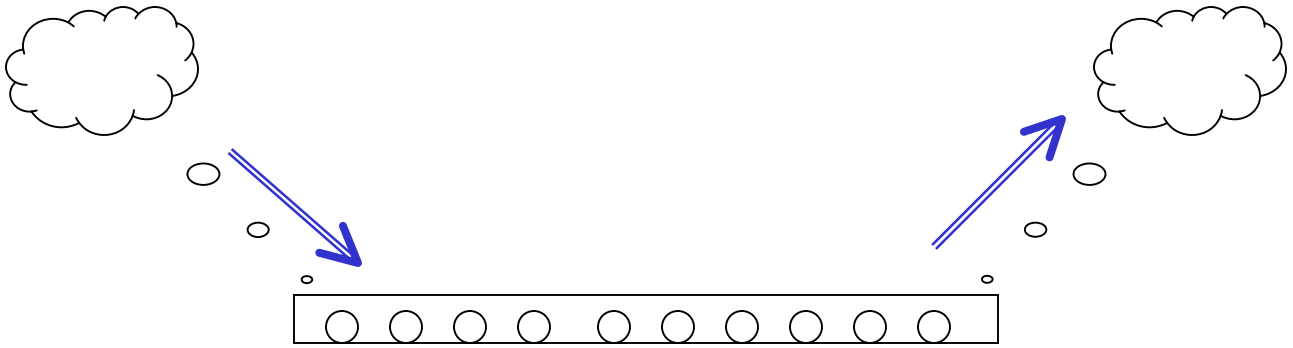
Data Survival
Streams, Files, Persistence

- Persistence
- Stream Communication
- Java Streams
- Sources and Sinks
- File I/O
- Object Persistence

What is Persistence?

- OO alternative to data i/o
- Objects "survive" program termination
- how?
- written to file at program termination
- read from file at program start



..so we need file i/o

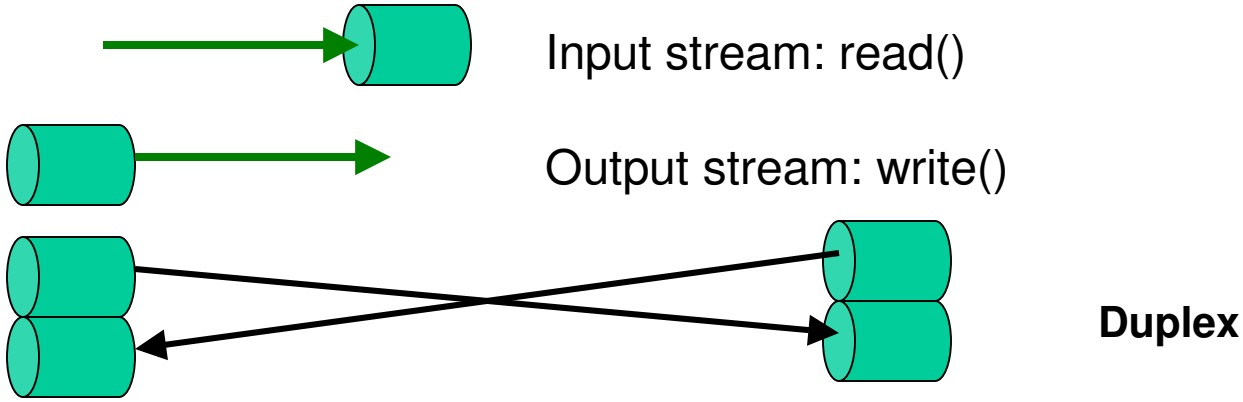
Persistence: course template

```
// Javbva-like
class Persistent {
  private File file;
  public Persistent (File file) {
     this.file = file;
     attr1 = file.readAttribute();
     attr.2 = file.readAttribute();
     ...
  }
  public void save() {
      file.writeAttribute(attr1);
      file.writeAttribute(attr2);
     ...
  }
  public void finalize() { save(); }
}
```

Streams

A Stream behaves like a can phone

It provides an ordered sequence of data.



It connects a sender to a receiver.

Communication depends on properties of the stream, not of sender or receiver.

