

# Opening Exercise

Suppose that you are given three integers in `int` variables.

*Describe a way to encode their values  
into a single `int` value.*

Now suppose that all three integers have values  $\leq 256$ .

*How can you encode their values  
into a single `int` value  
without losing any information?*

# Possible Solutions, Part 1

add the values  
choose the largest  
average the values

The last of these corresponds to **grayscale** in an image.  
Is it a lossy or lossless encoding?

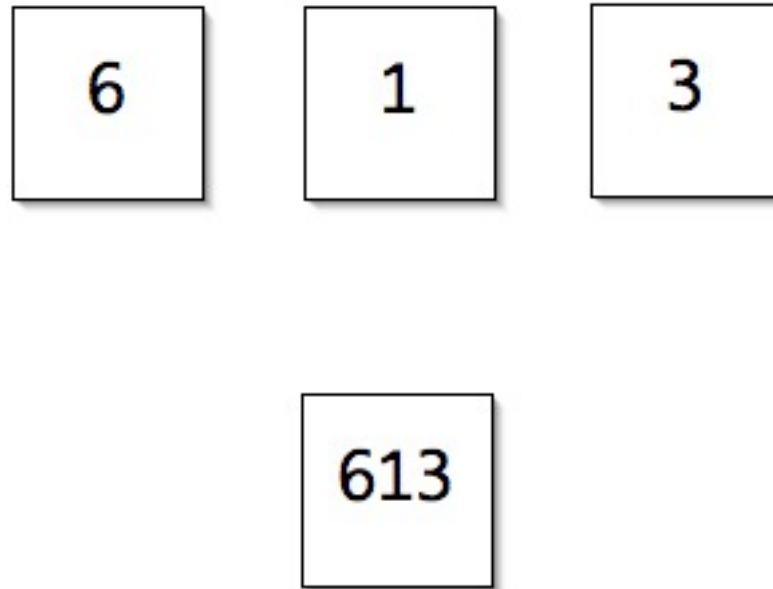
## Possible Solutions, Part 2

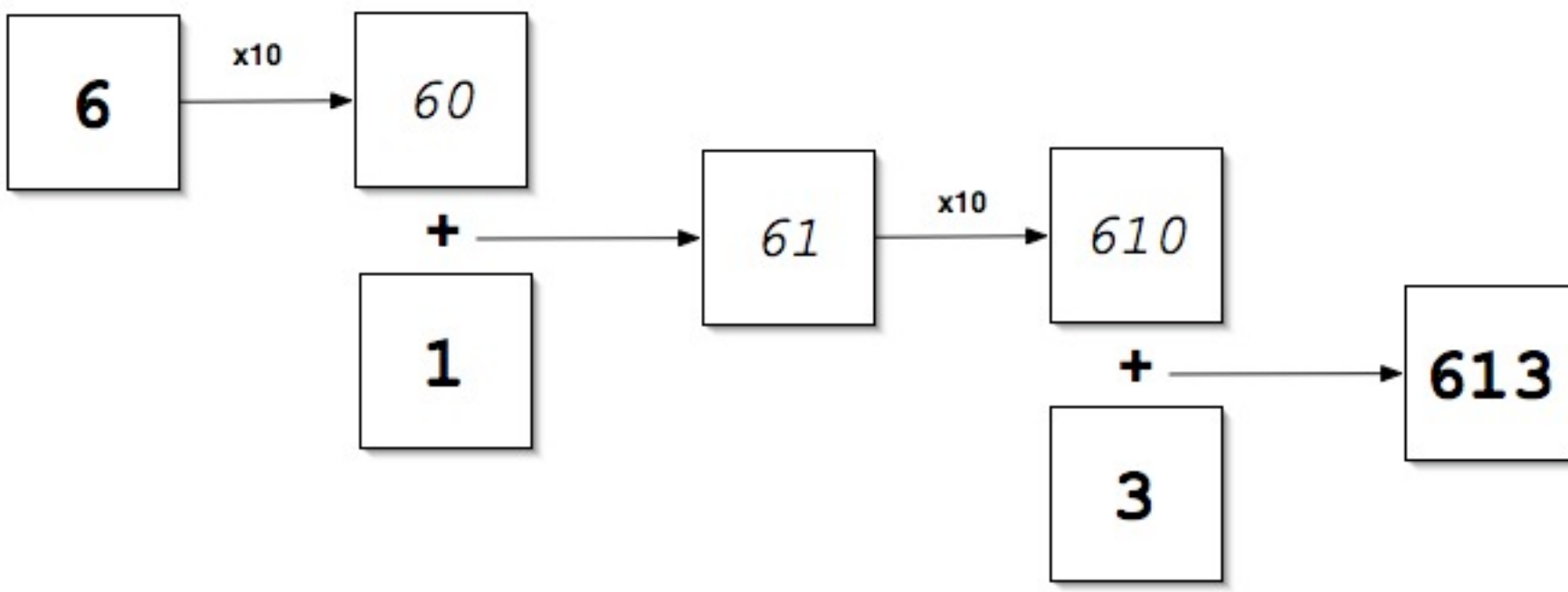
```
> Integer.MAX_VALUE  
2147483647  
> 256 * 256 * 256  
16777216
```

