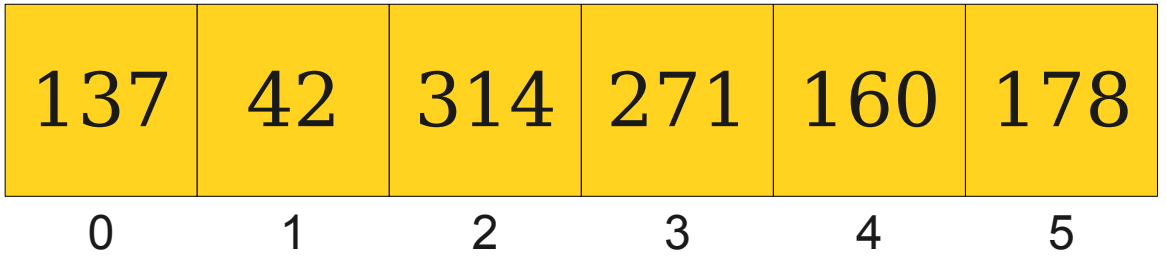
# Arrays

# A Different Way to Store Data

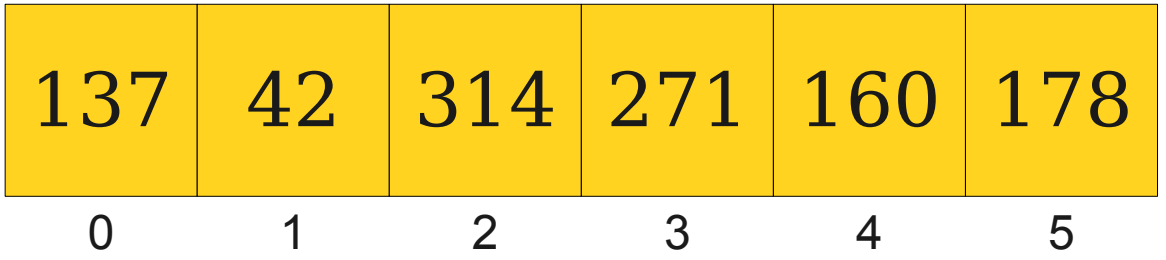
- On Monday, we saw the **`ArrayList`** as a way to store lots of data.
  - Lines of text.
  - US cities!
- Java also supports a concept called the **array** that can used to store lots of data.

# Recapping `ArrayList`




- An **`ArrayList`** stores a **sequence** of multiple objects.
  - Can access objects by index by calling **`get`**.
- All stored objects have the same type.
  - You get to choose the type!
- Must store objects; primitive types not allowed.
- Can grow as long as it needs.

# Introducing Arrays




- An array stores a **sequence** of multiple objects.
  - Can access objects by index using square brackets (more on that soon).
- All stored objects have the same type.
  - You get to choose the type!
- Can store *any* type, even primitive types.
- Size is fixed; cannot grow once created.

# Default Values in Arrays

- Because arrays have a fixed size, when declaring an array, all values in that array will initially be set to a default value:
  - `int`, `double`, etc. default to 0,
  - `boolean` defaults to `false`, and
  - Objects default to `null`.

# Basic Array Operations

- To create a new array, specify the type of the array and the size in the call to **`new`**:

```
Type[] arr = new Type[size]
```

- To access an element of the array, use the square brackets to choose the index:

```
arr[index]
```

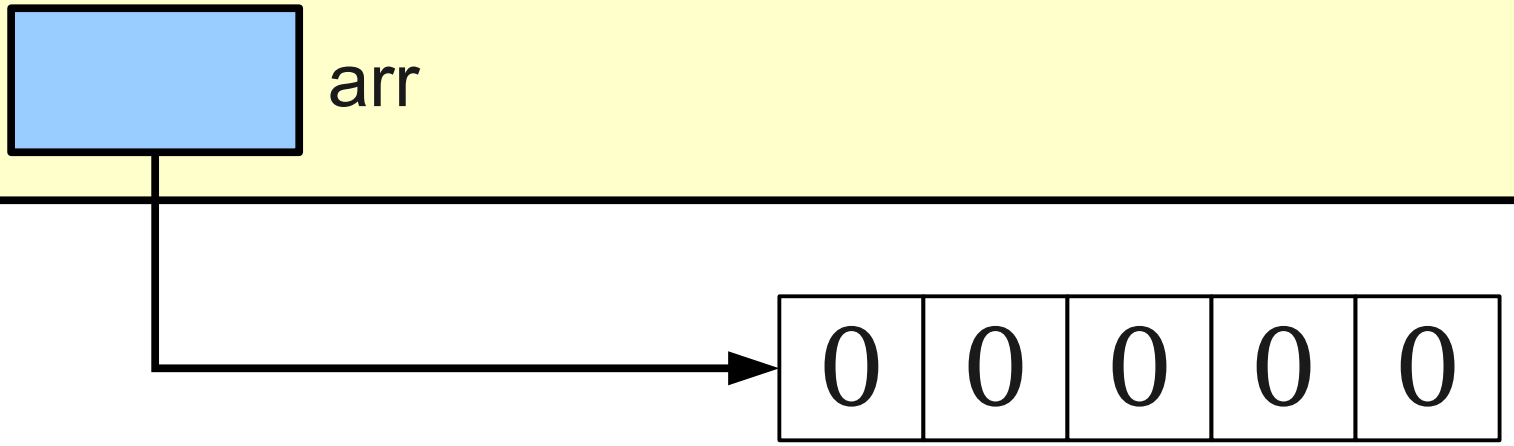
- To read the length of an array, you can read the **`length`** field (without parentheses):

```
arr.length
```