Stream connections in Java

Connection between	Connection Type	Implementation
Java objects (Threads)	general channel type	Connection between two streams (Piping)
Java program and external devices	Reading from data sources, writing to data sinks	Console (Keyboard, Screen) File (String) (sources and sinks)
Java programs on different machines		Socket (source and sink)

I.e. there is a single concept behind console io, file io and networking

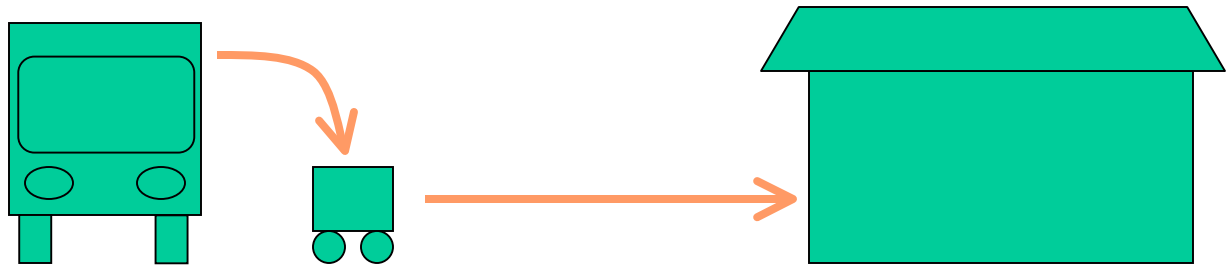
Stream Basics

- *InputStreamType* offers
  - **`inStream.read()`**
  - **`close()`**
- *OutputStreamType* offers
  - **`outStream.write(msg)`**
  - **`close()`**

If you want to test the following examples, use ***`Console`*** *as source and sink:*
as InputStreamType: ***`readln`****()*
as OutputstreamType: ***`println`****()*
msg Type is String

Behavior of Output Streams

- Method: `outstream.write(msg)`
- Writing object has to be active,
  (Stream does not ask for input ...)
- Will the sent message be received?
  ... possibly delayed ...



transport happens only when carrier full

- `flush()` causes immediate transport.

StreamWriter

```
public interface StreamWriter{
   public void send (MsgType msg);
}
```

```
public class GenericStreamWriter
                          implements StreamWriter{

   private OutputStreamType out;

   public GenericStreamWriter
                            (OutputStreamType out)
   {    this.out = out; }

   public void send(MsgType msg) {
        out.write(msg);
        out.flush();   // evtl.
   }
}
```

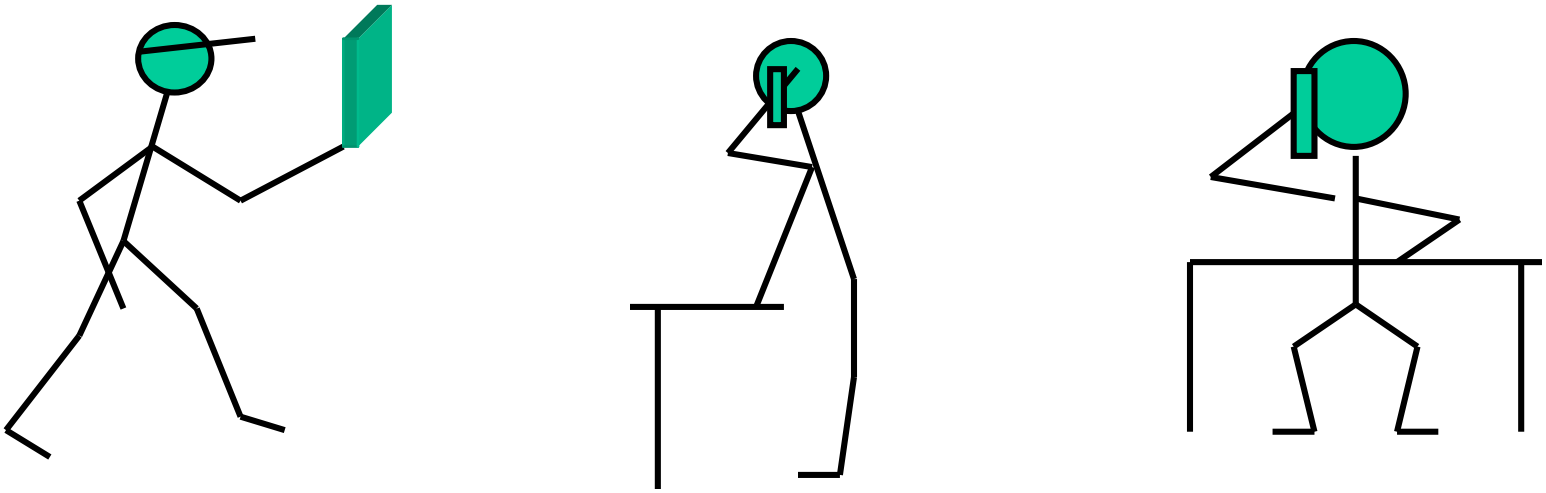
Behavior of Input Streams

- Method: `instream.read()`
- Reading object has to actively look for messages
  – no input notification
- Can you try to `read()` to check for messages ?
- NO:
  `read()` blocks until message arrives

- Devise independant active reader:
  – calls read() in an endless loop, always waiting for msg
  – if message received, it provides it
  – actively: Raising an event or callback
  – passively: Enter into a channel which supports message checking

Defusing the Read Blockage: Active Reader

Porter actively waits for mail.

