
        int b;
        while ((b = f.read()) != -1)
            System.out.print((char) b);
            } catch (FileNotFoundException fnfe) {
            // System.out.println(fnfe);
            fnfe.printStackTrace();
            } catch (IOException ioe) {
            ioe.printStackTrace();
        }
    System.out.flush();
    }
}
```

Writting bytes to a file

```
import java.io.*;
public class Write {
        public static void main(String[] args) {
                try {
                byte ova[] = {'o', 'u', 't', '\n'};
                FileOutputStream f = new
                FileOutputStream(args[0]);
                f.write(ova);
                f.close();
                } catch (IOException ioe) {
                ioe.printStackTrace();
                }
        }
}
```

FilenameFilter : .HTML extension only filter

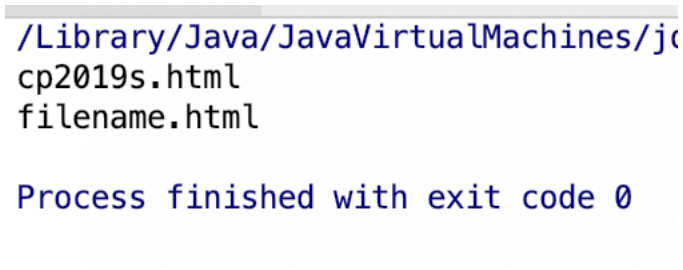


```java
import java.io.*;

class OnlyExt implements FilenameFilter {
    String ext;
    public OnlyExt(String ext) {
        this.ext = "." + ext;
    }
    public boolean accept(File dir, String name) {
        return name.endsWith(ext);
    }
}

public class DirListOnly {
    public static void main(String args[]) {
        String dirname = "./java";
        File f1 = new File(dirname);
        FilenameFilter only = new OnlyExt("html");
        String s[] = f1.list(only);
        for (int i=0; i < s.length; i++) {
            System.out.println(s[i]);
        }
    }
}
```

Result :



