

Persistent Storage

Note Title

Objective: Manage long-term data storage & retrieval

Typical scenario:

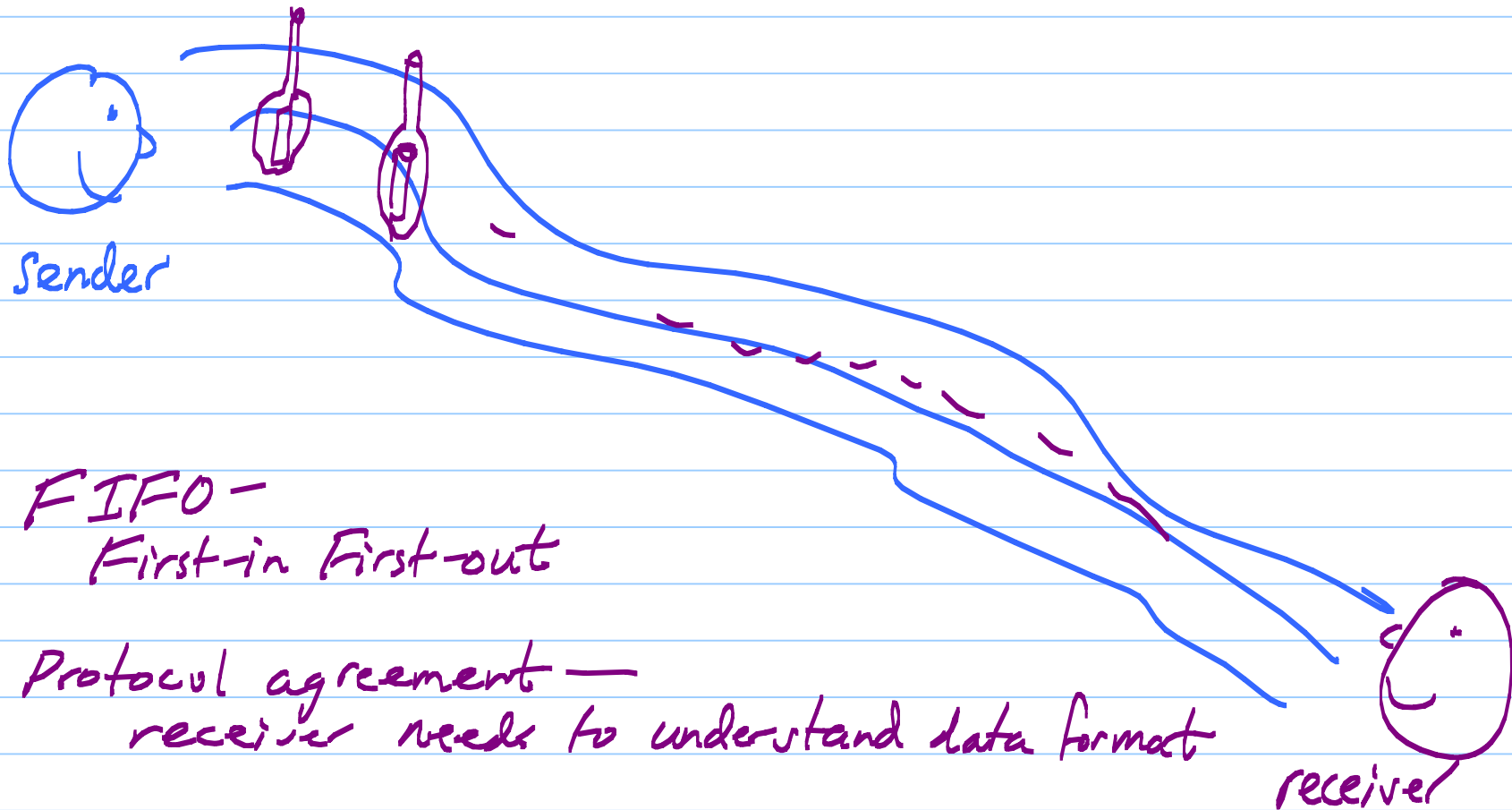
① open a file { sequential access
random access

② read and/or write data — various formats

③ close the file

Generalize: Data may not be coming from/going to a file
(ex. another program, network connection, console...)