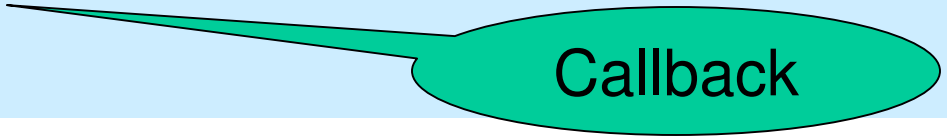


event driven: porter phones "parcel received"
passive: clerk calls at times, checking for the parcel.

Stream Reader

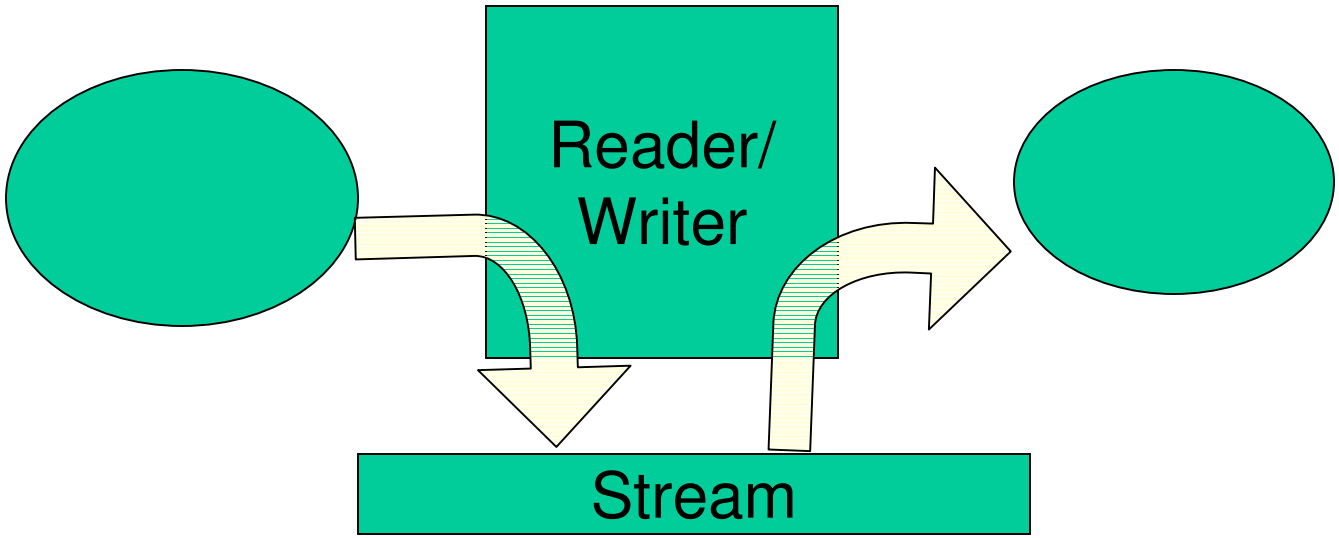
```
public interface StreamReaderListener {
   public void messageReceived(MsgType msg);
}
```

```
public class GenericStreamReader
                          extends AnimateObject {

   private InputStreamType in;
   private StreamReaderListener ear;

   public GenericStreamReader
                            (InputStreamType in)
   {    this.in = in; }
   public void addListener(StreamReaderListener ear)
   {    this. ear = ear;  }

   public void act() {
        ohr.messageReceived(eing.read()); // blocks
   }
}
```



Reader-Writer-Combination

- constitutes a communication manager
- can enforcerules for reading and writinf
  → en/decoding, protocols



Stream Communicator

```
public class Communicator
                    extends GenericStreamReader
                    implements StremaWriter {
   /* both reading and writing interface */
   private OutputStreamType out;
   private InputStreamType in;

   public Communicator ( OutputStreamType out,
                         InputStreamType in) {
       this.out = out;
       this.in = in;
   }
 // read method inherited from GnericStreamReader
 // write method copied - no multiple inheritance
   public void send(MsgType msg) {
       out.write(msg);
       out.flush();   // evtl.
   }
}
```

