# Object Serialization

Using Byte Streams

Week 8 Presentation 4

InputStream and OutputStream are the parent classes for all byte streams.

# Byte Streams

Byte streams are not the most used and there are libraries specifically for multimedia…

Remember JavaFx: when you create a new Image or Media object, a binary file is read into memory by the constructor. There is necessarily a byte stream behind the scene, but it's all done by the constructor.

See also the docs: https://docs.oracle.com/javase/tutorial/essential/io/bytestreams.html

# ByteStream (unbuffered)

```
    FileInputStream in = null;
    FileOutputStream out = null;

    try {
        in = new FileinputStream("filename");
        out = new FileOutputStream("filename");


        } catch (...)  {

        } finally {
            ...
    }
}
```

The examples in the previous presentation were for byte streams.

# ByteStream (buffered)

```
    BufferedInputStream in = null;
    BufferedOutputStream out = null;

    try {
        in = new BufferedinputStream(new FileInputStream("filename"));
        out = new BufferedOutputStream( ... );


        } catch (...)  {

        } finally {
            ...
   }
}
```

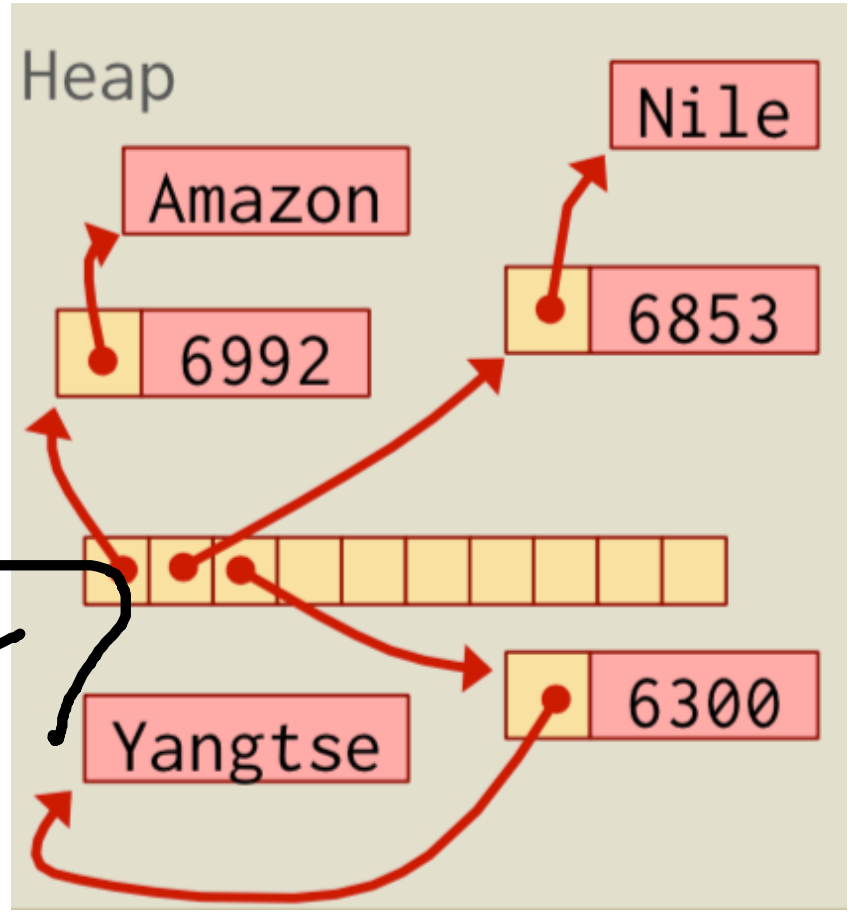
… including the buffered version.


Terminology: when you are referring to “bytes” in I/O operations you are really dealing with int variables. Not byte variables. For historical reasons they are called byte streams.

Don't be misled by the "byte" in "byte stream". Operations deal with more than one byte.

# Object Serialization

```
class Obj2 {
    private String name;
    private int value;
    ...

}

class Obj1 {
    private Obj2[] o = null;
    private int count = 0;

    public Obj1() {
        o = new Obj2[10];
    }

    ...

}
```



