Java I/O

Università di Modena e Reggio Emilia
Prof. Nicola Bicocchi (nicola.bicocchi@unimore.it)

ING

Stream

- All I/O operations rely on the abstraction of STREAM ("bytes flow")
- Random access unsupported (see java.io.RandomAccessFile)
- I/O operations work in the same way with ALL kinds of streams. For example, a stream can be:
  - Standard input, output, error
  - A regular file
  - A data-flow from/to memory or a pipe, …
  - A network connection

Stream

- Abstract classes Reader Writer
  - stream of chars (Unicode Chars 16 bit)
    - Characters
- Abstract classes InputStream OutputStream
  - stream of bytes (8 bit)
    - Binary data (e.g., sounds, images)
- package java.io
- All related exceptions are subclasses of java.io.IOException

java.io.Reader

- int read()
  - Reads a single character.
- int read(char[] buffer)
  - Reads characters into a char array.
- long skip(long n)
  - Skips characters.
- void close()
  - Closes the stream.

java.io.InputStream

- int read()
  - Reads a single byte (packed into an int).
- int read(byte[] buffer)
  - Reads bytes into a char array.
- long skip(long n)
  - Skips bytes.
- void close()
  - Closes the stream.

java.io.Writer

- void write(int c)
  - Writes a single character.
- void write(char[] buffer)
  - Writes an array of characters.
- void write(String str)
  - Writes a string.
- void flush()
  - Flushes the stream.
- void close()
  - Closes the stream, flushing it first.

java.io.OutputStream

- void write(int b)
  - Writes a single byte.
- void write(byte[] buffer)
  - Writes an array of bytes.
- void flush()
  - Flushes the stream.
- void close()
  - Closes the stream, flushing it first.

java.io.PrintStream

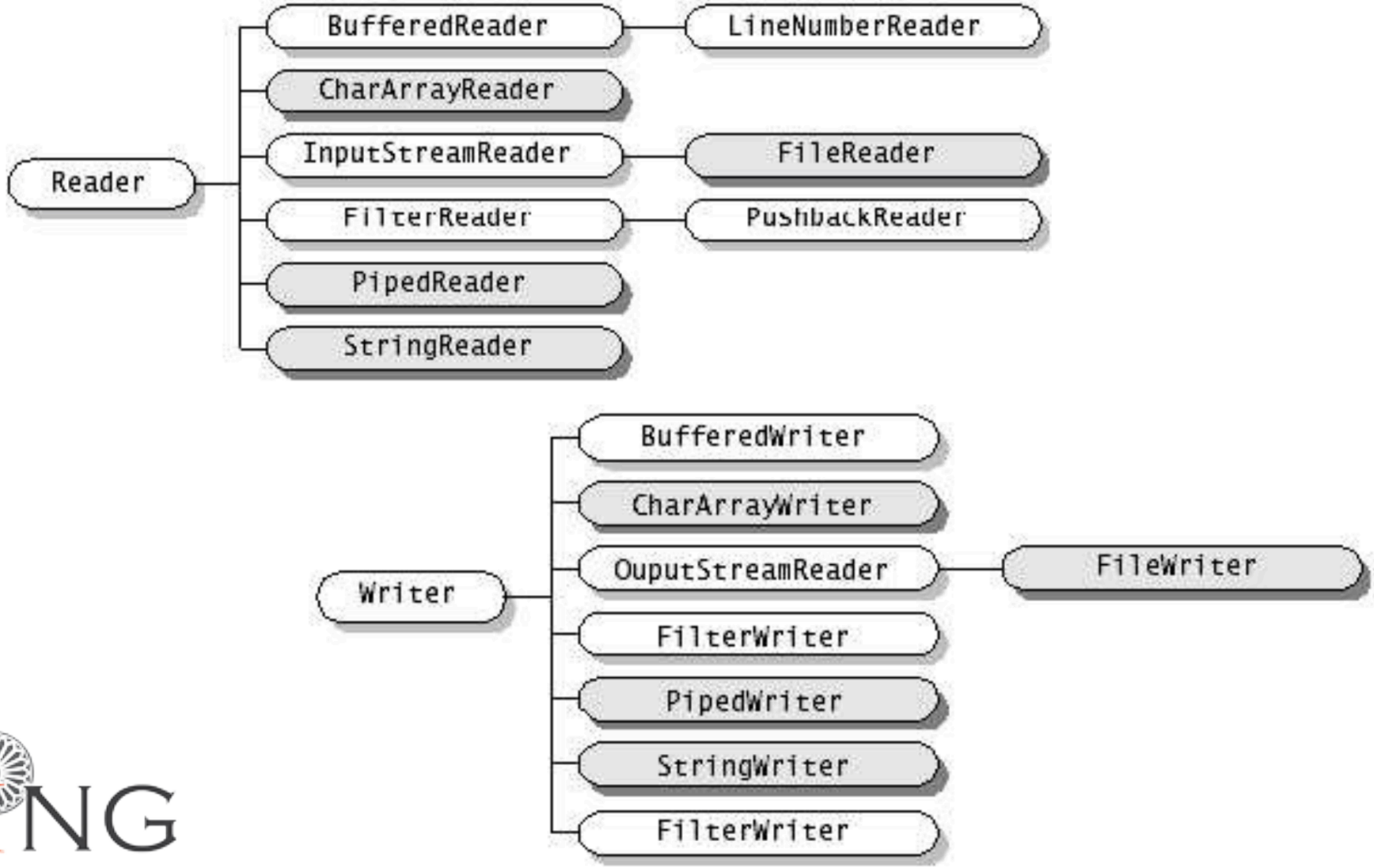
- A PrintStream adds functionality to another output stream, namely the ability to print representations of various data values conveniently.
- Adds methods like print(), println(), printf(), format() for string formatting
- Unlike other output streams, a PrintStream never throws an IOException; instead, exceptional situations merely set an internal flag that can be tested via the checkError method.

