תוכנה 1

תרגול 9: קלט / פלט

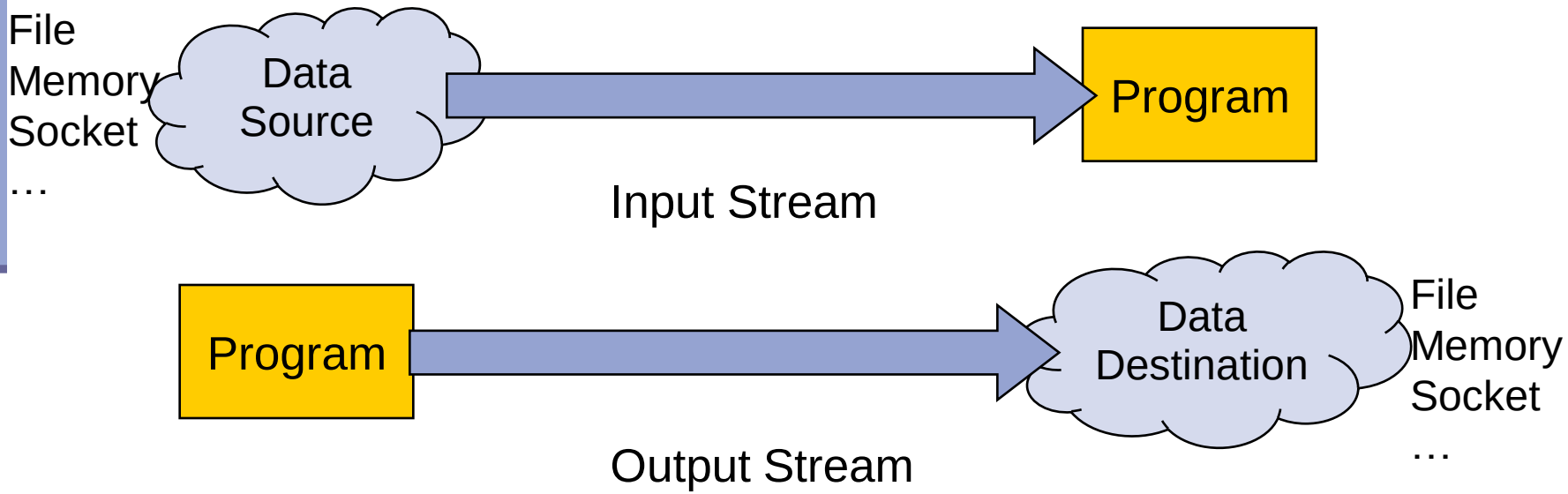
נעמה מאיר ומתי שמרת

I/O Streams

- An ***I/O Stream*** represents an input source or an output destination
  - disk files, network, other programs etc.
- Simple model: a sequence of data
- All kind of data, from primitive values to complex objects

I/O Streams

- Streams are one-way streets
  - ***Input*** streams for reading
  - ***Output*** streams for writing



Streams Usage Pattern

- Usage Flow:

```
create a stream
while more data
    Read/Write data
close the stream
```

