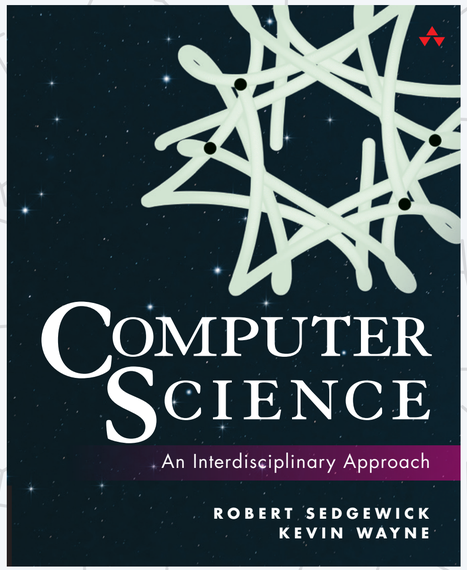
Robert Sedgewick | Kevin Wayne

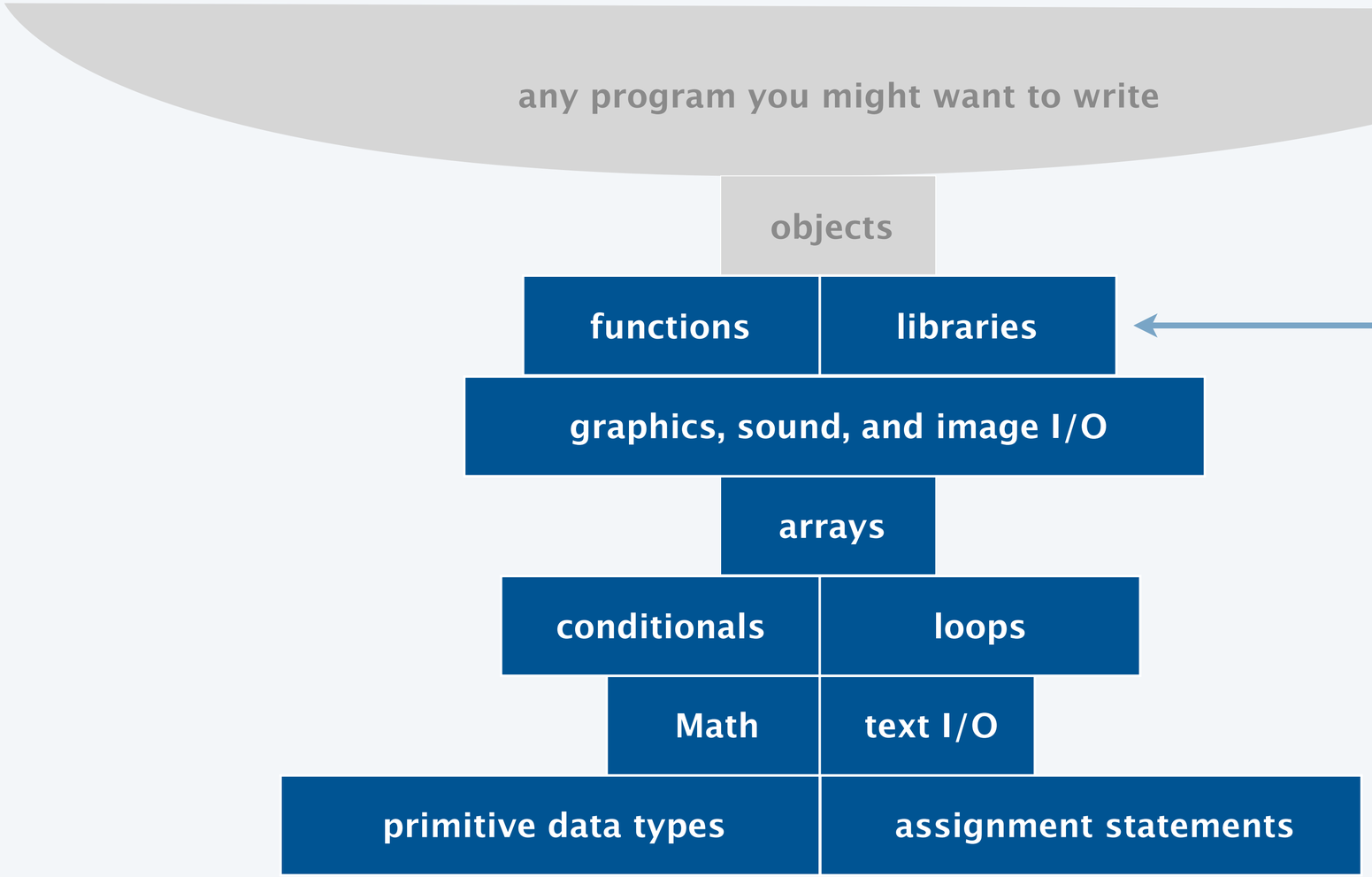


https://introcs.cs.princeton.edu

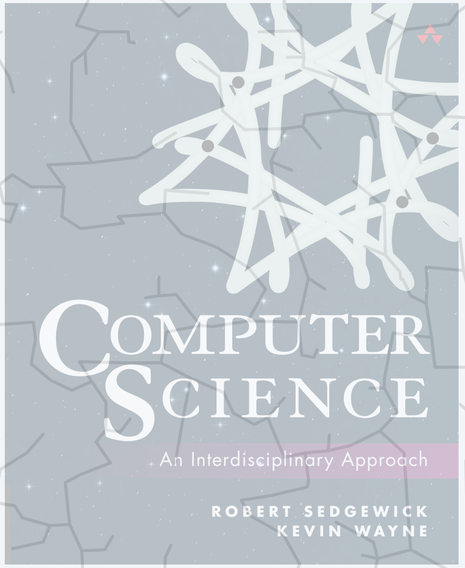
2.2 Libraries and Clients

- *random number library*
- *designing libraries*
- *sound synthesis*
- *synthesizer library*

Basic building blocks for programming



build reusable libraries



https://introcs.cs.princeton.edu

2.2 Libraries and Clients

- *random number library*
- *designing libraries*
- *sound synthesis*
- *synthesizer library*

Standard random library

Goal. Design a library to generate pseudo-random numbers.

`public class StdRandom`

`static double uniformDouble()`	*real number between* 0 *and* 1
`static double uniformDouble(double lo, double hi)`	*real number between* `lo` *and* `hi`
`static boolean bernoulli(double p)`	`true` *with probability* `p`, `false` *otherwise*
`static int uniformInt(int n)`	*integer between* 0 *and* `n-1`