# Arrays as Parameters

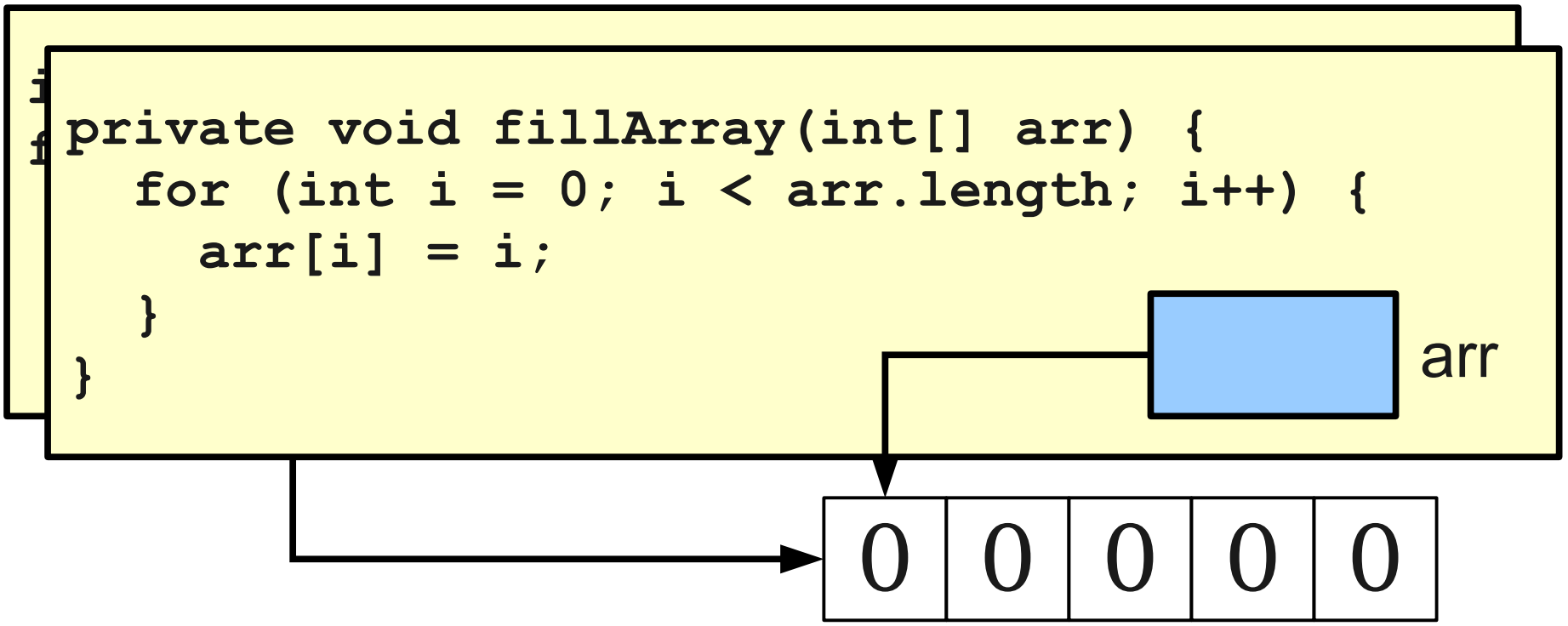
- Arrays are objects, so they are passed by reference.
- The elements of an array can be modified inside of a method.

```
int[] arr = new int[5];
fillArray(arr);
```