```
/Library/Java/JavaVirtualMachines/j
cp2019s.html
filename.html

Process finished with exit code 0
```

Modifying Stream Behavior

Binary	Character
• **FilterInputStream**	• **FilterReader**
• **FilterOutputStream**	• **FilterWriter** (abstract class with no subclasses)
• **BufferedInputStream**	• **BufferedReader** (also has **readLine( )**)
• **BufferedOutputStream**	• **BufferedWriter**
• **DataInputStream**	• Use **DataInputStream** (except when you must use **readLine( )**, then use a **BufferedReader**)
• **PrintStream**	• **PrintWriter**
• **LineNumberInputStream**	• **LineNumberReader**
• **StreamTokenizer**	• **StreamTokenizer** (Use constructor that takes a **Reader** instead)
• **PushBackInputStream**	• **PushBackReader**

© 1992-2012 by Pearson Education, Inc.
All Rights Reserved.

FilterReader & FilterWriter

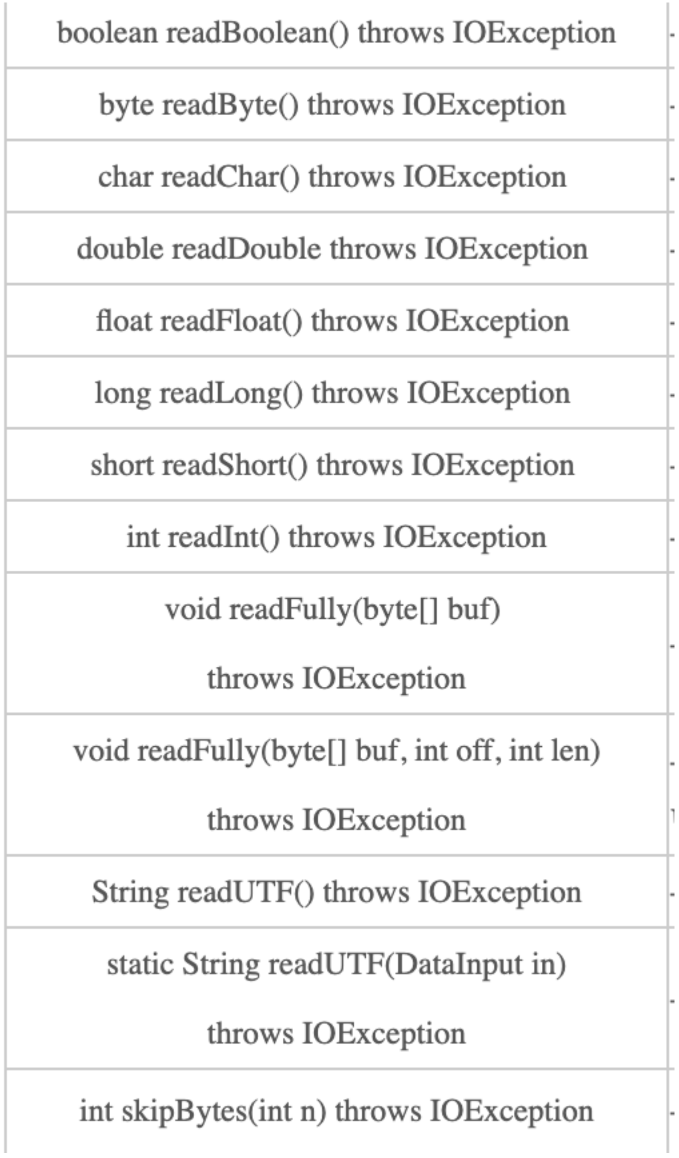
- The classes are abstract classes that read characters an d filter them in some way before passing the text along.
- You can imagine a FilterReader that converts all charact ers to uppercase.
- public abstract class FilterReader extends Reader
- public abstract class FilterWriter extends Writer
- There are no concrete subclasses of FilterWriter in the j ava packages and only one concrete subclass of FilterRe ader. These classes exist so you can write your own filte rs.

BufferedReader & BufferedWriter

- BufferedReader/BufferedWriter are IO classes from Buffer.
- For BufferedReader, readLine() method, reading lin e by line, makes file read very easy.
- For BufferedWriter, it is mandatory to use either flu sh() method call or close() method call because of t he buffer.

DataInputStream

- With DataOutputStream, data types (char, int, long, ...) can be read and written.
- (With File I/O Stream, only byt e[] data I/O possible)



boolean readBoolean() throws IOException
byte readByte() throws IOException
char readChar() throws IOException
double readDouble throws IOException
float readFloat() throws IOException
long readLong() throws IOException
short readShort() throws IOException
int readInt() throws IOException
void readFully(byte[] buf) throws IOException
void readFully(byte[] buf, int off, int len) throws IOException
String readUTF() throws IOException
static String readUTF(DataInput in) throws IOException
int skipBytes(int n) throws IOException

PrintWriter

- The class without Reader
- Prints formatted representations of objects to a text-ou tput stream.
- It does not contain methods for writing raw bytes, for which a program should use unencoded byte streams.
- Unlike the PrintStream class, if automatic flushing is en abled it will be done only when one of the println, print f, or format methods is invoked, rather than whenever a newline character happens to be output.
- Methods in this class never throw I/O exceptions, altho ugh some of its constructors may.

```
import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;

public class PrintWriter {

    public static void main(String[] args) throws IOException {

        //PrintWriter writer = new PrintWriter(System.out,true); // Auto flush

        PrintWriter writer = new PrintWriter(new FileWriter("./print.txt"),true); // Auto flush
with setting the file location

        writer.println("-------------");
        writer.printf("line line\n");
        writer.printf("line line2\n");
        writer.printf("line line3\n");