`static double gaussian()`	*normal with mean* 0 *and stddev* 1
`static double gaussian(double mu, double sigma)`	*normal with mean* `mu` *and stddev* `sigma`
`static void shuffle(String[] a)`	*shuffle the string array* `a[]`
`static int discrete(int[] freq)`	`i` *with probability proportion to* `freq[i]`
⋮	⋮

Standard random implementation: random numbers from various distributions

```
public class StdRandom {

   public static double uniformDouble() {
      return Math.random();
   }

   public static double uniformDouble(double lo, double hi) {
      return lo + (uniformDouble() * (hi - lo));
   }

   public static boolean bernoulli(double p) {
      return uniformDouble() < p;
   }

   public static int uniformInt(int n) {
      return (int) (uniformDouble() * n);
   }

   ⋮
}
```

calls a method (in a different class)

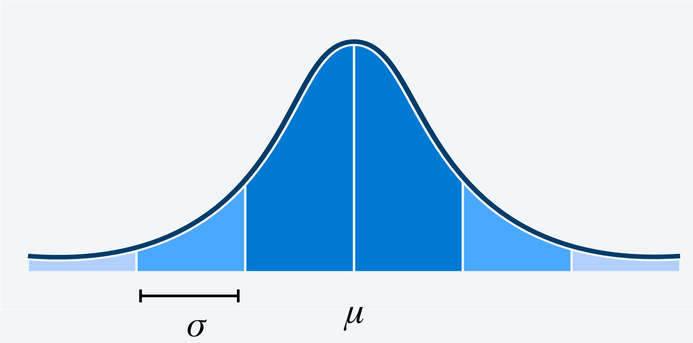
you could re-implement these methods in each program, but now you don't have to!

calls a method (in the same class)

