

15 Streams and files

```
import java.io.*;
```

Overview

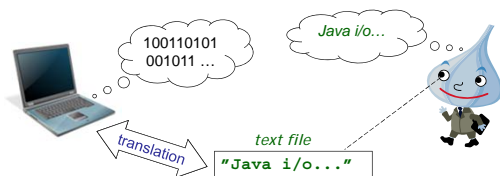
- Binary data vs textual data
- Simple file processing - examples
- The stream model
- Bytes and characters
- Buffering
- Byte streams
- Character streams
- Binary streams
- Direct access files

Object oriented programming, DAT042, D2, 12/13, lp 1

Lecture 15 2

Binary data vs text

- Internally, all data is stored in binary format during program execution.
- *Text is more readable to humans than bits!*

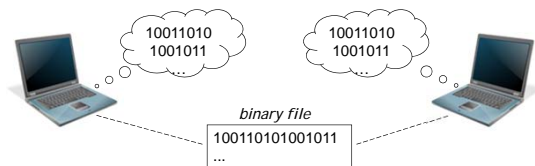


Object oriented programming, DAT042, D2, 12/13, lp 1

Lecture 15 3

Binary data vs text

- *... but bits are more readable to computers than text.*
- *No need for expensive translations.*
- *Binary files use less space than text files.*



Object oriented programming, DAT042, D2, 12/13, lp 1

Lecture 15 4

input-output

- The `java.io` package supports input-output.
- Input-output is particularly error-prone.
 - It involves interaction with the external environment.
- `java.io.IOException` is a checked exception.

Object oriented programming, DAT042, D2, 12/13, lp 1

Lecture 15 5

Readers, writers, byte streams

- Readers and writers deal with textual input and output.
 - Based around the `char` type.
- Byte streams deal with binary data.
 - Based around the `byte` type.
- The *address-book-io* project illustrates textual IO.

Object oriented programming, DAT042, D2, 12/13, lp 1

Lecture 15 6

Text output to file

- Use the `FileWriter` class.
 - Open a file.
 - Write to the file.
 - Close the file.
- Failure at any point results in an `IOException`.

Object oriented programming, DAT042, D2, 12/13, lp 1

Lecture 15 7

Text output to file

```
try {
    FileWriter writer =
        new FileWriter("name of file");
    while(there is more text to write) {
        ...
        writer.write(next piece of text);
        ...
    }
    writer.close();
}
catch(IOException e) {
    something went wrong when accessing the file
}
```

Object oriented programming, DAT042, D2, 12/13, lp 1

Lecture 15 8

Text input from a file

- Use the `FileReader` class.
- Augment with `BufferedReader` for line-based input.
 - Open a file.
 - Read from the file.
 - Close the file.
- Failure at any point results in an `IOException`.

Object oriented programming, DAT042, D2, 12/13, lp 1

Lecture 15 9

Text input from the keyboard

- `System.in` maps to the keyboard.
 - `java.io.InputStream`
- Often wrapped in a `java.util.Scanner`:
 - `new Scanner(System.in);`
- `Scanner` supports *parsing* of textual input.
 - `nextInt`, `nextLine`, etc.
- `Scanner` with `File` an alternative to `BufferedReader` with `FileReader`.
 - `new Scanner(new File("filename"));`

Object oriented programming, DAT042, D2, 12/13, lp 1

Lecture 15 10

Text input from a file using a `FileReader`

```
try {
    BufferedReader reader =
        new BufferedReader(
            new FileReader("filename"));
    String line = reader.readLine();
    while(line != null) {
        do something with line
        line = reader.readLine();
    }
    reader.close();
}
catch(FileNotFoundException e) {
    the specified file could not be found
}
catch(IOException e) {
    something went wrong with reading or closing
}
```

Object oriented programming, DAT042, D2, 12/13, lp 1

Lecture 15 11

Text input from the keyboard using a `Scanner`

```
try {
    Scanner in = new Scanner(System.in);

    while(in.hasNextLine()) {
        String line = in.nextLine();
        do something with line
    }
}
catch(FileNotFoundException e) {
    the specified file could not be found
}
catch(IOException e) {
    something went wrong with reading or closing
}
```

Object oriented programming, DAT042, D2, 12/13, lp 1

Lecture 15 12

The stream i/o model

- Isolates programming from low level environment details
 - Streams add an abstract layer smoothing out the differences between various sources (or targets) for i/o operations.
- Streams can connect to external files, communication sockets, internal data structures like arrays and strings, etc