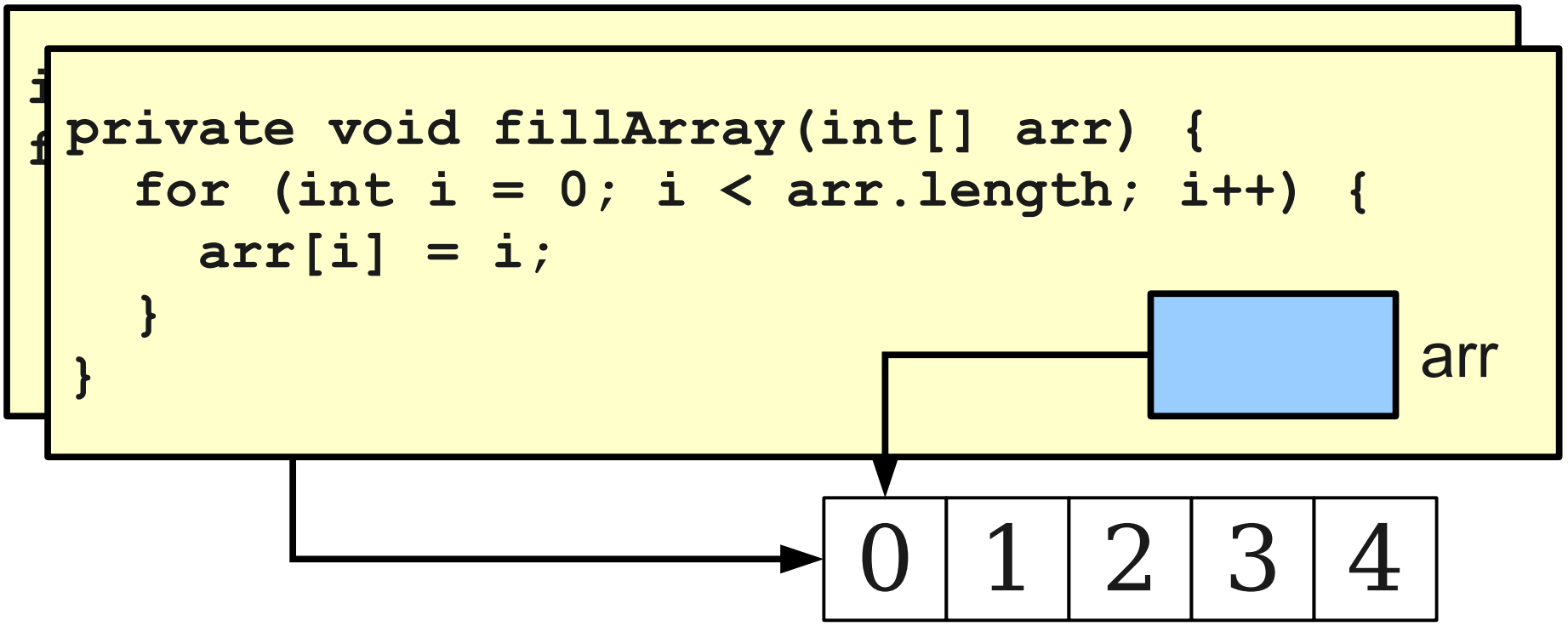
# Arrays as Parameters

- Arrays are objects, so they are passed by reference.
- The elements of an array can be modified inside of a method.



# Arrays as Parameters

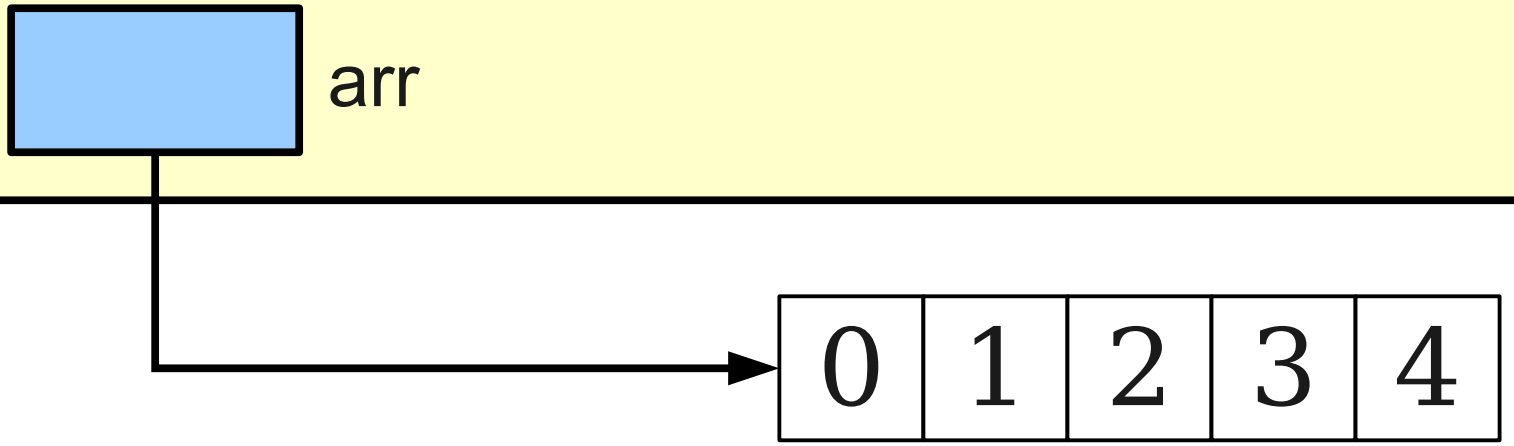
- Arrays are objects, so they are passed by reference.
- The elements of an array can be modified inside of a method.