Standard random implementation: random numbers from a Gaussian distribution

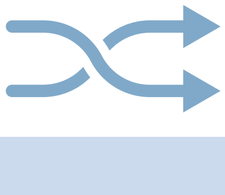
```
public class StdRandom {

   public static double gaussian() {
      double r, x, y;
      do {
         x = uniformDouble(-1.0, 1.0);
         y = uniformDouble(-1.0, 1.0);
         r = x*x + y*y;
      } while (r >= 1 || r == 0);
      return x * Math.sqrt(-2 * Math.log(r) / r);
   }

   public static double gaussian(double mu, double sigma) {
      return mu + gaussian() * sigma;
   }

   ⋮
}
```

can call a method without knowing how it is implemented



Standard random implementation: shuffling the elements in an array

```
public class StdRandom {

   private static void exch(String[] a, int i, int j) {
      String temp = a[i];
      a[i] = a[j];
      a[j] = temp;
   }

   public static void shuffle(String[] a) {
      for (int i = 0; i < a.length; i++) {
         int r = uniformInt(i+1);
         exch(a, i, r);
      }
   }

   ⋮
}
```

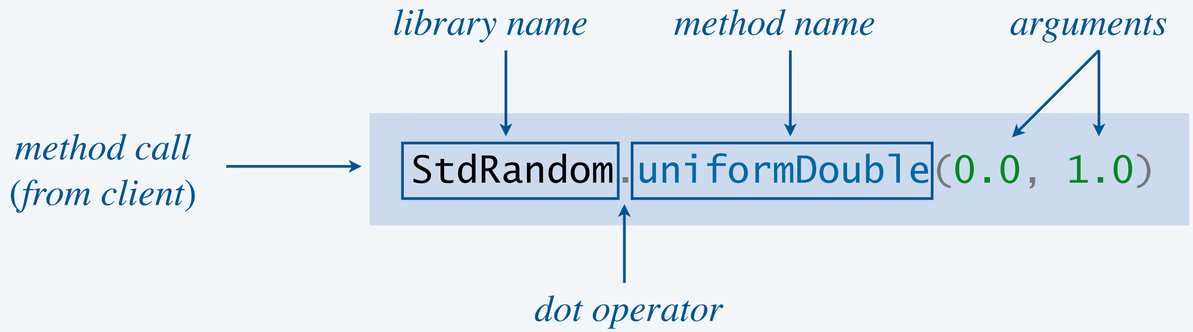


calls a `private` *method*
(in the same class)

`private` *helper method*
(cannot be called from outside this class)

Calling a library function

Calling from a client. Specify library name, dot operator, function name, and arguments.



Note. Must use fully qualified name if calling a function from another file.

Standard random clients

StdRandom client 1

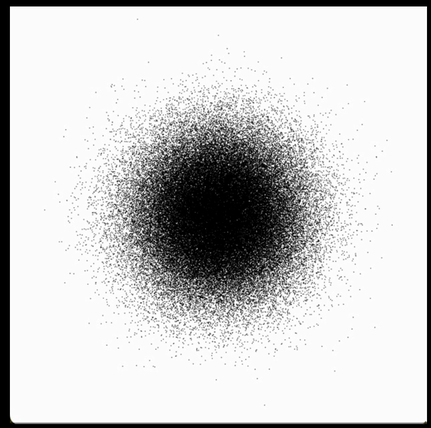
```
public class Shuffle {
   public static void main(String[] args) {
      StdRandom.shuffle(args);
      for (int i = 0; i < args.length; i++) {
         StdOut.print(args[i] + " ");
      }
      StdOut.println();
   }
}
```

```
~/cos126/libraries> java-introcs Shuffle A B C D E
E A D B C

~/cos126/libraries> java-introcs Shuffle A B C D E
C A E B D

~/cos126/libraries> java-introcs Shuffle 2C 2D 2H ... AS
4S 2D AC 9H QH 8C ... JS 4H 2S
```

StdRandom client 2

```
public class RandomPoints {
   public static void main(String[] args) {
      int n = Integer.parseInt(args[0]);
      for (int i = 0; i < n; i++) {
         double x = StdRandom.gaussian(0.5, 0.1);
         double y = StdRandom.gaussian(0.5, 0.1);
         StdDraw.point(x, y);
      }
   }
}
```

```
~/cos126/libraries> java-introcs RandomPoints 100000
```



Libraries and clients: quiz 1

What is the probability that the following code fragment prints "Paper" ?