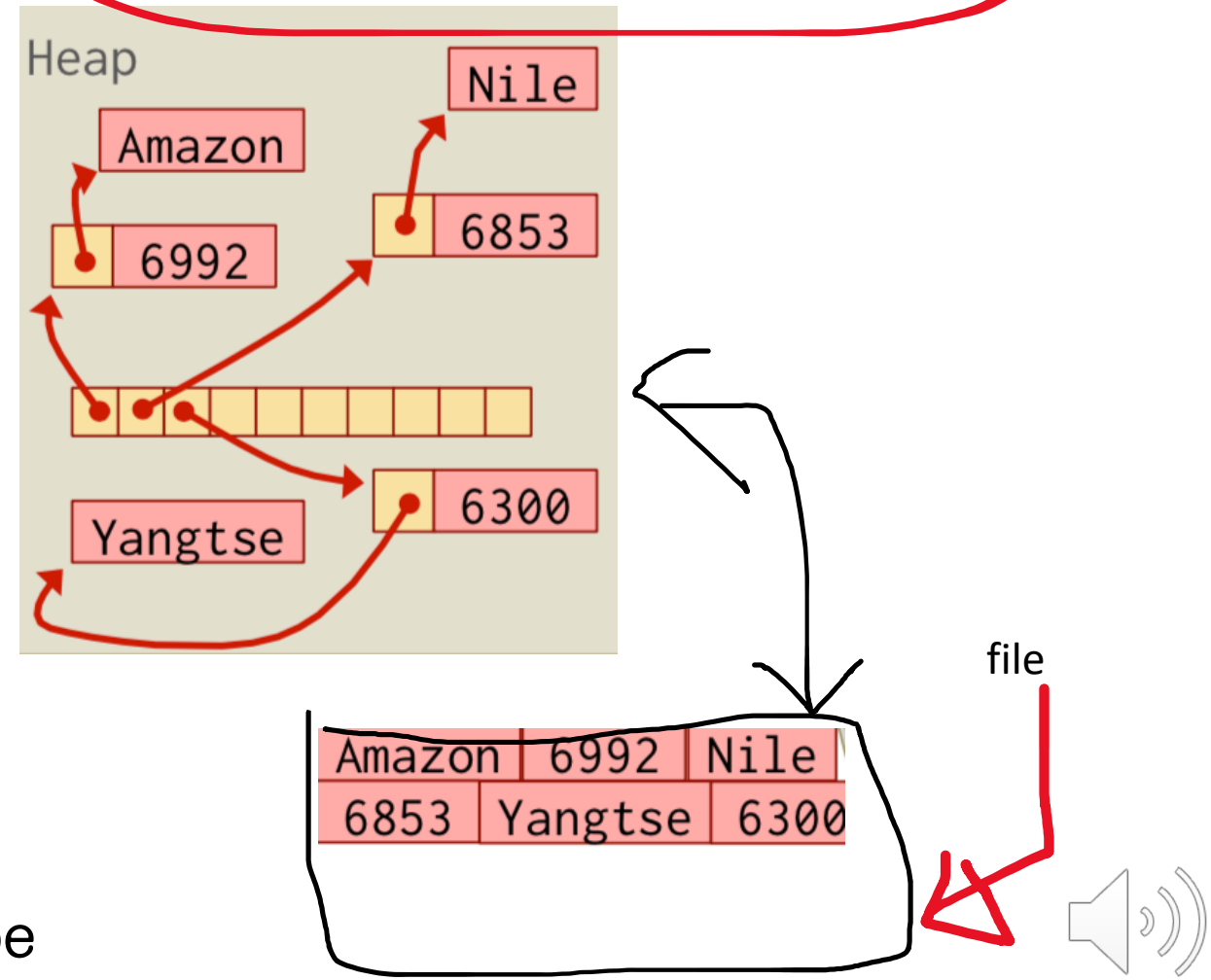

One application of byte streams is "serialization", dumping a memory object to file...

# Object Serialization

```
class Obj2 {
    private String name;
    private int value;
    ...

}

class Obj1 {
    private Obj2[] o = null;
    private int count = 0;

    public Obj1() {
        o = new Obj2[10];
    }

    ...
}
```

You can also declare attributes ”transient” meaning they shouldn’t be dumped.

When doing so you want to store the data – **not** the memory addresses that change when you reload



# Object Serialization: requisites

(What you need to do to use it)

```
class Obj2 implements java.io.serializable {
        private String name;
        private int value;
…
}

class Obj1 implements java.io.serializable{
        private Obj2[] o = null;
        private int count = 0;

        public Obj1() {
                o = new Obj2[10];
        }}
```

## 1. Interface

No method!
Its just a declaration and there is no method to implement. It is just to tell javac to generate necessary requirements to save the data to disk.

# Object Serialization: requisites

```
class Obj2 implements java.io.serializable {
        private String name;
        private int value;
…
}

class Obj1 implements java.io.serializable{
        private Obj2[] o = null;
        private int count = 0;

        public Obj1() {
                o = new Obj2[10];
        }}
```

**2. Constructor**

# Object Serialization: requisites

And you need an "ObjectOutputStream" that is a special flavor of byte stream. This one comes by default with a buffer.

```
FileOutputStream fileOut = new FileOutputStream("file.dat");
ObjectOutputStream out = new ObjectOutputStream(fileOut);
out.writeObject(o);
out.close();
fileOut.close();
```

**3. Stream**

has a buffer included

A file written on one computer can be read on any computer

Next: using character streams and databases to save program state