# Arrays as Parameters

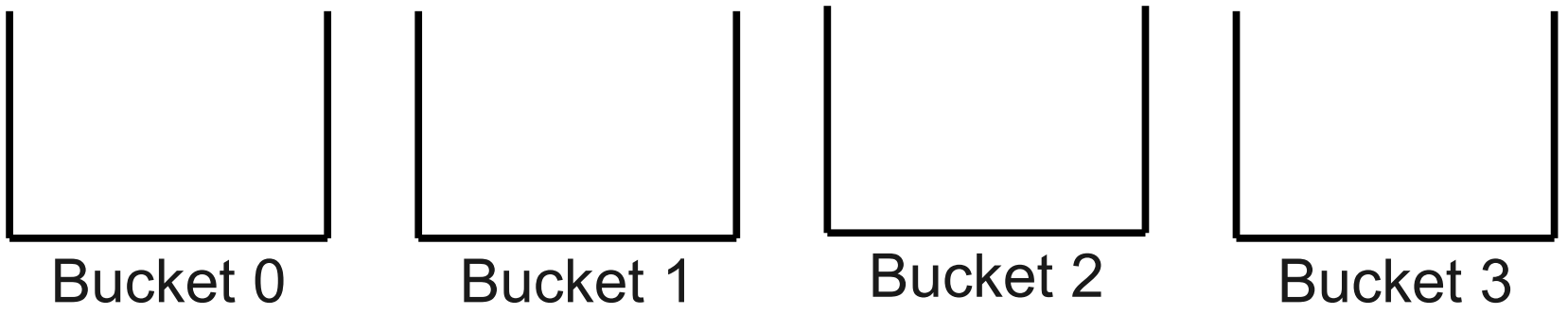
- Arrays are objects, so they are passed by reference.
- The elements of an array can be modified inside of a method.

```
int[] arr = new int[5];
fillArray(arr);
```



# Why Arrays?

- Arrays are excellent for representing a fixed-size list of **buckets**.
- We can store values in the appropriate bucket by looking up the bucket by index.



How many people need to be in a room before two of them will share a birthday?

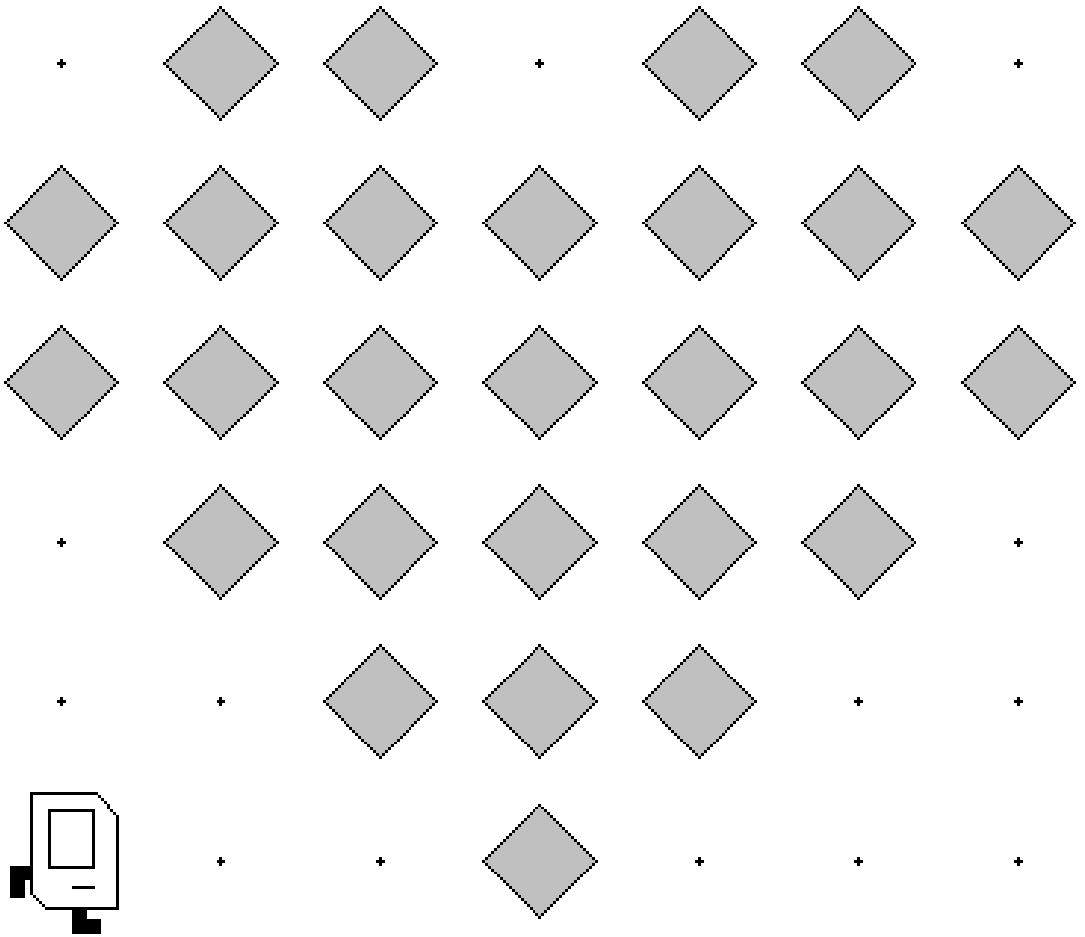
# The Birthday Paradox

- In a room of 23 people, there is a 50% chance that two of them have the same birthday.
- More generally, if you have an $n$-sided die, you only need to roll it around $\sqrt{2n}$ times before you have a 50% chance of getting the same outcome twice.