Basic Stream Types in java.io

- Stream Types read/write either byte or char:
- 4 abstract classes

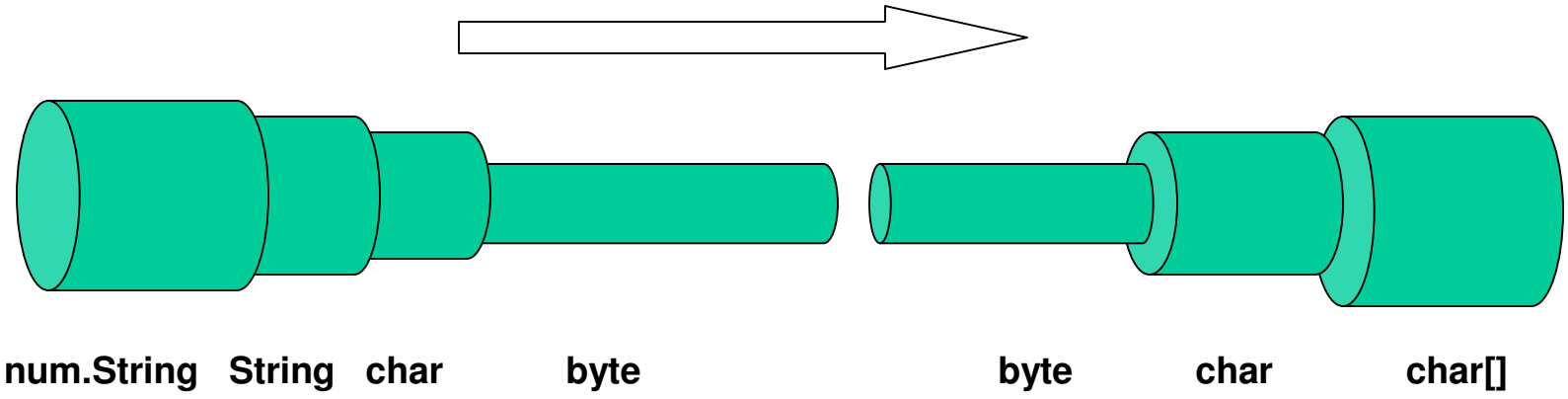
InputStream	**Reader**	read() close()
OutputStream	**Writer**	write() close()
MsgType = byte	MsgType = char	

- ALL methods throw IOException
- Subclasses for special tasks

Stream Transformers

- really "re-encoder"
- How do I want my data to be encoded – how do I understand my encoded data?
- Pattern: "Decoration":

  "adapters" to existing streams



- Pass stream to transformer as constructor argument

Frequently Used Transformers

• InputStreamReader	is a Reader
• OutputStreamWriter	is a Writer
• PrintStream	print(), println() without Exceptions!
• BufferedInputStream / BufferedInputReader	read() for a byte or char array
• BufferedOutputStream / BufferedOutputWriter	write() for a byte or char array
• LineNumberReader	getLineNumber(), setLineNumber()
• DataInputStream / DataOutputStream	readBoolean(), readInt(), readFloat()... writeBoolean(), writeInt() ...
• ObjectInputStream / ObjectOutputStream	is a DataInputStream, readObject() is a DataOutputStream,writeObject()

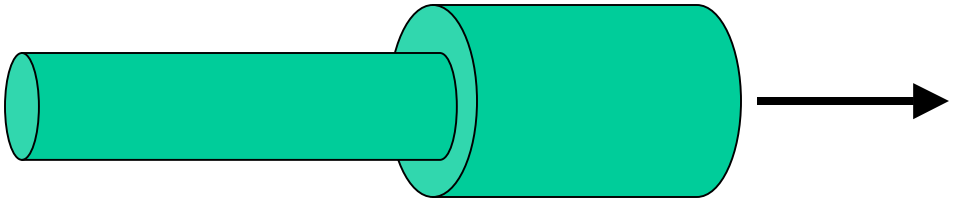
InputStreamReader

Objective:

Read char-s from an existing InputStream

Solution: use an InputStreamReader to transform the InputStream.