        // writer.flush(); => No need : Auto flush
        System.out.println(”File Written.");
    }
}
```

LineNumberReader

- A buffered character-input stream that keeps track of li ne numbers.
- This class defines methods setLineNumber(int) and getL ineNumber() for setting and getting the current line nu mber respectively.
- By default, line numbering begins at 0. This number inc rements at every line terminator as the data is read, an d can be changed with a call to setLineNumber(int).
- Note however, that setLineNumber(int) does not actual ly change the current position in the stream; it only cha nges the value that will be returned by getLineNumber( ).

StreamTokenizer

- The **Java.io.StreamTokenizer** class takes an input st ream and parses it into "tokens", allowing the toke ns to be read one at a time.
- The stream tokenizer can recognize identifiers, num bers, quoted strings, and various comment styles

PushBackReader

- The **java.io.PushbackReader** is intended to be used when you parse data from a Reader.
- Sometimes you need to read ahead a few character s to see what is coming, before you can determine how to interpret the current character.
- The class allows you to push back the read characte rs into the Reader. These characters will then be rea d again the next time you call read().

Example :

```
PushbackReader pushbackReader = new PushbackReader(new FileReader(”./input.txt"));
int data = pushbackReader.read();
pushbackReader.unread(data);
```

실습 – Stack 구현

- Custom stack implementation
  - JAVA ver.
  - C++ ver.

JAVA ver.

```java
import java.util.Arrays;

public class CustomStack <E>
{
   private int size = 0;
   private static final int DEFAULT_CAPACITY = 10;
   private Object elements[];

   public CustomStack() {
      elements = new Object[DEFAULT_CAPACITY];
   }

   public void push(E e) {
      if (size == elements.length) {
         ensureCapacity();
      }
      elements[size++] = e;
   }

   @SuppressWarnings("unchecked")
   public E pop() {
      E e = (E) elements[--size];
      elements[size] = null;
      return e;
   }

   private void ensureCapacity() {
      int newSize = elements.length * 2;
      elements = Arrays.copyOf(elements, newSize);
   }

   @Override
   public String toString()
   {
      StringBuilder sb = new StringBuilder();
      sb.append('[');
      for(int i = 0; i < size ;i++) {
         sb.append(elements[i].toString());
         if(i < size-1){
            sb.append(",");
         }
      }
      sb.append(']');
      return sb.toString();
   }
}
```

```java
public class Main

{

   public static void main(String[] args)

   {

      CustomStack<Integer> stack = new CustomStack<>();



      stack.push(10);

      stack.push(20);

      stack.push(30);

      stack.push(40);



      System.out.println(stack);



      System.out.println( stack.pop() );

      System.out.println( stack.pop() );

      System.out.println( stack.pop() );



      System.out.println( stack );

   }

}
```

C++ ver.

```
#include <iostream>

using namespace std;

class Node {
  public:
    int data;
    Node(int data);
    Node();
    ~Node();
};

Node::Node(int data) {
  this->data = data;
}

Node::~Node() {
  cout << "deleting node (" << this->data << ")" << endl;
}

class Stack {
  public:
    int capacity;
    int top;
    Node** nodes;

    Stack(int capacity);
    ~Stack();

    void push(int data);
    int pop();
    bool isEmpty();
    int getSize();
};

Stack::Stack(int capacity){
  this->nodes = new Node*[capacity];
  this->capacity = capacity;
  this->top = 0;
}

Stack::~Stack() {
  for(int i = 0; i < this->capacity; i++) {
    if(this->nodes[i]) {
      delete this->nodes[i];
    }
  }

  delete[] this->nodes;
}

void Stack::push(int data) {
  this->nodes[this->top++] = new Node(data);
  cout << "push: " << data << endl;
}

int Stack::pop() {
  int data = this->nodes[--(this->top)]->data;
  cout << "pop: " << data << endl;
  return data;
}

bool Stack::isEmpty() {
  return (this->top == 0);
}

int Stack::getSize() {
  return this->top - 1;
}

int main() {
  Stack stack(30);
  stack.push(1);
  stack.push(2);
  stack.push(3);

  cout << "size: " << stack.getSize() << endl;

  stack.pop();
  stack.pop();

  cout << "empty: " << (stack.isEmpty() ? "true" : "false") << endl;

  stack.pop();

  cout << "empty: " << (stack.isEmpty() ? "true" : "false") << endl;
}
```