Fun programming exercise: How many people do you need, on average, for three people to share a birthday?

Time-Out for Announcements!

# Happy Valentine's Day!



# Friday Four Square!

Today at 4:15PM, outside Gates

# Assignment 4

- Assignment 4 is due next Wednesday at 3:15PM.
- Recommendations:
  - Complete steps 1 and 2 as soon as possible.
  - Try to get steps 3, 4, and 5 completed by Monday.
  - Finish steps 6 and 7 by Wednesday.
- Questions? Feel free to stop by the LaIR (closed Sunday, but open Monday and Tuesday), email your SL, or ask on QuestionHut.

Hemingway: Computational Editing!

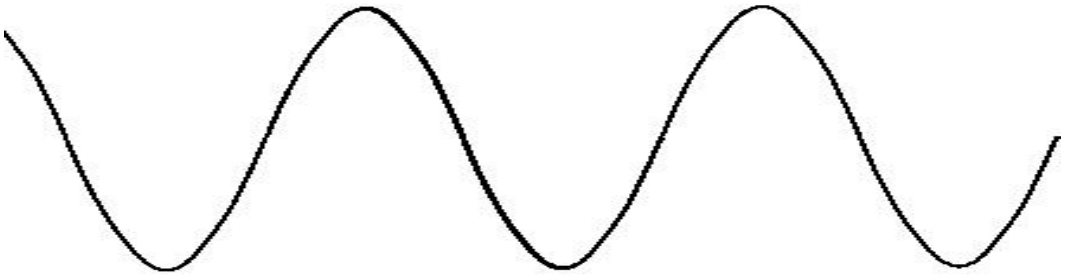
http://www.hemingwayapp.com/

# Back to CS106A!

# Sound Processing

# The Physics of Sound

- Sound is a wave that propagates through the air.



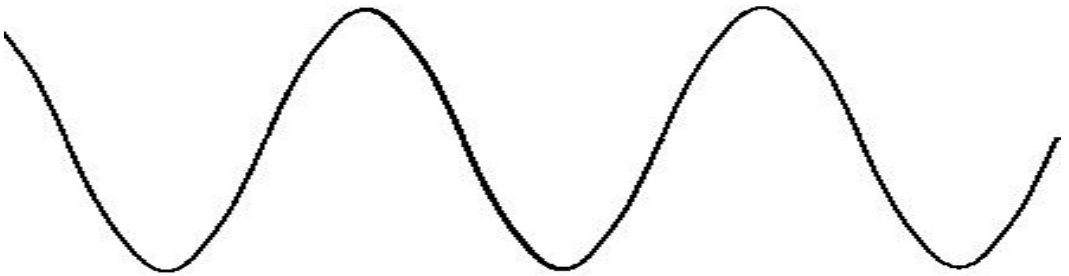
- The **frequency** of the wave is how closely packed together the peaks are.
  - Corresponds to **pitch**.
- The **amplitude** of the wave is how tall the peaks are.
  - Corresponds to **loudness**.

# Representing Sound

- The computer can represent a sound by storing the sound wave.

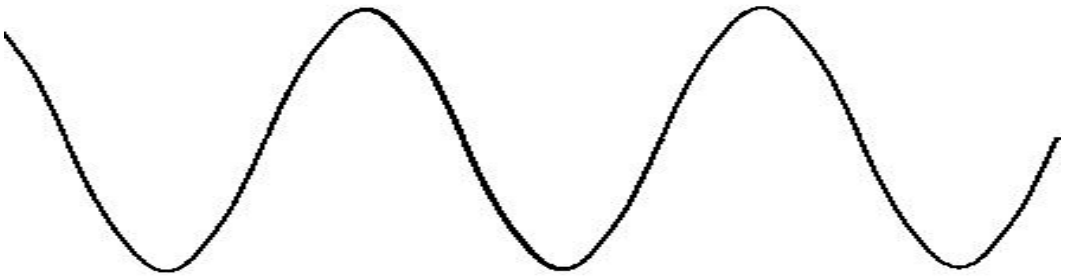
# Representing Sound

- The computer can represent a sound by storing the sound wave.



# Representing Sound

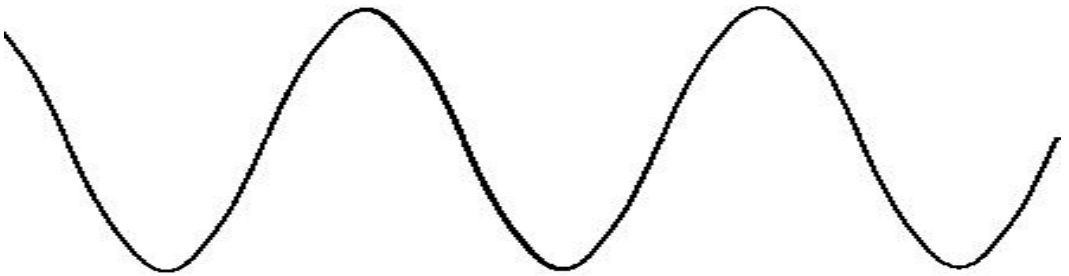
- The computer can represent a sound by storing the sound wave.