Object oriented programming, DAT042, D2, 12/13, lp 1

Lecture 15 13

Bytes and characters

- Byte
 - 8 bit signed integer [-128,127]
- Character
 - Java uses Unicode 16 bit character *internally* in programs.
- Common *external* character formats:
 - Unicode 16 bit character
 - ASCII 8 bit
 - UTF-8 variable length
 - "UTF8"
 - LATIN_1
 - "8859_1"
 - MS-DOS (Swedish)
 - "Cp850"

Object oriented programming, DAT042, D2, 12/13, lp 1

Lecture 15 14

Buffering

- A large number of single byte or character i/o operations is inefficient.
- By using *buffering* larger chunks of data can be treated in fewer i/o operations
 - thus faster performance.
- Java uses *buffered streams* to implement buffered i/o.

Object oriented programming, DAT042, D2, 12/13, lp 1

Lecture 15 15

Wrapper classes

- Many classes in `java.io` act as *wrapper classes* (the *decorator* design pattern).
- Ex.

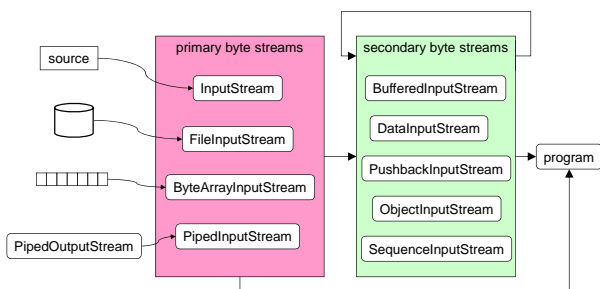


```
PrintWriter out =  
    new PrintWriter(  
        new BufferedWriter(  
            new FileWriter("output file name")));
```

Object oriented programming, DAT042, D2, 12/13, lp 1

Lecture 15 16

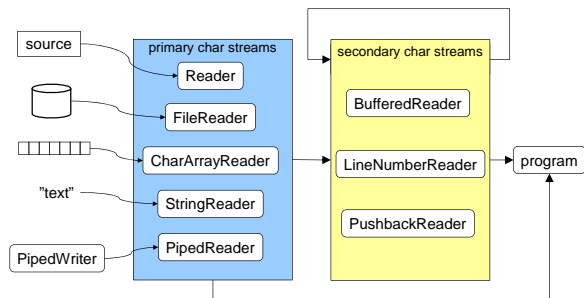
Byte input stream data flow



Object oriented programming, DAT042, D2, 12/13, lp 1

Lecture 15 17

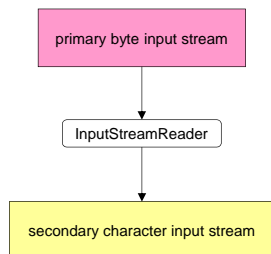
Character input stream data flow



Object oriented programming, DAT042, D2, 12/13, lp 1

Lecture 15 18

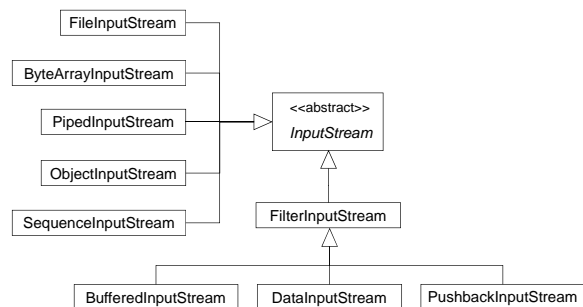
Bridging from byte in-streams to character in-streams