Stream Abstraction



Example:

WRITING

```
→ File f = new File(filename);  
→ OutputStream out = new FileOutputStream(f);  
→ DataOutputStream dataOut =  
    new DataOutputStream(out);  
    dataOut.writeInt(myNumber);  
    dataOut.writeChars(myString);  
    out.close();
```

READING

```
File f = new File(filename);  
InputStream in = new FileInputStream(f);  
DataInputStream dataIn =  
    new DataInputStream(in);  
myNumber = dataIn.readInt();  
myString = dataIn.readChars();  
in.close();
```

The java.io library supports orthogonal choice of:

- source/destination (file, console, socket, URLconn., pipe)

- management (sequential, line numbered, buffered, ...)

- format (characters, bytes, binary data, objects, XML, zip)

Accomplished through streams that wrap other streams...

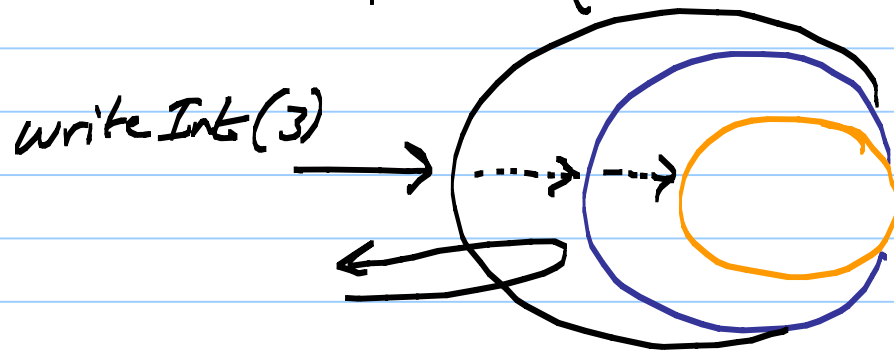
How streams can wrap other streams

- ① A stream can take another stream as a parameter to its constructor
- ② The outer stream delegates to the wrapped one