- Unfortunately, the wave is continuous, so the computer cannot store it completely.

# Representing Sound

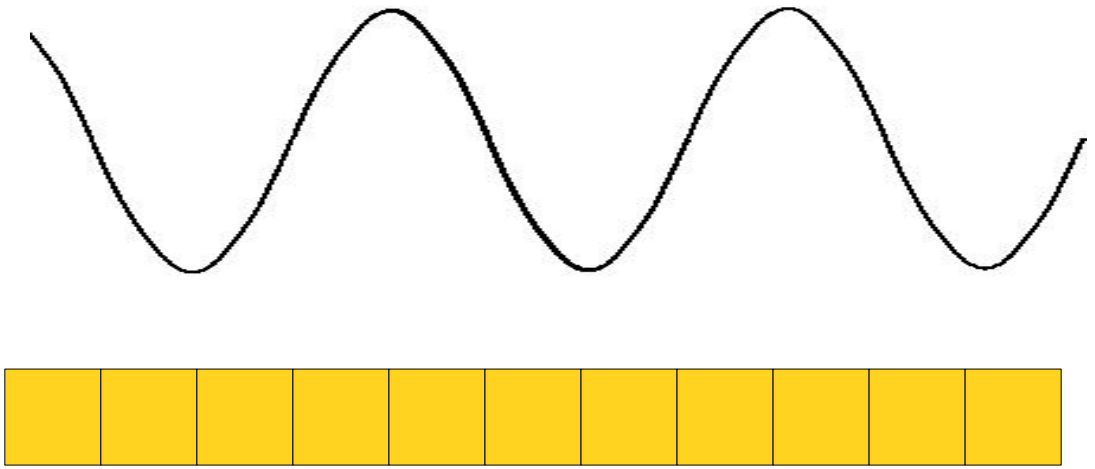
- The computer can represent a sound by storing the sound wave.



- Unfortunately, the wave is continuous, so the computer cannot store it completely.
- **Idea:** Sample points from the sound wave and store those instead.

# Representing Sound

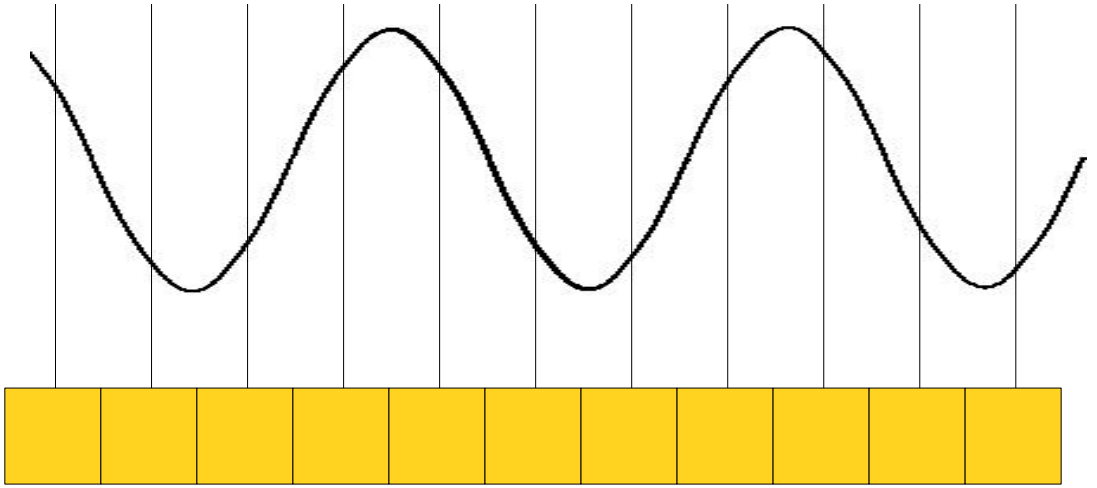
- The computer can represent a sound by storing the sound wave.



- Unfortunately, the wave is continuous, so the computer cannot store it completely.
- **Idea:** Sample points from the sound wave and store those instead.