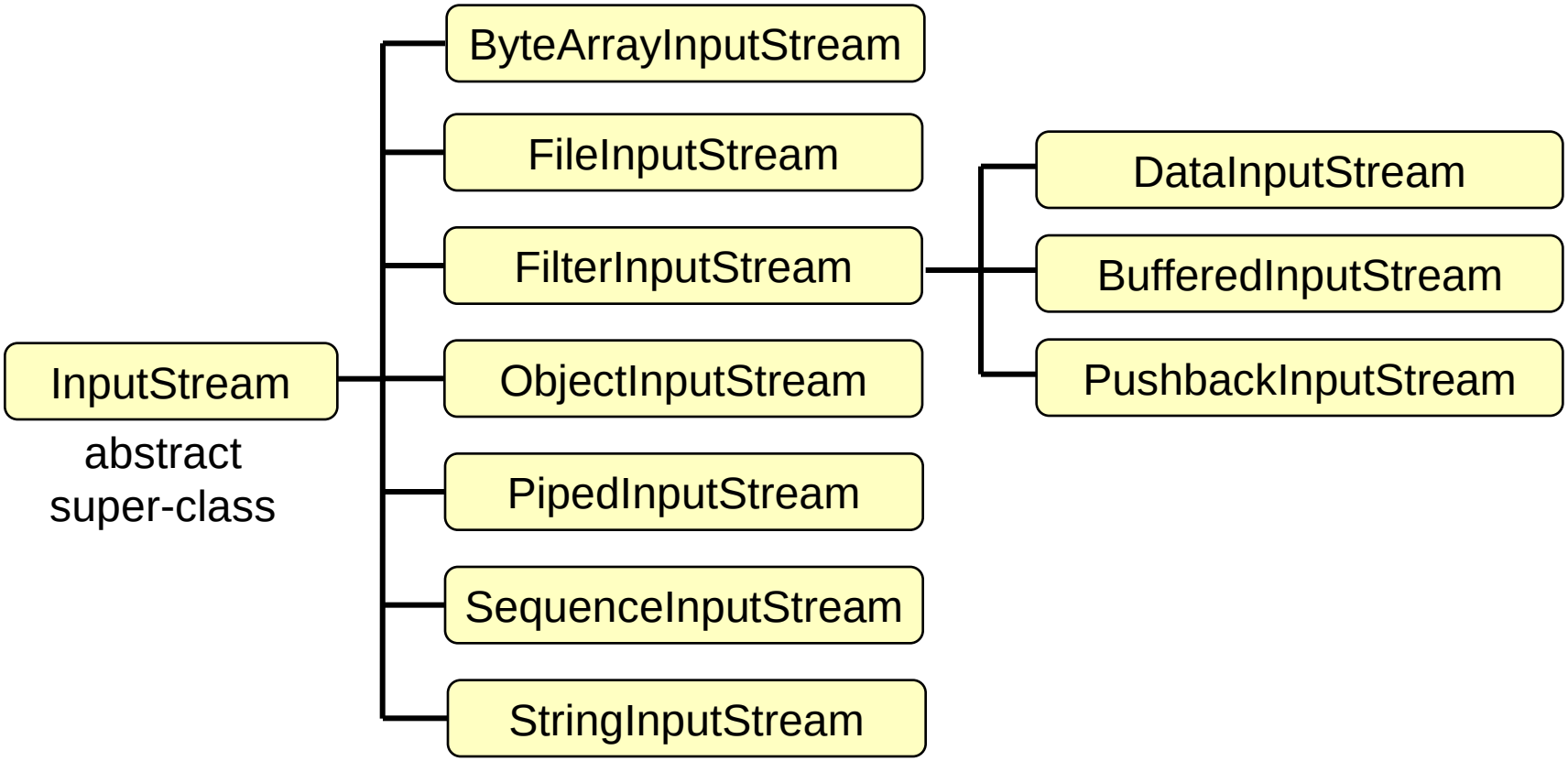
- All streams are automatically opened when created

Streams

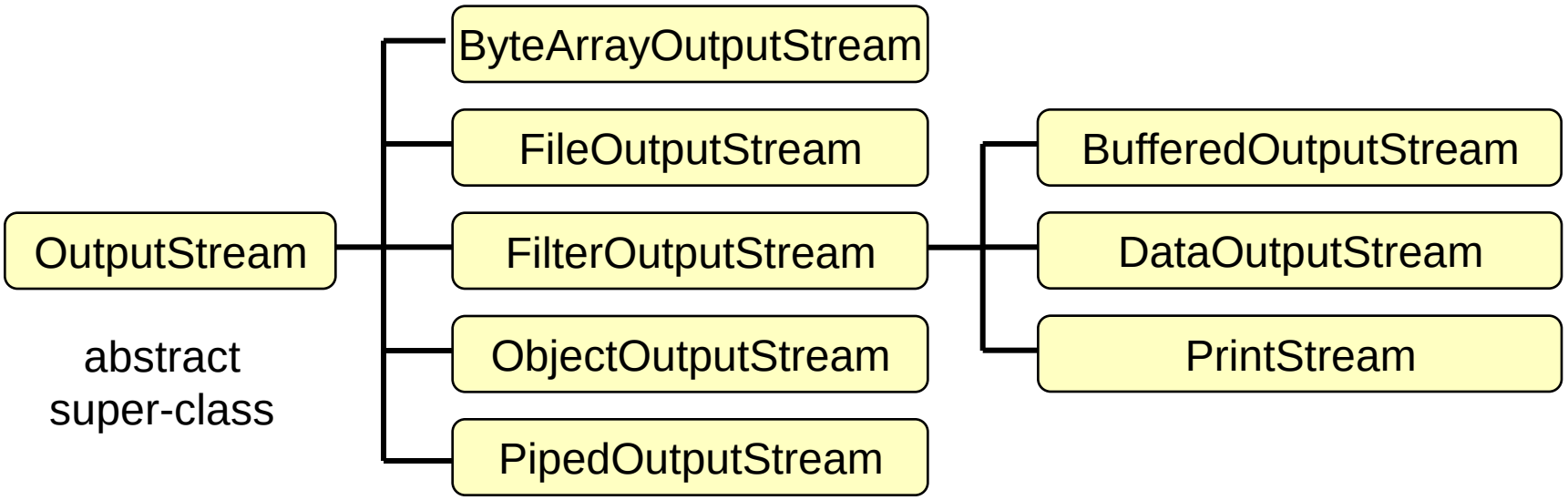
- There are two categories of streams:
  - ***Byte streams*** for reading/writing binary data
  - ***Character streams*** for reading/writing text
- Suffix Convention:

direction \ category	Byte	Character
Input	InputStream	Reader
Output	OutputStream	Writer

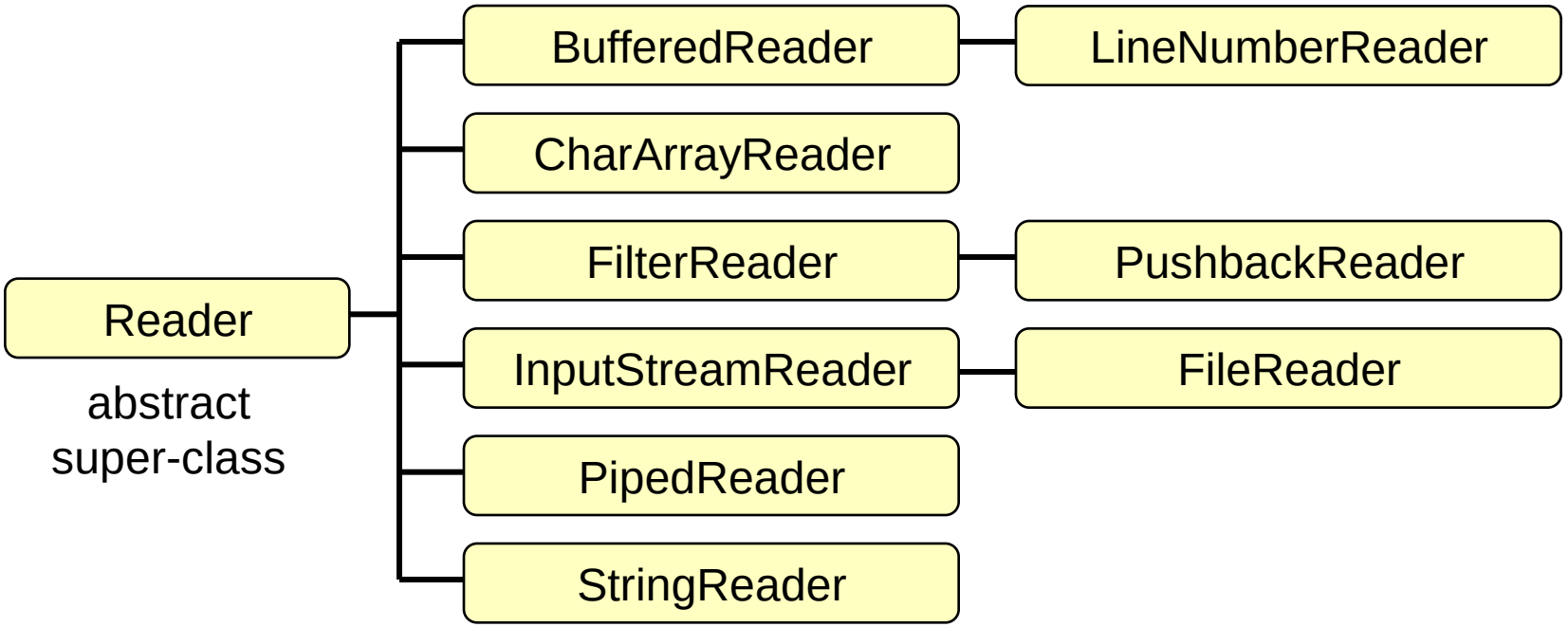
InputStreams Hierarchy



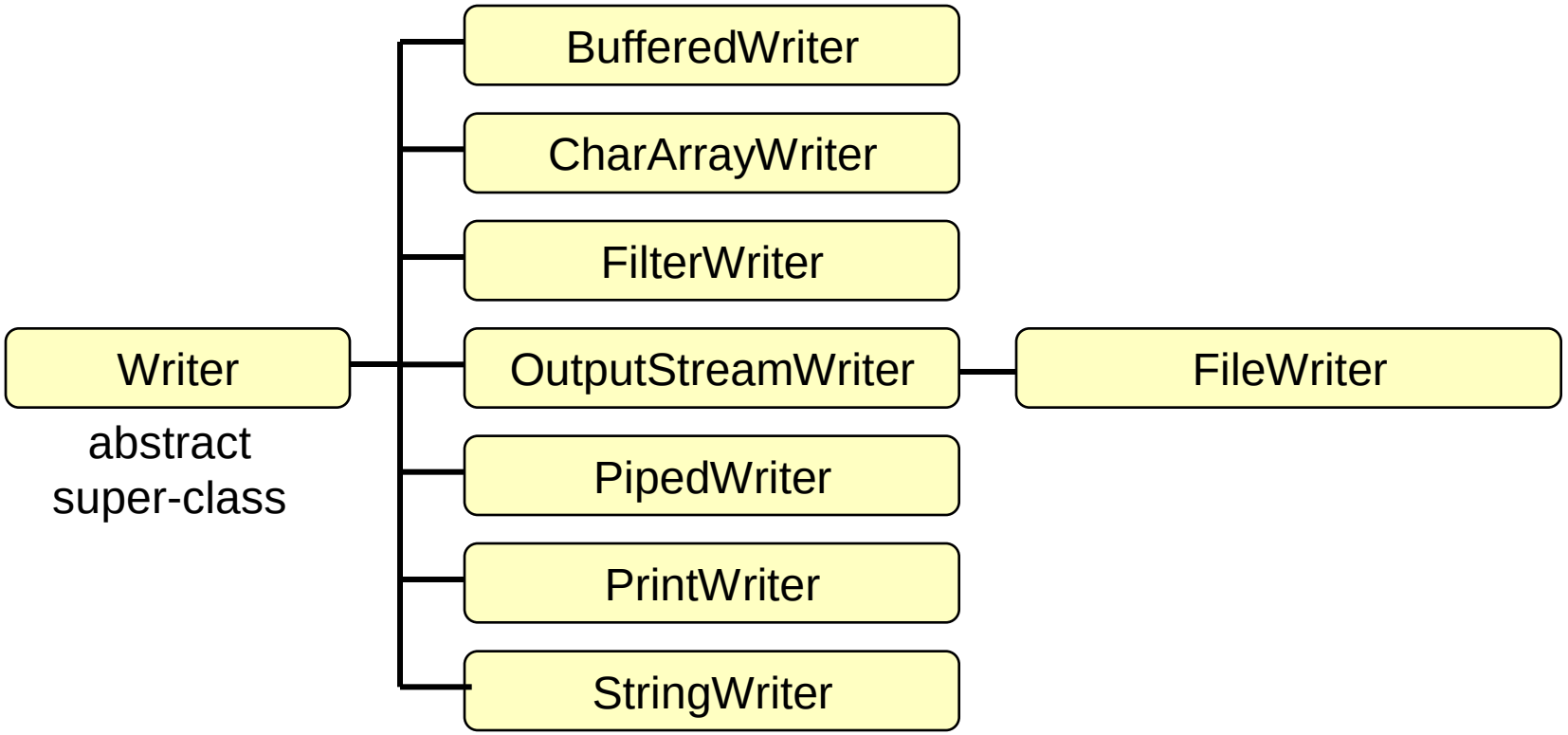
OutputStreams Hierarchy



Readers Hierarchy



Writers Hierarchy



The java.io package

- The java.io package provides:
  - Classes for reading input
  - Classes for writing output
  - Classes for manipulating files
  - Classes for serializing objects

Terminal I/O

- The System class provides references to the standard input, output and error streams:

Field Summary	
static PrintStream	**err** The "standard" error output stream.
static InputStream	**in** The "standard" input stream.
static PrintStream	**out** The "standard" output stream.