System.in and System.out

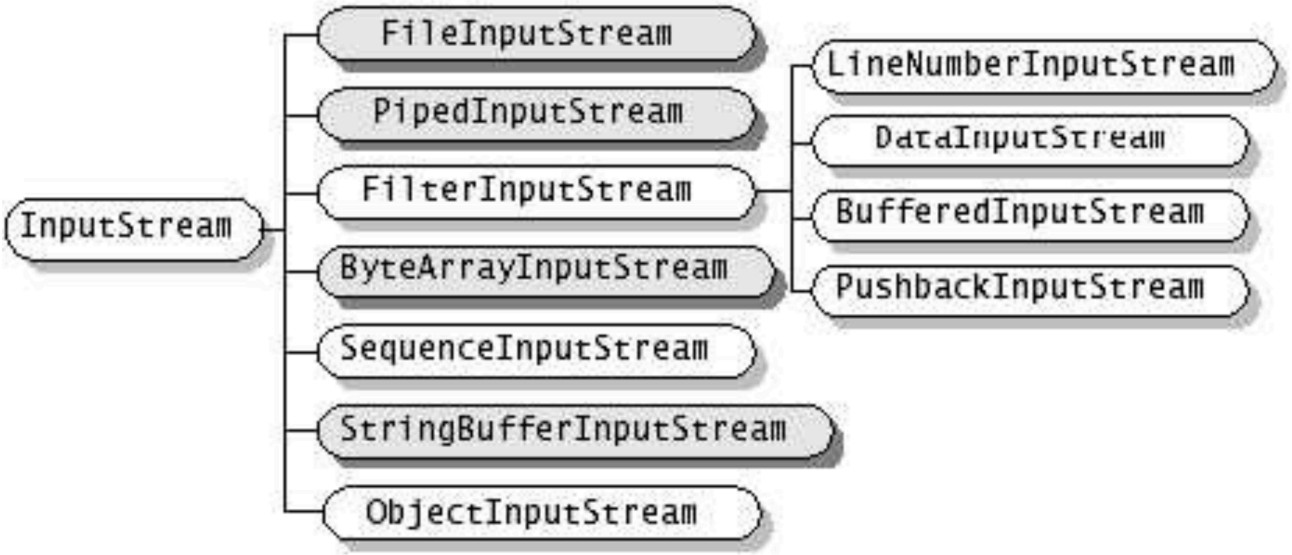
- System defines default input, output and error streams as