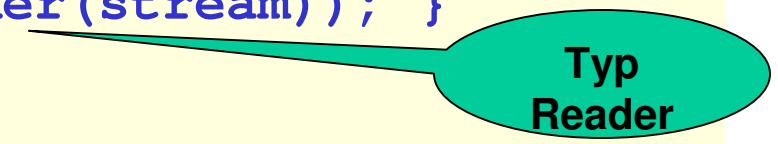
```
InputStream is = System.in;
Reader reader = new InputStreamReader(is);
char unicodeZeichen = reader.read();
```



inputStream InputStreamReader

Example Text Layouter

```
public class TextLayouter extends AnimateObject{
   private Reader reader;
   private String text = "";

   public TextLayouter(Reader reader)
   {   this.reader = reader; }

   public TextLayouter(InputStream stream)
   {  this(new InputStreamReader(stream)); }

   public void act() {
     try { char c = reader.read(); // blocks
           if (c!='\r' & c!='\n') text = text+c;
           else { layoutLine(text); reader.read(); }
     } catch (IOException e) {}
   }
}
```



Linewise Reading Layouter

```
public class TextLayouter extends AnimateObject{
   private BufferedReader reader;
   private String text = "";

   public TextLayouter(BufferedReader reader)
   {   this.reader = reader; }

   public TextLayouter(InputStream stream)
   {  this(new BufferedReader
             (new InputStreamReader(stream))); }

   public void act() {
     try { layoutLine(reader.readLine()); }
     } catch (IOException e) {}
   }
}
```

Object Streams

- What can be sent through a stream:
  - Characters as char or byte using Reader/Writer or InputStream/OutputStream
  - primitive Types using DataInputStream/DataOutputStream
- sending objects?
  - only attributes need to be sent
  - can bedone recursively using DataInputStream/..
- Decomposing an Object (recursively) into a data sequence is called Serialisation

Automatic Object Serialisation

- Transformation using ObjectOutputStream:

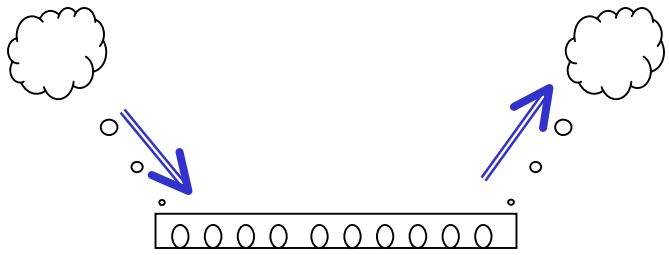
  ```
  new ObjectOutputStream(outputStream)
  ```

- `writeObject(Object o)` writes a serialized object to the outputStream, i.e. its type and version, and its attribute values
- Primitive values are written using DataOutputStream methods: **`writeBoolean()`** etc.
- Objects are written using **`writeObject()`**.
- Requirement: The type must be marked serializable.

  ```
  implements Serializable
  ```

- Serializable an empty, so-called marker interface.

- Deserialisation using ObjectInputStream `readObject() throws InvalidClassException`, (et al.)



Stream connections in Java

Connection between	Connection Type	Implementation
Java objects (Threads)	general channel type	Connection between two streams (Piping)
Java program and external devices	Reading from data sources, writing to data sinks	Console (Keyboard, Screen) File (String) (sources and sinks)
Java programs on different machines		Socket (source and sink)

I.e. there is a single concept behind console io, file io and networking

Console
(Standard Input, Standard Output)

- System has two public static attibutes:
- `public static InputStream in;`
- `public static PrintStream out;`
- **`System.in.read();`**
  reads from an InputStream (attention: IOExceptions)
- **`System.out.println();`**
  writes to a PrintStream (no Exceptions)
- Using `new InputStreamReader(System.in)` yields a char Reader, reading rom the Console.

Finally: File I/O

- java.io.File represents a file.
- Constructors:

```
public File (String path);
public File (URI uri);
```

- File streams are:

```
FileInputStream / Reader
FileOutputStream / Reader
```

- constructor parameters: String or File

```
public FileReader(String path);
public FileReader(File file);
```

- ... that's all...

Application: Persistence

- Task:
  Make TextHistory objects persistent
  (AutoComplete lab)

- TextHistory has an attribute of type ArrayList, which needs to "survive"

Persistence "hand wrought"

- Use fixed persistence file or pass file to constructor
- In the constructor, read lines of data from file and store in ArrayList
- For finalization, use loop to write list entries to file.

```
public class TextHistory {
  private BufferedInputStream in;
  private ArrayList liste = new ArrayList();

  public TextHistory() {
    try {in=new BufferedInputStream
                    (new FileInputStream("data/TH.txt"));
      while (in.available > 0)
         list.enter(in.readLine());
    } catch (IOException e) {} // don't update list...
  }
 // etc....
```

"Automatic" Persistence

- no loops, using writeObject()

```
public class TextHistory {
  private BufferedInputStream in;
  private ArrayList liste = new ArrayList();
  public TextHistory(String datei) {
    try {in=new BufferedInputStream
                  (new FileInputStream(datei));
         liste = in.readObject();
    } catch (IOException e) {}
 }
 // etc....
```

A lot of theory just to do some file i/o...

But the theory covers networking just as well.

Wait for it after the break....