A. 0

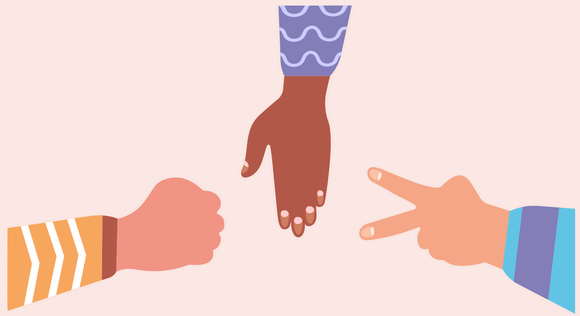
B. 2 / 9

C. 1 / 4

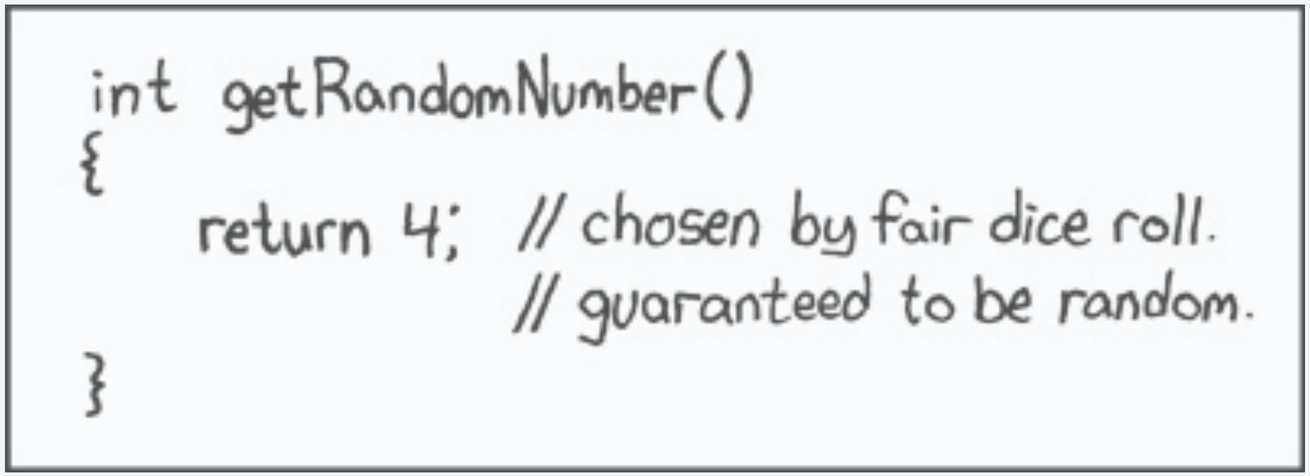
D. 1 / 3

E. 4 / 9

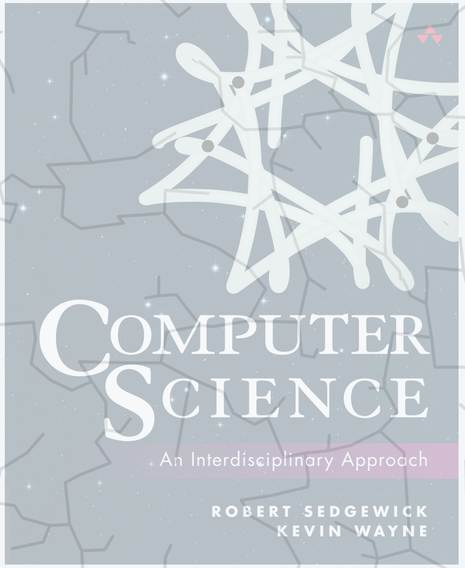
```
if (StdRandom.uniformInt(3) == 0) {
    System.out.println("Rock");
}
else if (StdRandom.uniformInt(3) == 1) {
    System.out.println("Paper");
}
else {
    System.out.println("Scissors");
}
```



Another bug...



https://xkcd.com/221/



https://introcs.cs.princeton.edu

2.2 Libraries and Clients

- *random number library*
- *designing libraries*
- *sound synthesis*
- *synthesizer library*

Libraries

Def. A module is a set of functions stored in a single file.

Def. A library is a module whose primary purpose is for use by other programs.