Object oriented programming, DAT042, D2, 12/13, lp 1

Lecture 15 19

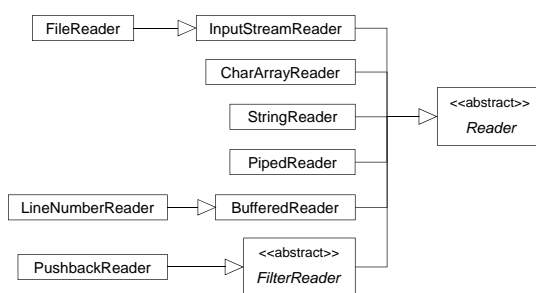
Byte input stream class relations



Object oriented programming, DAT042, D2, 12/13, lp 1

Lecture 15 20

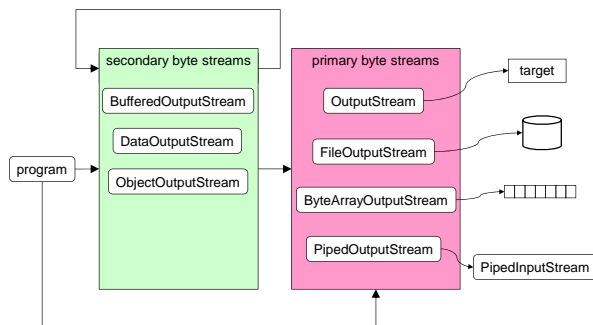
Character input stream class relations



Object oriented programming, DAT042, D2, 12/13, lp 1

Lecture 15 21

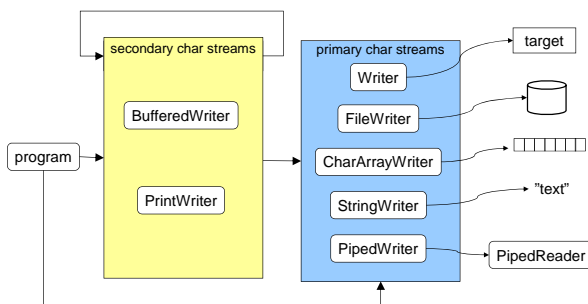
Byte Output stream data flow



Object oriented programming, DAT042, D2, 12/13, lp 1

Lecture 15 22

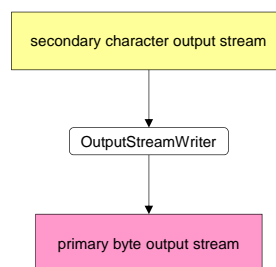
Character output stream data flow



Object oriented programming, DAT042, D2, 12/13, lp 1

Lecture 15 23

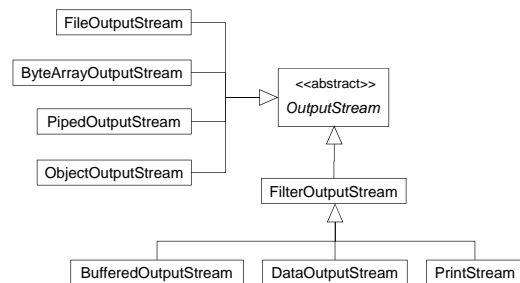
Bridging from character out-streams to byte out-streams



Object oriented programming, DAT042, D2, 12/13, lp 1

Lecture 15 24

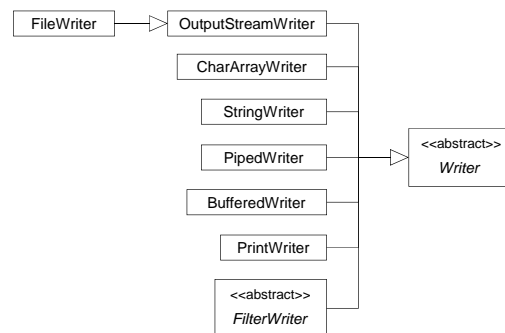
Byte output stream class relations



Object oriented programming, DAT042, D2, 12/13, lp 1

Lecture 15 25

Character output stream class relations



Object oriented programming, DAT042, D2, 12/13, lp 1

Lecture 15 26

Byte input stream operations

```
abstract class InputStream
int    read()
int    read(byte[] b)
int    read(byte[] b,int offset,int length)
long   skip(long n)
int    available()
boolean markSupported()
void   mark(int limit)
void   reset()
void   close()
```

Object oriented programming, DAT042, D2, 12/13, lp 1

Lecture 15 27

Byte output stream operations

```
abstract class OutputStream
void write(int b)
void write(byte[] b)
void write(byte[] b,int offset,int length)
void flush()
void close()
```

Object oriented programming, DAT042, D2, 12/13, lp 1

Lecture 15 28

Character input stream operations

```
abstract class Reader
int    read()
int    read(char[] b)
int    read(char[] b,int offset,int length)
long   skip(long n)
boolean ready()
boolean markSupported()
void   mark(int limit)
void   reset()
void   close()
```

Object oriented programming, DAT042, D2, 12/13, lp 1

Lecture 15 29

Character output stream operations

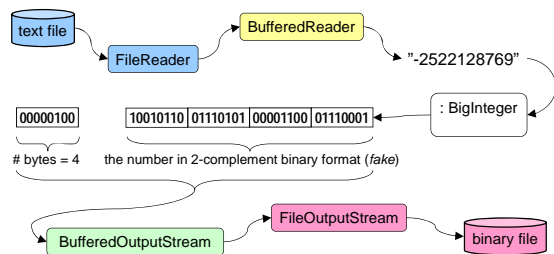
```
abstract class Writer
void write(int c)
void write(char[] b)
void write(char[] b,int offset,int length)
void write(String str)
void write(String str,int offset,int length)
void flush()
void close()
```

Object oriented programming, DAT042, D2, 12/13, lp 1

Lecture 15 30

Ex. Text to binary conversion

- Read a text file containing long digit strings and write the corresponding numbers to a binary file. Explore the *text_to_binary* project.