# Representing Sound

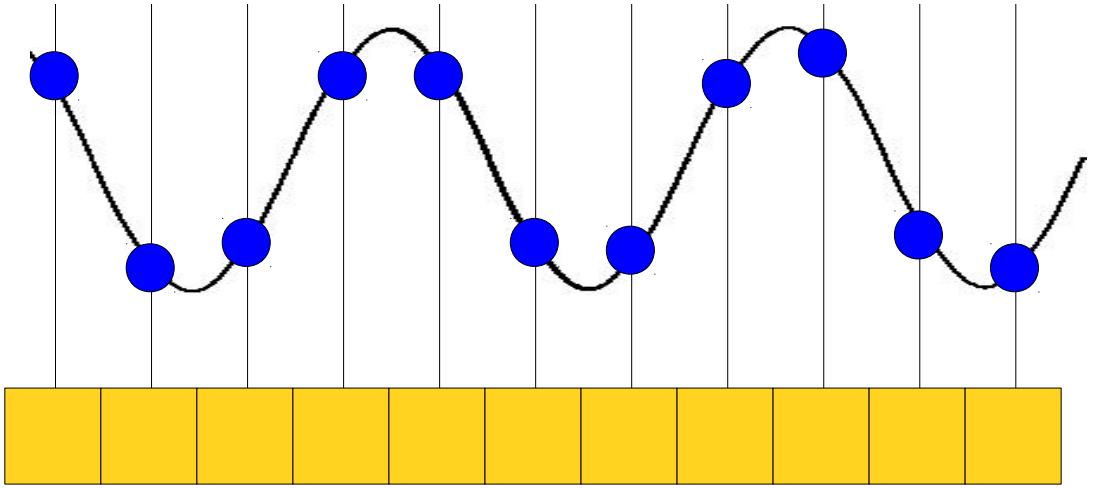
- The computer can represent a sound by storing the sound wave.



- Unfortunately, the wave is continuous, so the computer cannot store it completely.
- **Idea:** Sample points from the sound wave and store those instead.

# Representing Sound

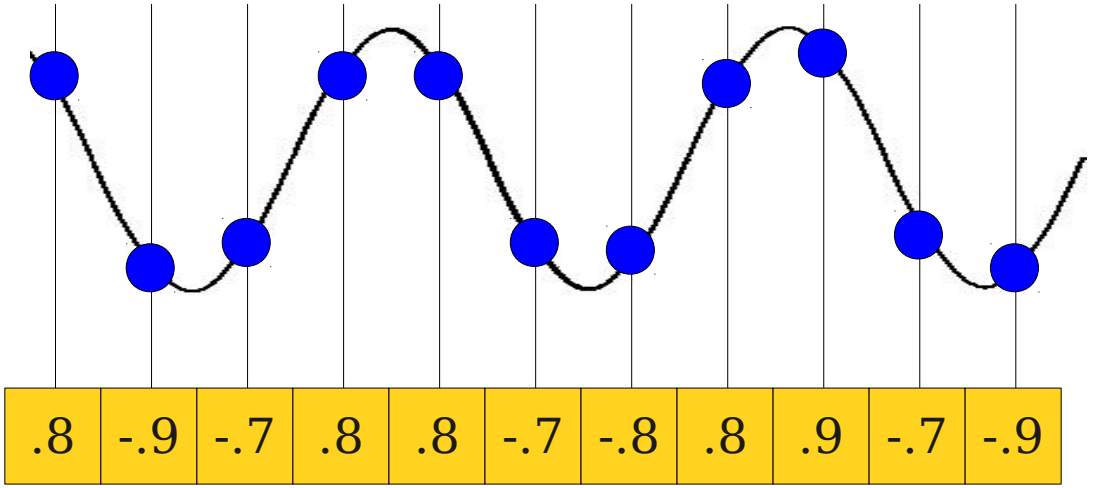
- The computer can represent a sound by storing the sound wave.



- Unfortunately, the wave is continuous, so the computer cannot store it completely.
- **Idea:** Sample points from the sound wave and store those instead.

# Representing Sound

- The computer can represent a sound by storing the sound wave.



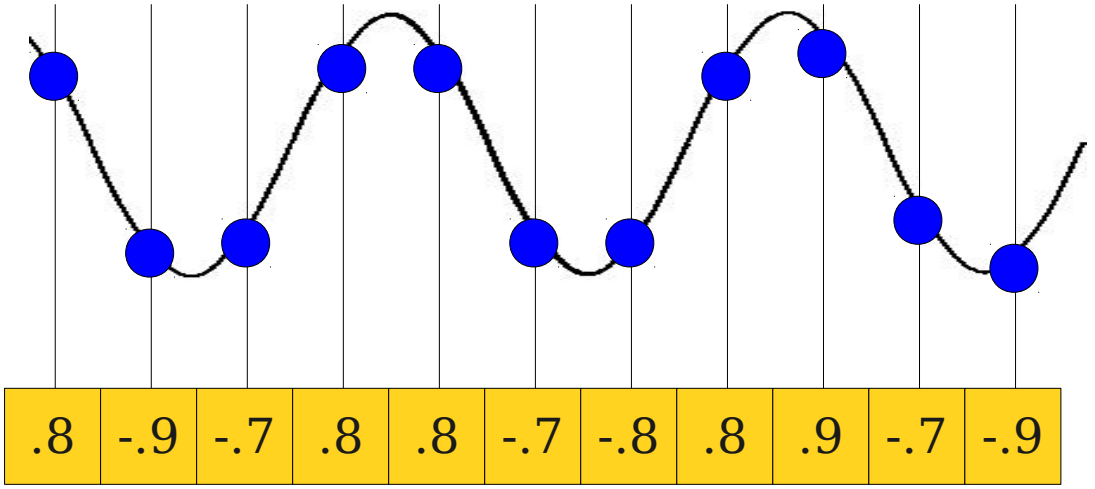

- Unfortunately, the wave is continuous, so the computer cannot store it completely.
- **Idea:** Sample points from the sound wave and store those instead.

