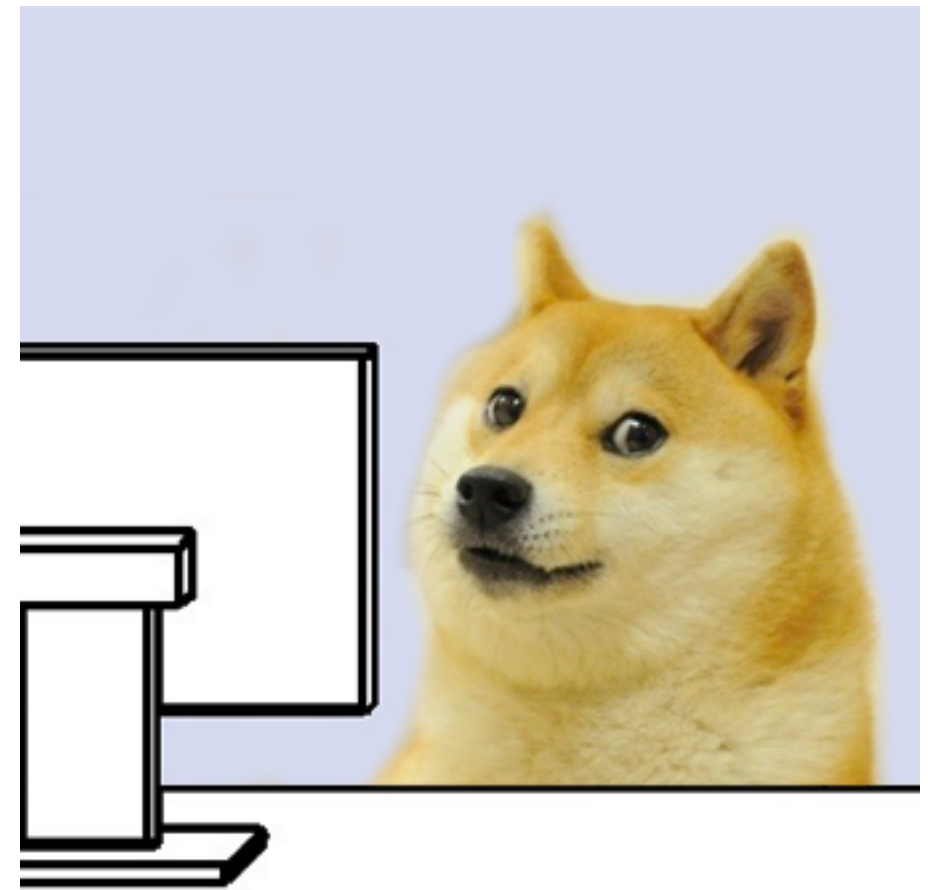


how 2 code

discussion 5



Hex

Which is easier to read?

1111111111011000 1111111111100000
11111111111011000 11111111111100001

or...

ffd8 ffe0
ffd8 ffe1

- Hex is just a way of displaying stuff
- Under the hood, it's all the same: **1**s and **0**s

Hex <-> binary

Just go 4 bits at a time

1111 1111 1101 1000

(this step may need explaining)
(but for HW5, you don't need to convert it manually)

15 15 13 8

f f d 8

0..9 = 0..9

10=a

11=b

12=c

13=d

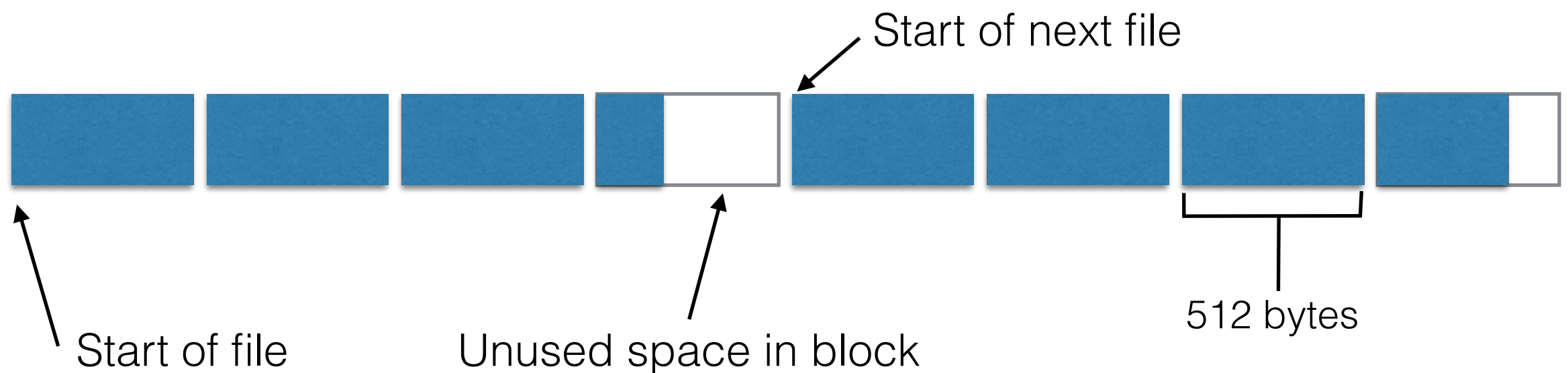
14=e

15=f

FAT Blocks

- All files are block aligned
- Each block contains a piece of exactly 1 file

Example with 2 files: 1705 bytes and 1874 bytes



HW5

- Open card.raw with `FileInputStream`
`FileInputStream f = new FileInputStream(...);`
- Give that to `BufferedInputStream`, e.g.
`BufferedInputStream b_in = new BufferedInputStream(f);`
- Use `read()` and your loop skills to find headers
- Get all the data from the start of one header to the start of the next
- Use `FileOutputStream` and `BufferedOutputStream` to dump this to `<some file>.jpg`

Useful things

- This overloaded version of `read()` in `BufferedInputStream`:

```
read(byte[] b)
```

Reads up to `byte.length` bytes of data from this input stream into an array of bytes.

- This method is 'inherited', easy to miss it in the docs:

Methods inherited from class `java.io.FilterInputStream`

`read`

- If you have a `byte[]` array to hold 1 block, `read()` can automatically fill it up
- Simply work on 1 block at a time; no need to scan individual bytes

Annoying things

- **Always** `close()` your outputs!
- Declaring byte arrays in Java:

```
Main.java:25: error: possible loss of precision
        byte[] header = {0xff, 0xd8, 0xff, 0xe1};
                        ^
required: byte
found:    int
```

wtf... `0xff` is clearly one byte

- You have use a cast:

```
byte[] stfu_java = {(byte)0xff, ...}
```

The Signed Byte®

- A byte in java is *signed*
- Instead of displaying a byte as 0..255, Java decided to display them like this:

Normal: 0,1,2,3,...,124,125,126,127, 128, 129, 130, 131,...,255

Java: 0,1,2,3,...,124,125,126,127, -128, -127, -126, -125,..., -1

- Try this:
`byte b = (byte)255;`
`System.out.println(b);`
- It prints 255 right?
- Nope...-1
- But don't worry, it only affects what you see.
- It doesn't change the underlying bits

The error, explained

```
Main.java:25: error: possible loss of precision
        byte[] header = {0xff, 0xd8, 0xff, 0xe1};
                        ^
required: byte
found:    int
```

- Java expects a byte to be -128..127
- You give it 255, which is out of range
- It needs more than a byte to store 255, so it chooses the next best thing: `int`
- Thus: required: byte, found: int

Questions