Input and Output

Streams

- A **stream** is an abstraction that is used to deal with reading and writing data regardless of the medium on which the data is read from/written to (e.g. files, internet content, magnetic tapes, another application, etc.).
- An **input stream** is used to read data from a data source into a program.
- An **output stream** is used to send data from a program to a data destination.
- A **filtered stream** is a type of stream that modifies the way an existing stream is handled; it processes the data during the reading or writing operations before they reach the destination.
- Types of streams:
  - **Byte streams** can be used to process any data that can be represented as bytes (numbers from 0 to 255)
  - **Character streams** are specialized types of byte streams that handle only textual data.
- Both input and output streams need to be closed after being used.

Byte streams → FileInputStream

- Most frequently used type of streams.
- OS agnostic way of working with files.
- Methods:
  - `int read()` - Returns the byte which has been read or -1 if the end of the stream has been reached.
  - `read(byte[], int, int)` - Reads a specified number of bytes from a stream and stores them into the array starting from a specified position.

Example: ByteReader

Byte streams → FileOutputStream

Methods:

- `write(int)` - Writes a single byte into the stream.
- `write(byte[], int, int)` - Writes a specified number of bytes from the byte array starting from a specified position.

Example: ByteWriter

Byte streams → Filtered streams → Buffered streams

- A buffered input stream fills a buffer with data that hasn't been requested yet. When a program needs this data, it looks to the buffer first before going to the original stream source.
- With buffered output stream, when data is written it first goes to a buffer. Only when the buffer is filled up is the data written to the actual file.
- They provide a more efficient way of processing data.
- Constructors:
  - `BufferedInputStream(InputStream)`
  - `BufferedInputStream(InputStream, int)`
  - `BufferedOutputStream(OutputStream)`
  - `BufferedOutputStraem(OutputStream, int)`

Example: BufferDemo

Byte streams → Filtered streams → Data streams

- Convenient for working with data which is represented by the Java primitive types.
- Supported methods:
  - `readBoolean(), writeBoolean(boolean)`
  - `readByte(), writeByte(integer)`
  - `readDouble(), writeDouble(double)`
  - `readFloat(), writeFloat(float)`
  - `readInt(), writeInt(int)`
  - `readLong(), writeLong(long)`
  - `readShort(), writeShort(int)`

Example: PrimeWriter and PrimeReader

Example: ConsoleInput

Character streams → FileReader and BufferedReader

- FileReader methods:
  - `int read()` - Returns the next character on the stream as an integer.
  - `int read(char[], int, int)` - Reads characters into the specified character array with the indicated starting point and number of characters read. Returns the number of read characters.
- For better efficiency, use BufferedReader:
  - `BufferedReader(Reader)` - Creates a buffered character stream associated with the specified Reader object, such as FileReader.
  - `BufferedReader(Reader, int)` - Creates a buffered character stream associated with the specified Reader and with a buffer of int size.
  - `String readLine()` - Returns a line of text terminated with either:
    - A newline character ('\n'); A carriage return character ('\r'); A carriage return followed by a newline ("\n\r").

Example: SourceReader

Character streams → FileWriter and BufferedWriter

- FileWriter methods:
  - `write(int)` - Writes a character.
  - `write(char[], int, int)` - Writes characters from the specified character array with the indicated starting point and number of characters written.
  - `write(String, int, int)` - Writes characters from the specified string with the indicated starting point and number of characters written.
- For better efficiency, use BufferedWriter.

Predefined streams

- `System.in` - Byte input stream
- `System.out` - Character output stream
- `System.err` - Error character output stream

File

- Constructors:
  - `File(String)` - Creates a File object with the specified folder; no filename is indicated, so this refers only to a file folder.
  - `File(String, String)` - Creates a File object with the specified folder path and the specified name.
  - `File(File, String)` - Creates a File object with its path represented by the specified File and its name indicated by the specified String.
- Methods:
  - `boolean exists(); long length();`
  - `boolean renameTo(File); boolean delete();`
  - `String getName(); String getPath();`
  - `boolean mkDir(); boolean isDirectory();`
  - `File[] listFiles();`

Exercises

Exercise: WordCounter

Count the number of occurrences of each word in a file specified as an argument to the application.

Application

Application: Twitter

- Create social network application that (for now) allows a anonymous user to:
  1. Tweet a message.
  2. List all tweets in a chronological order with the latest tweet displayed first.
  3. Exit the application.
- The user chooses a functionality from a menu using the standard input stream (console).
- Hint: The java.util.Scanner class offers a convenient set of methods for capturing input from a stream.