InputStream Example

- Reading a single byte from the standard input stream:

```
try {
    int value = System.in.read();
    if (value != -1) {
        ...
    }
    ...
} catch (IOException e) {
    ...
}
```

an int with a byte information

returns -1 if a normal end of stream has been reached

is thrown in case of an error

Character Stream Example

```
public static void main(String[] args) {

    try {
      FileReader in = new FileReader("in.txt");
      FileWriter out = new FileWriter("out.txt");

      int c;
      while ((c = in.read()) != -1) {
          out.write(c);
      }

      in.close();
      out.close();

    } catch (IOException e) {
      // Do something
    }
}
```

Create streams

Copy input to output

Close streams

Stream Wrappers

- Some streams wrap others streams and add new features.
- A wrapper stream accepts another stream in its constructor:

```
DataInputStream din =
    new DataInputStream(System.in);
double d = din.readDouble();
```

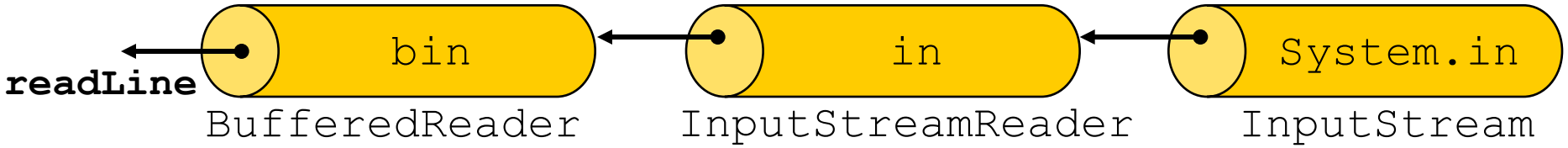
Stream Wrappers Example

- Reading a text string from the standard input:

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

    BufferedReader bin = new BufferedReader(in);

    String text = bin.readLine();
    ...
} catch (IOException e) {...}
```



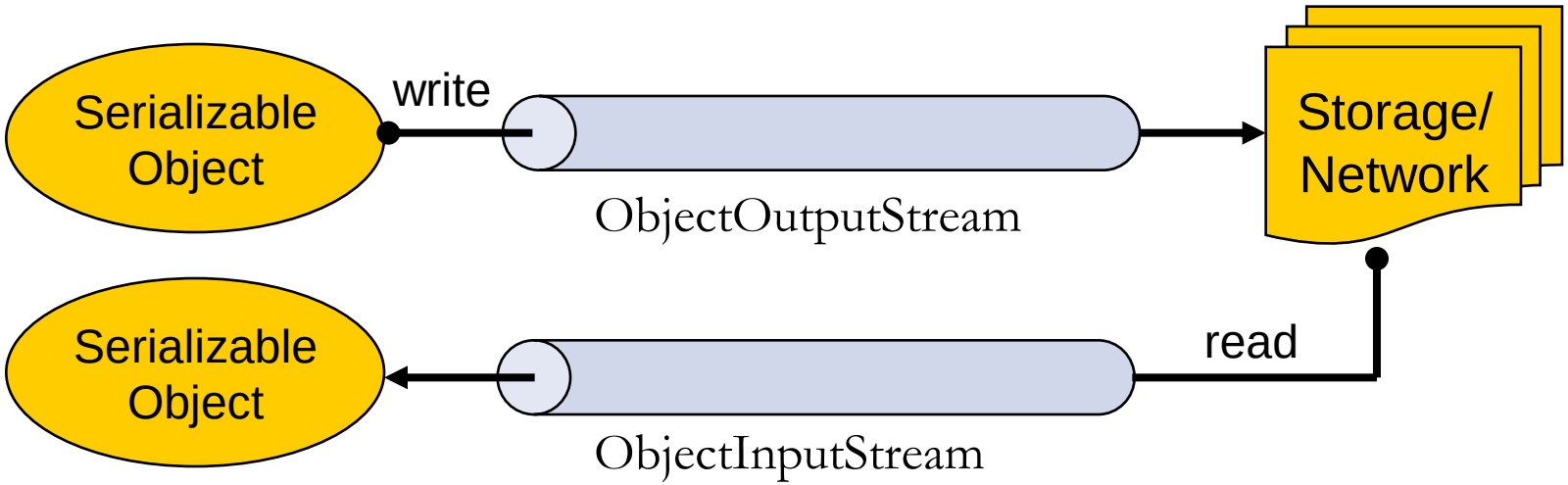
Object Serialization

- A mechanism that enable objects to be:
  - saved and restored from byte streams
  - persistent (outlive the current process)

- Useful for:
  - persistent storage
  - sending an object to a remote computer

The Default Mechanism

- The default mechanism includes:
  - The Serializable interface
  - The ObjectOutputStream
  - The ObjectInputStream



The Serializable Interface

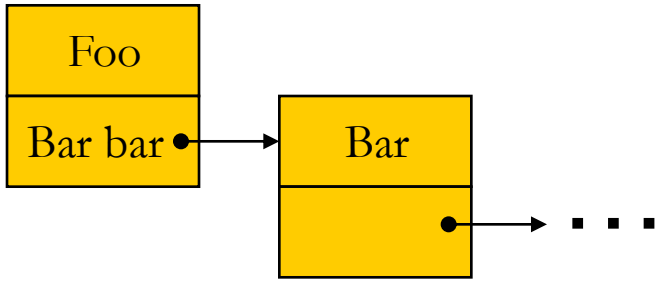
- Objects to be serialized must implement the java.io.Serializable interface
- An empty interface
- Most objects are Serializable:
  - Primitives, Strings, GUI components etc.
- Subclasses of Serializable classes are also Serializable

Recursive Serialization

- Can we serialize a Foo object?