← *definitions for this course*

library	description	example method call	source	logo
StdRandom	*generate random numbers*	`StdRandom.uniformInt(6)`	textbook	
StdDraw	*draw geometric shapes*	`StdDraw.circle(0.5, 0.5, 0.25)`		
Math	*compute mathematical functions*	`Math.sqrt(2.0)`	Java system	
java.util.Arrays	*manipulate arrays*	`Arrays.sort(a)`		
Gaussian	*compute Gaussian pdf and cdf*	`Gaussian.pdf(3.0)`	user-defined	
SayNumber	*speak numbers*	`SayNumber.sayInteger(126)`		
⋮	⋮		⋮	

API, client, and implementation

Application programming interface (API). Specifies method headers and behavior for a library.

Implementation. Program that implements the methods in an API.

Client. Program that uses a library through its API.

contract between client and implementation

client

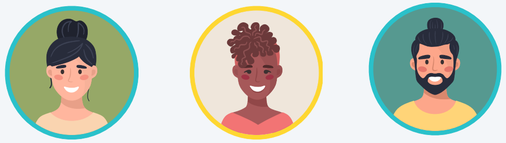
```
StdRandom.uniformInt(6);

StdRandom.shuffle(args);
```

API

```
public class StdRandom
static int  uniformInt(int n)
static void shuffle(String[] a)
      ⋮
```

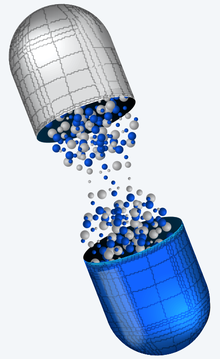
implementation

```
public class StdRandom {
   ...
   public static int uniformInt(int n) {
      ...
   }

   public static void shuffle(String[] a) {
      ...
   }
}
```



Encapsulation

Encapsulation. Separating clients from implementation details by hiding information.



Principle. A client does not need to know how a method is implemented in order to use it.

Benefits.

- Can develop client code and implementation code independently.
- Can change implementation details without breaking clients.

Private access modifier. Designates a method as not for use by a client.

- API does not list *private* methods.
- Compile-time error for client to call a *private* method.
- Advantage: implementation can add/remove *private* methods without impacting clients.

Accessing a library

Java classpath. Places where Java looks for user-defined libraries (and other resources).

- Simplest: put library `.class` file in same directory as client program.
- Best practice: bundle library `.class` files in a `.jar` file; add `.jar` file to Java classpath.

`stdlib.jar` *contains*:
`StdRandom.class`
`StdIn.class`
`StdOut.class`
`StdDraw.class`
`StdPicture.class`
`StdAudio.class`
⋮

```
~/cos126/libraries> javac Shuffle.java
Shuffle.java:3: error: cannot find symbol
      StdRandom.shuffle(args);

~/cos126/libraries> javac-introcs Shuffle.java

~/cos126/libraries> java-introcs Shuffle A B C D E
C A E B D
```

adds `stdlib.jar` *to Java classpath*

Unit testing

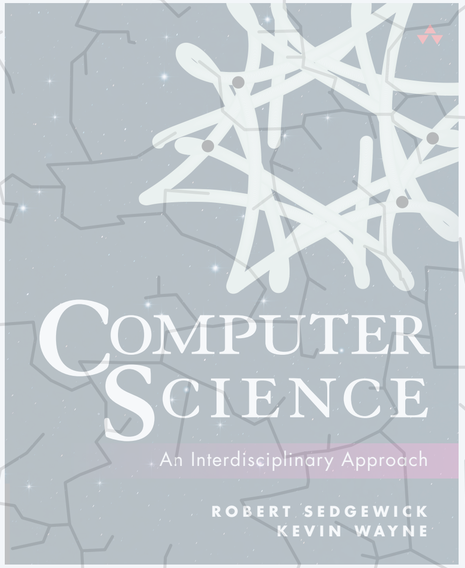
Best practice. Include a *main()* method in each class as a test client.