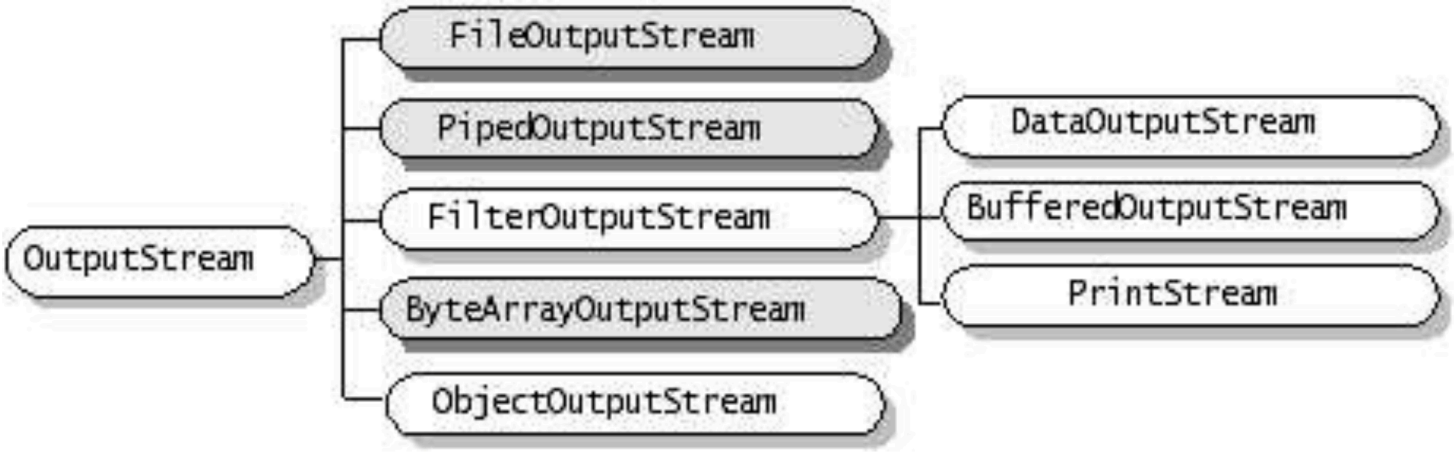
```
class System {
   static InputStream in;
   static PrintStream out;
   static PrintStream err;
   …
}
```

Reader and Writer



InputStream and OutputStream





ING

Read/Write of files

- FileReader
- FileWriter
  - R/W char from file
- FileInputStream
- FileOutputStream
  - R/W byte from file
- File
  - handles filename and pathname

Read/Write of pipes

- PipedReader
- PipedWriter
  - R/W chars from pipe
- PipedInputStream
- PipedOutputStream
  - R/W bytes from pipe

Read/Write in memory

- CharArrayReader
- CharArrayWriter
  - R/W chars from/to array in memory
- StringReader
- StringWriter
  - R/W chars from/to String
- ByteArrayInputStream
- ByteArrayOutputStream
  - R/W bytes from/to array in memory

Buffered Streams

- Transparently add buffering functionality. The manual alternative is to use read(int[] buf) or read(char[] buf)
- BufferedInputStream
- BufferedOutputStream
- BufferedReader
- BufferedWriter
- Examples:
  - `BufferedInputStream(InputStream in)`
  - `BufferedReader(Reader in)`