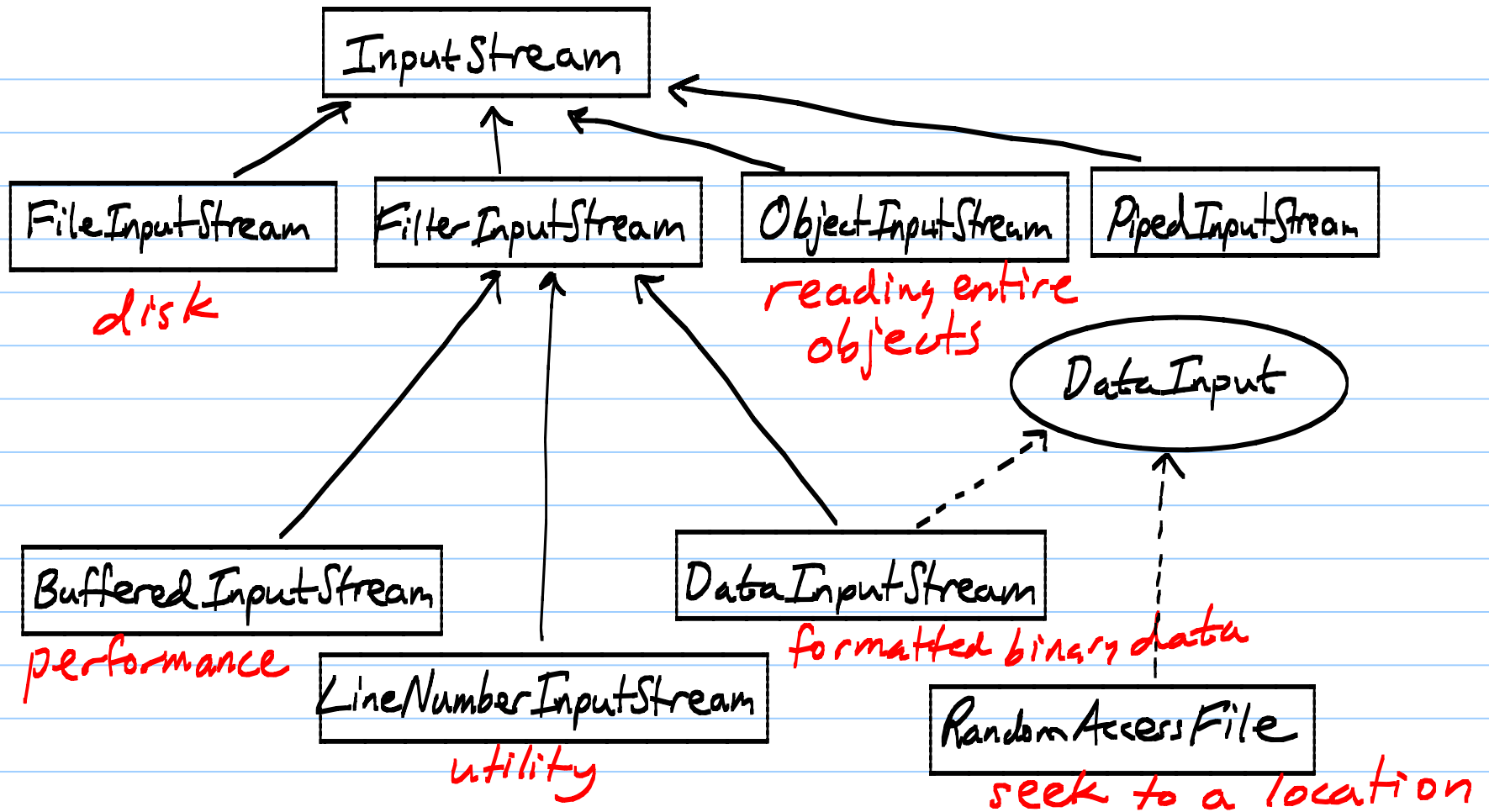
Example:

DataOutputStream out =

```
new DataOutputStream(new BufferedOutputStream(new FileOutputStream(".")));
```



package java.io



(Similar hierarchies for output streams, readers, and writers)

Design Decision: Data Formats and Management

	Human readable/modifiable can manually read/edit in files	Machine readable only can be more compact & faster
Customized programmer defines what is saved & how it is saved		
Standardized provided algorithms automatically save & load data		
Managed separate subsystem maintains the data		

Design Decision: Data Formats and Management

	Human readable/modifiable can manually read/edit in files	Machine readable only can be more compact & faster
Customized programmer defines what is saved & how it is saved	[character files in a customized format Ex: PrintStream (out) Scanner (in)]	application uses system-provided formats to read/write data Ex: DataOutputStream, DataInputStream
Standardized provided algorithms automatically save & load data	high-level semantics is captured in a standard hierarchical format Ex: XML encoder XML decoder	[low-level traversal of & restoration of heap structures Ex: Java serialization w/ ObjectOutputStream & ObjectInputStream]
Managed separate subsystem maintains the data	data model & access available in database UI Ex: relational database, JDBC	database system manages app's persistent heap objects Ex: JDO