Object oriented programming, DAT042, D2, 12/13, lp 1

Lecture 15 31

Ex. Text to binary (2)

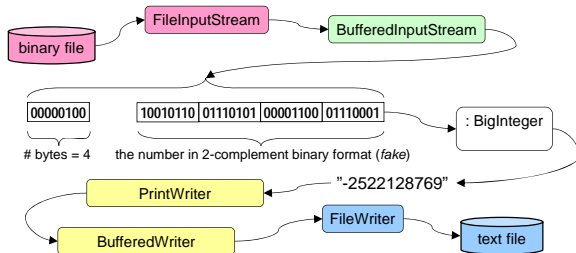
```
BufferedReader in =
    new BufferedReader(
        new FileReader("text infile name"));
OutputStream out =
    new BufferedOutputStream(
        new FileOutputStream("binary outfile name"));
...
inputLine = in.readLine();
if ( inputLine == null )
    // done
BigInteger i = new BigInteger(inputLine);
byte[] bytes = i.toByteArray();
out.write(bytes.length);
out.write(bytes,0,bytes.length);
...
} loop
```

Object oriented programming, DAT042, D2, 12/13, lp 1

Lecture 15 32

Ex. Binary to text conversion

- Read a binary file containing large numbers in 2-complement binary form and write the corresponding digits to a text file. Explore the *binary_to_text* project.



Object oriented programming, DAT042, D2, 12/13, lp 1

Lecture 15 33

Ex. Binary to text (2)

```
BufferedInputStream in =
    new BufferedInputStream(
        new FileInputStream("binary infile name"));
PrintWriter out =
    new PrintWriter(
        new BufferedWriter(
            new FileWriter("text outfile name")));
...
int noOfBytes = in.read();
byte[] buf = new byte[noOfBytes];
if ( in.read(buf,0,noOfBytes) == -1 )
    // done
BigInteger i = new BigInteger(buf);
out.println(i);
...
} loop
```

Object oriented programming, DAT042, D2, 12/13, lp 1

Lecture 15 34

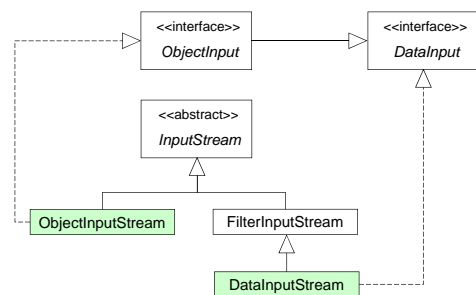
Binary streams

- Binary data are treated differently depending on their kind of data type.
- For built in types like int, char, float, ... use *DataStreams*
- For object types, use *ObjectStreams*

Object oriented programming, DAT042, D2, 12/13, lp 1

Lecture 15 35

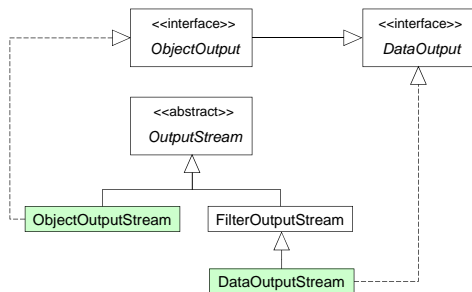
Byte input stream class relations



Object oriented programming, DAT042, D2, 12/13, lp 1

Lecture 15 36

Byte output stream class relations



Object oriented programming, DAT042, D2, 12/13, lp 1

Lecture 15 37

DataInput operations

interface DataInput	
boolean	readBoolean()
byte	readByte()
char	readChar()
short	readShort()
int	readInt()
long	readLong()
float	readFloat()
double	readDouble()
String	readUTF()

Object oriented programming, DAT042, D2, 12/13, lp 1

Lecture 15 38

DataInput operations (2)

interface DataInput (cont.)	
int	readUnsignedByte()
int	readUnsignedShort()
void	readFully(byte[] b)
void	readFully(byte[] b, int off, int len)
void	skipBytes(int len)