An integer consists of 32 bits,  
and each of our values takes at most 8 bits.

How can we use the empty bits?

## Part 2 — With 1-Digit Values





## Exercise #2

Write a `Pixel` method  
named `setColorFrom( int encoded )`  
that changes the pixel's color  
to the RGB values encoded  
in the `int` argument.

Methods you can use in `Pixel`:

```
getRed(), getGreen(), getBlue()  
setRed(), setGreen(), setBlue()  
getColor(), setColor()
```

# Our Picture Compression

Success!

For a Picture of  $n$  pixels,  
we now require only  $n$  integers,  
not  $3n$  integers.

... but.

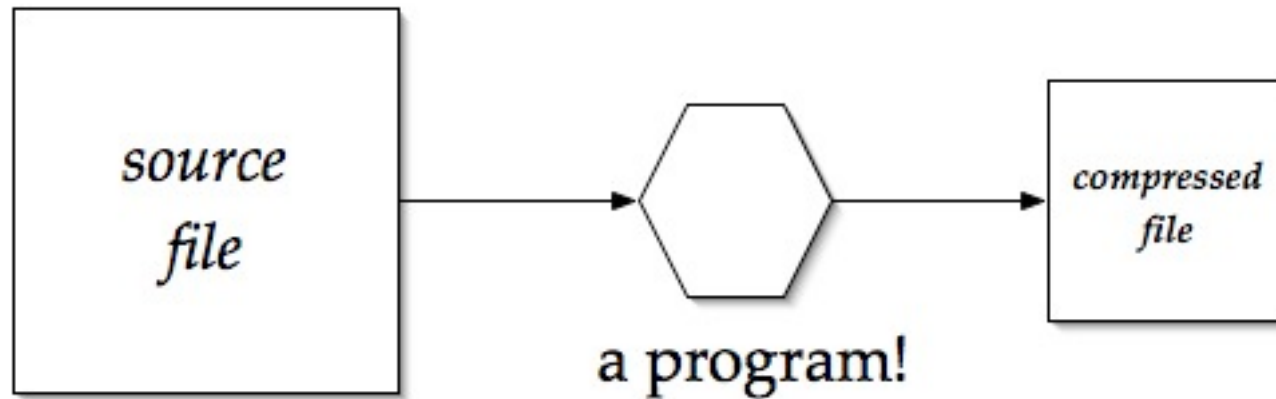
# But What?

That is already how Pixels are represented!

The methods  
getRed(),  
getGreen(),  
getBlue()

don't look up the values of variables.  
They *compute* the values upon request.

# Recap: Data Compression



Often we can find a way to store the same amount of information in less space — **with a different encoding.**

# Recap: Lossy versus Lossless

A compression algorithm that loses information is called **lossy**.

A compression algorithm that retains all information is called **lossless**.

Lossy algorithms can generate smaller files, at some cost in the quality of the file for some purposes — including decompression.

# Our Idea for Compressing Sound

Instead of using 2 bytes for each sample value, we could use 1 byte to record each sample change!

42	-21	14	42	6	-57
----	-----	----	----	---	-----

42	-63	35	25	-36	-63
----	-----	----	----	-----	-----

For a Sound of size  $n$ ,  
we now require  $n+1$  bytes  
instead of  $2n$  bytes.

# Our DiffSound Class

class declaration  
instance variables  
constructors  
methods

access modifiers  
public versus private  
static versus (not)

# Exercise: Decompression

Write a `DiffSound` method  
named `decompress()`  
that returns a `Sound` object.

The returned `Sound` should reconstruct  
the original set of sample values.

# Javadoc as a Tool

At a command-line prompt:

```
mac os x > javadoc DiffSound.java
```

generates the file: DiffSound.html

```
mac os x > javadoc *.java
```

generates hyperlinked documentation  
for all the Java source files in the current directory.