Interpreted Streams

- Translates primitive types in standard format (UTF-8) on file
- `DataInputStream(InputStream i)`
  - `readByte(), readChar(), readDouble(), readFloat(), readInt(), readLong(), readShort()`
- `DataOutputStream(OutputStream o)`
  - `writeByte(), writeChar(), writeDouble(), writeFloat(), writeInt(), writeLong(), writeShort()`

java.io.File

- An abstract representation of file and directory pathnames. Provides a conversion between File and String
- methods:
  - exists(), isFile(), isDirectory(), isHidden(), length(), canRead(), canWrite(), canExecute(), getPath()…
- File f = new File("/etc/passd").
  - Is this truly portable?

java.io.File

- Platform dependent
  - `File f = new File("tmp/abc.txt");`
- Platform independent
  - `File f = new File(“tmp” + File.separator + "abc.txt");`

- See Java System Properties
- See File static attributes

java.nio.file.Files

- This class consists exclusively of static methods that operate on files, directories, or other types of files.
- In most cases, the methods defined here will delegate to the associated file system provider to perform the file operations.

java.util.StringTokenizer

- StringTokenizer
  - Works on String
  - set of delimiters (blank, “,”, \t, \n, \r, \f)
  - Blank is the default delimiter
  - Divides a String in tokens (separated by delimiters), returning the token
  - hasMoreTokens(), nextToken()
  - Does not distinguish identifiers, numbers, comments, quoted strings

java.io.StreamTokenizer

- StreamTokenizer
  - Works on Stream (Reader subclasses)
  - More sophisticated, recognizes identifiers, comments, quoted string, numbers
  - Use symbol table and flag
  - nextToken(), TT_EOF if at the end

Deep and Shallow copies

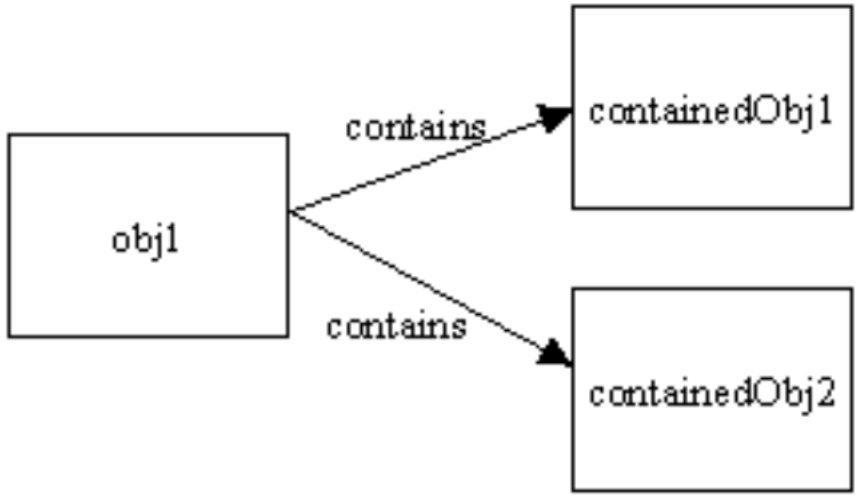


Figure 1. The original state of obj1

ING

Serialization

- Read / write of an object imply:
  - read/write attributes (and optionally the type) of the object
  - Correctly separating different elements
  - When reading, create an object and set all attributes values
- These operations (serialization) are automated by
  - ObjectInputStream
  - ObjectOutputStream

Serialization

- Methods to read/write objects are:
  - void writeObject(Object)
  - Object readObject()
- ONLY objects implementing interface Serializable can be serialized. Serializable is an empty interface. It is used to avoid serialization of objects, without the permission of the class developer

Serialization, deep copy

- An ObjectOutputStream saves automatically all objects referred by its attributes
  - objects serialized are numbered in the stream
  - references are saved like ordering numbers in the stream
- If I save 2 objects pointing to a third one, this is saved just once
  - Before saving an object, ObjectOutputStream checks if it has not been already saved
  - Otherwise it saves just the reference (as a number)

Serialization, type recovery

- When reading, an object is created
- ... but which is its type?
- Down casting to the exact type is useful only to send specific messages
- A viable solution could be down casting to a common ancestor