Object oriented programming, DAT042, D2, 12/13, lp 1

Lecture 15 39

DataOutput operations

interface DataOutput	
void	writeBoolean()
void	writeChar()
void	writeDouble()
void	writeShort()
void	writeInt()
void	writeLong()
void	writeFloat()
void	writeDouble()
void	writeUTF(String s)

Object oriented programming, DAT042, D2, 12/13, lp 1

Lecture 15 40

DataOutput operations (2)

interface DataOutput (cont.)	
void	writeChars(String s)
void	writeBytes(String s)
void	write(int b)
void	write(byte[] b)
void	write(byte[] b, int off, int len)
int	size()

Object oriented programming, DAT042, D2, 12/13, lp 1

Lecture 15 41

Ex. Array save/load operations

```
public class ArrayIo {
    public static void
    save(long[] array, String fileName)
    throws IOException
    { ... }

    public static long[]
    load(String fileName) throws IOException
    { ... }
}
```

Explore the *array_io* project.

Object oriented programming, DAT042, D2, 12/13, lp 1

Lecture 15 42

Array save operation

```
public static void
save(long[] array, String fileName)
throws IOException
{
    DataOutputStream out =
        new DataOutputStream(
            new FileOutputStream(fileName));
    out.writeInt(array.length);
    for ( long element : array )
        out.writeLong(element);

    out.close();
}
```

Object oriented programming, DAT042, D2, 12/13, lp 1

Lecture 15 43

Array load operation

```
public static long[]
load(String fileName)
throws IOException
{
    DataInputStream in =
        new DataInputStream(
            new FileInputStream(fileName));
    int size = in.readInt();
    long[] array = new long[size];
    for ( int i = 0; i < array.length; i++ )
        array[i] = in.readLong();

    in.close();
    return array;
}
```

Object oriented programming, DAT042, D2, 12/13, lp 1

Lecture 15 44

Binary streams of object types

- Whole networks of inter-connected objects may be "flattened" and written to object streams
 - and later be read back into the program again.
- To state that an object can be flattened, it's class must implement
interface Serializable
- Typical application: Saving the program state for later resumption, e.g. in computer games.
- *More on this subject in a forthcoming lecture!*

Object oriented programming, DAT042, D2, 12/13, lp 1

Lecture 15 45

Random access files

- A random access file consists of a sequence of *equally sized records*.
- A *file pointer* contains the current position for the next read/write operation.
- The file pointer is advanced after each read/write operation, but it may also be *manipulated explicitly* to direct i/o-operations to particular records.

Object oriented programming, DAT042, D2, 12/13, lp 1

Lecture 15 46

Random access files (2)

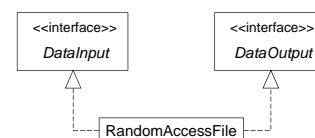
- All records must have *exactly the same size*.
- Record types solely based on the primitive data types can be handled without complications.
- If a record type contains *strings*, they have to be *padded* to make up for size differences.
- Typical application: A database file of updatable bank accounts.

Object oriented programming, DAT042, D2, 12/13, lp 1

Lecture 15 47

Random access file type relations

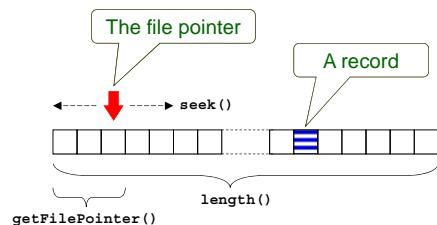
- Random access files are *both* input files and output files at the same time!



Object oriented programming, DAT042, D2, 12/13, lp 1

Lecture 15 48

Random access file operations



Object oriented programming, DAT042, D2, 12/13, lp 1

Lecture 15 49

Random access file operations (2)

class RandomAccessFile	
<i>Object construction</i>	Mode is either "r" (read), or "wr" (read/write).
<code>RandomAccessFile (name,mode)</code>	
<code>long getFilePointer ()</code>	Return the file pointer offset measured from the beginning of this file.
<code>void seek (pos)</code>	Set file pointer offset measured from the beginning of this file.
<code>long length ()</code>	Return the file length measured in bytes.
<code>void setLength (n)</code>	Set the file length to n bytes.

Object oriented programming, DAT042, D2, 12/13, lp 1

Lecture 15 50

The java.io.File class

- Objects of class **File** provide system-independent information about files and directory pathnames.
- Useful for directory traversal, file creation, renaming, ...

Object oriented programming, DAT042, D2, 12/13, lp 1

Lecture 15 51