```
public class Foo implements Serializable {
     private transient Bar bar;
     …
}

public class Bar implements Serializable {…}
```



- No, since Bar is not Serializable
- Solutions:
  1. Implement Bar as Serializable
  2. Mark the bar field of Foo as transient

HashMap Serialization

```
Map<Integer, String> map = new HashMap<…>();
...
try {

    FileOutputStream fileOut =
            new FileOutputStream("map.s");

    ObjectOutputStream out =
            new ObjectOutputStream(fileOut);

    out.writeObject(map);

} catch (Exception e) {...}
```

* HashMap is Serializable, so are all the other concrete collection types we've seen

Reading Objects

```
try {

   FileInputStream fileIn =
                  new FileInputStream("map.s");

   ObjectInputStream in =
                  new ObjectInputStream(fileIn);

   Map<Integer, String> map =
     (Map<Integer, Sring>)in.readObject();

} catch (Exception e) {...}
```

The File Class

- A utility class for file or directory properties (name, path, permissions, etc.)
- Performs basic file system operations:
  - removes a file: `delete()`
  - creates a new directory: `mkdir()`
  - checks if the file is writable: `canWrite()`
  - creates a new file: `createNewFile()`
- No direct access to file data
  - Use file streams for reading and writing

The File Class

Constructors

- Using a full pathname:

```
File f = new File("/doc/foo.txt");
File dir = new File("/doc/tmp");
```

- Using a pathname relative to the current directory defined in `user.dir`:

```
File f = new File("foo.txt");
```

Note: Use `System.getProperty("user.dir")` to get the value of `user.dir`

(Usually the default is the current directory of the interpreter. In Eclipse it is the project's directory)

The File Class

Constructors (cont)

- `File f = new File("/doc","foo.txt");`

- `File dir = new File("/doc");`
  `File f = new File(dir, "foo.txt");`

- A `File` object can be created for a non-existing file or directory
  - Use `exists()` to check if the file/dir exists

The File Class

Pathnames

- Pathnames are system-dependent
  - `"/doc/foo.txt"` (UNIX format)
  - `"D:\\doc\\foo.txt"` (Windows format)
- On Windows platform Java excepts path names either with `'/'` or `'\'`
- The system file separator is defined in:
  - `File.separator`
  - `File.separatorChar`

The File Class
Directory Listing

- Printing all files and directories under a given directory:

```
public static void main(String[] args) {
    File dir = new File(args[0]);

    for (String file : dir.list()) {
        System.out.println(file);
    }
}
```

The File Class
Directory Listing (cont.)

- Printing only files with "`.txt`" suffix:

```
public static void main(String[] args) {
    File dir = new File(args[0]);
    FilenameFilter filter = new
                SuffixFileFilter(".txt");

    for (String file : dir.list(filter)) {
         System.out.println(file);
    }
}
```

The File Class
Directory Listing (cont.)

```
public class SuffixFileFilter
                    implements FilenameFilter {

   private String suffix;

   public SuffixFileFilter(String suffix) {
      this.suffix = suffix;
   }

   public boolean accept(File file, String name) {
      return name.endsWith(suffix);
   }
}
```