Parsing regular expressions

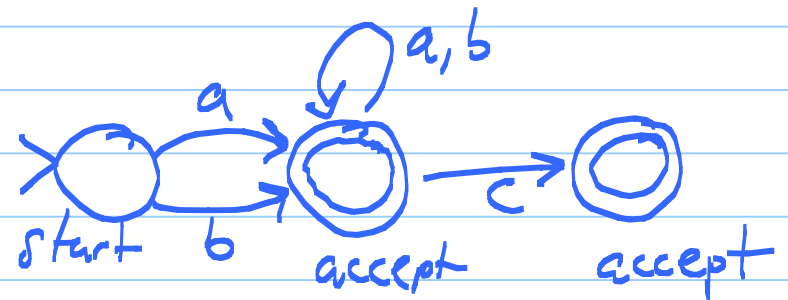
Alphabet $a-z, ., ..$

ab

$(a|b)c$ \leftarrow zero or more $ac \quad bc$

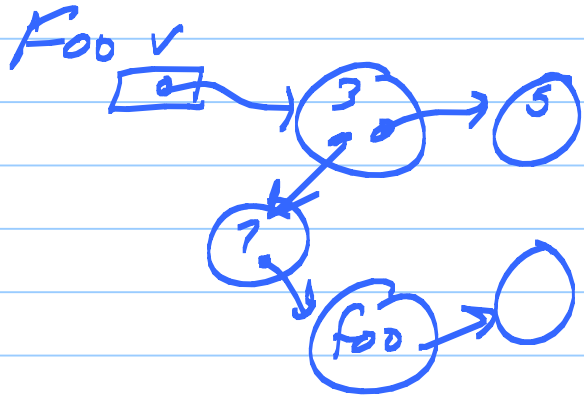
$(a|b)^*c$ $abac \quad bbbc \quad c$

$(a|b)^+c$ \leftarrow 1 or more



Supported by Java's scanner class

low-level traversal of +
restoration of heap structures



Foo implements Serializable

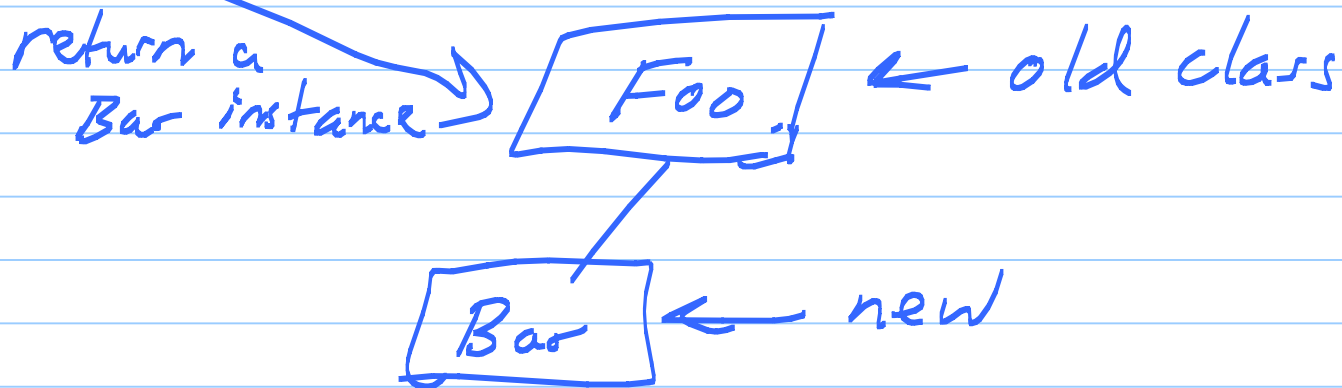
no methods!
(marker interface)

Ex: Java serialization w/
ObjectOutputStream + ObjectInputStream

```
File f = ...  
FileOutputStream fos =  
    new FileOutputStream(f);  
ObjectOutputStream oos =  
    new ObjectOutputStream(fos);  
oos.writeObject(v);  
oos.close();  
oos.close();  
v = (Foo) ois.readObject();
```

readResolve — Upgrades

add this method to a class if you want to replace the obj. on load



writeReplace — "disk references"