Deep and Shallow copies

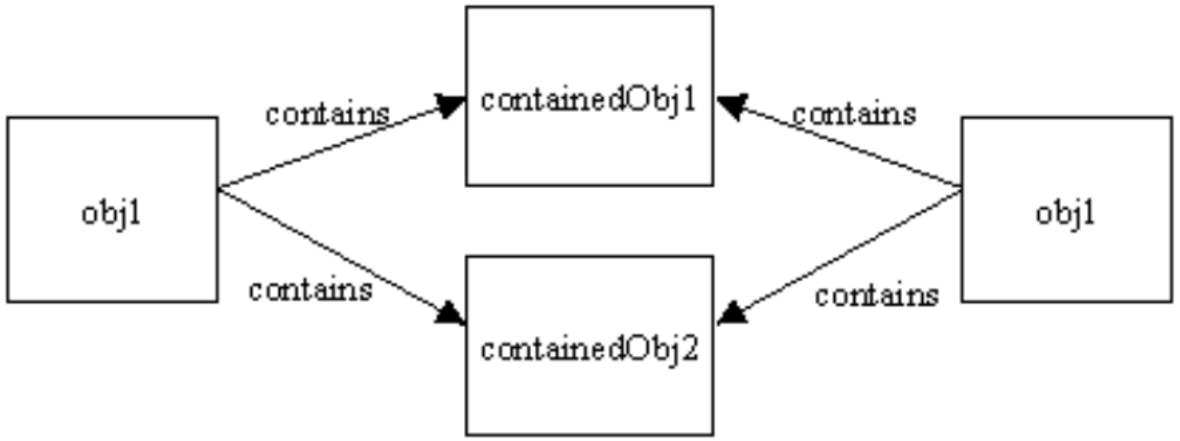


Figure 2. After a shallow copy of obj1

ING

Deep and Shallow copies

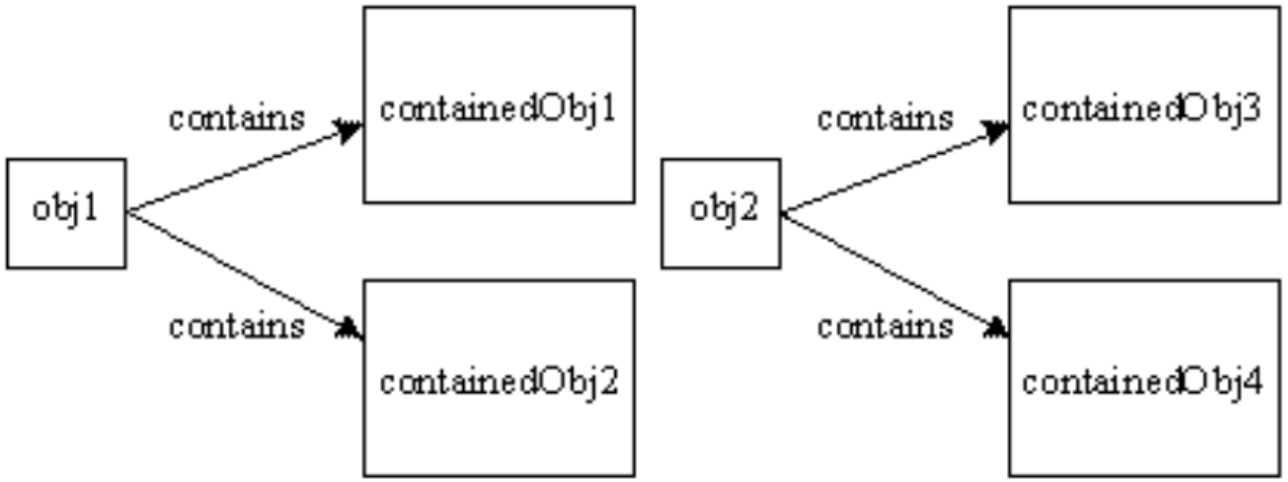


Figure 3. After a deep copy of obj1

ING