# The Sampling Rate

- The **sampling rate** of a sound clip is the frequency at which the wave's intensity is recorded.
  - Measured in hertz (Hz).
- Example: If sampling rate is 44,100Hz, there are 44,100 samples per second.
- High sampling rate makes for better sound.
- Low sampling rate uses less storage space.

# Frequency and Wavelength

- Typically, tones are specified as frequencies (number of wavelengths per second).
- When manipulating sound, it is easier to use the **wavelength** (the number of sound samples corresponding to one complete wave).




Conversion formula:

**wavelength = sampling rate / frequency**

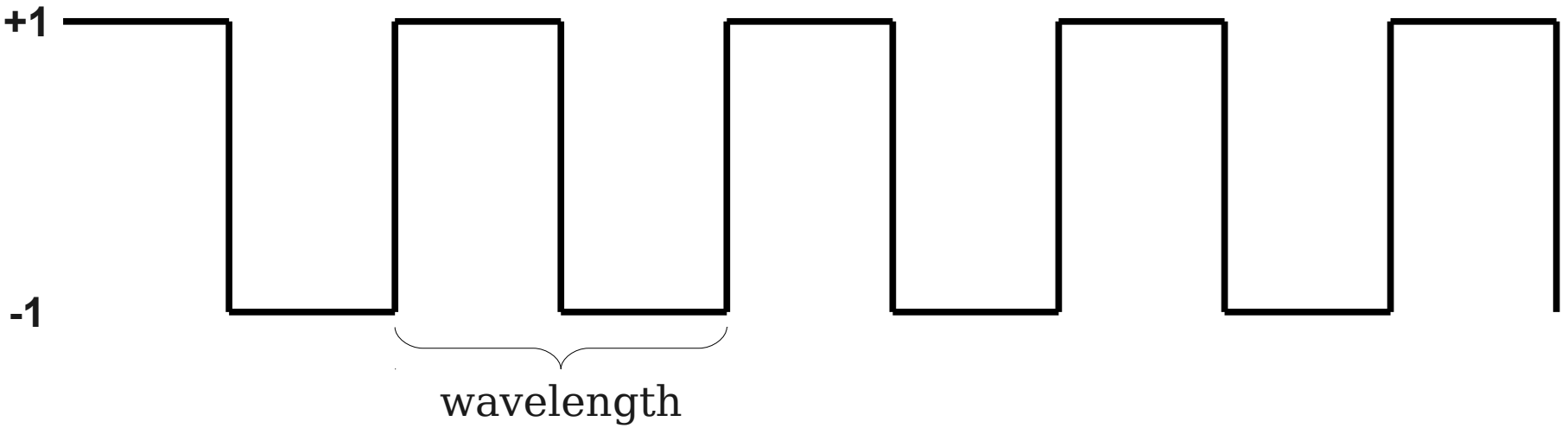
# Generating Sound

- Today, we'll use Princeton's **`StdAudio`** class to play sounds.
- Each sound clip is represented as a **`double[]`**, where each entry is between -1 and +1.
- We can play the sound by calling

```
StdAudio.play(soundClip)
```

# Square Waves

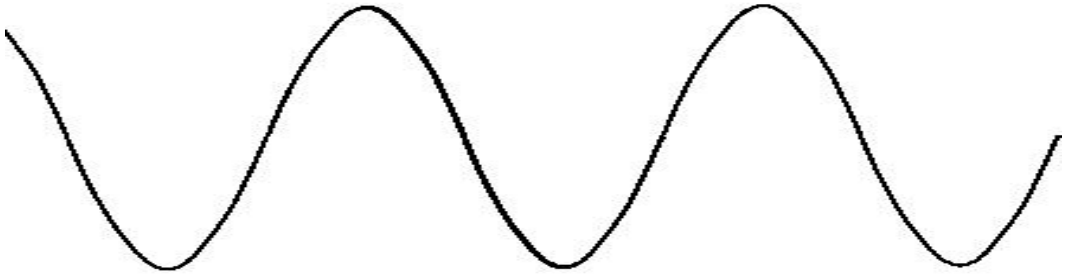
- A **square wave** is a simple wave that alternate between on and off at a predictable rate.




- We can generate a square wave at a given frequency as follows:
  - Compute the wavelength.
  - Fill in the array by writing +1 if we're in the first half of a wave and -1 if we're in the second half.

# Sine Waves

- A **sine wave** is a simple wave that humans perceive as a pure tone.



- To generate a sine wave, we can do the following:
  - Compute the wavelength.
  - For each array index, figure out where in the wave we are and compute the sine of that value.