- Call each public method at least once.
- Use result to check behavior.
- Identify failed tests programmatically.

minimum requirements (in this course)

```
public class StdRandom {
   ⋮
   public static void main(String[] args) {
      int n = Integer.parseInt(args[0]);
      for (int i = 0; i < n; i++) {
         StdOut.printf("%8.5f ", uniformDouble(10.0, 99.0));
         StdOut.printf("%5b "  , bernoulli(0.5));
         StdOut.printf("%2d "  , uniformInt(100));
         StdOut.printf("%7.5f ", gaussian(9.0, 0.2));
         StdOut.println();
      }
      ⋮
   }
}
```

unit tests for `shuffle()` *and other methods*



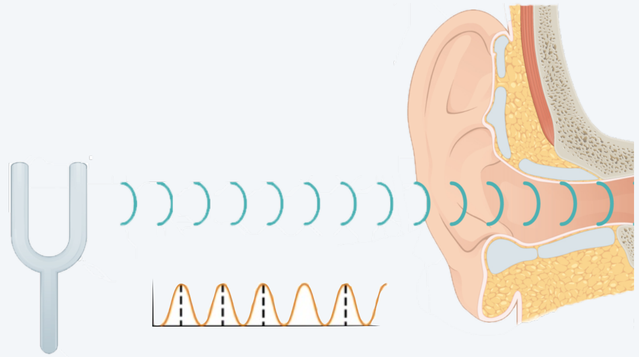
https://introcs.cs.princeton.edu

2.2 Libraries and Clients

- *random number library*
- *designing libraries*
- *sound synthesis*
- *synthesizer library*

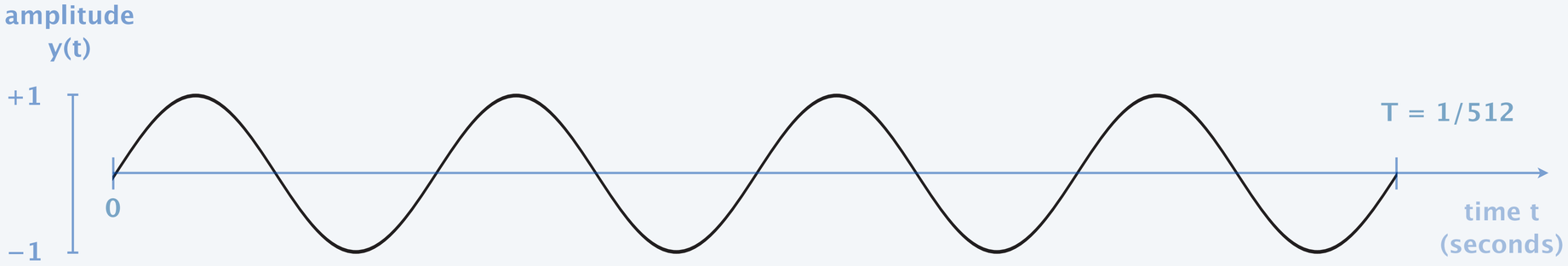
Review: digital audio

Sound is the perception the vibration of our eardrums.



Audio signal. Real-valued (between -1 and $+1$) function of time.

Pure tone. Sound wave defined by the sine function of given *frequency*, *amplitude* and *duration*.



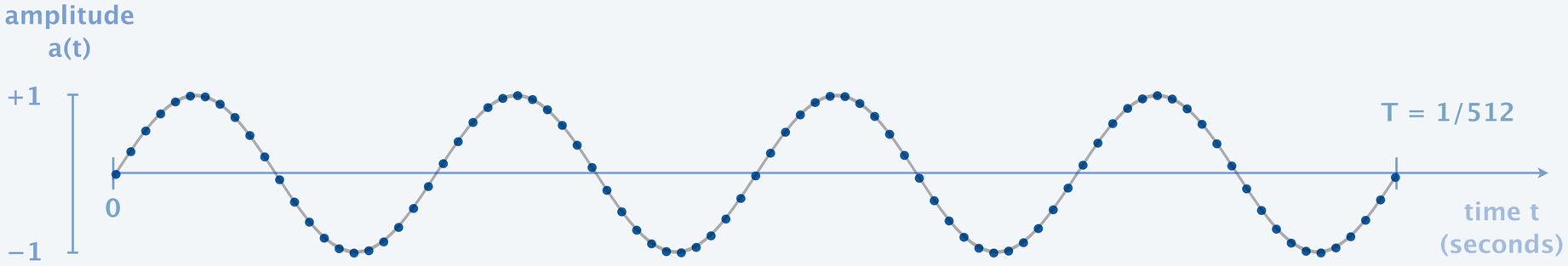
$$y(t) = \sin(2\pi \cdot 2048 \cdot t), \quad 0 \le t \le T$$

Review: audio sampling

Goal. Convert a continuous-time signal into a discrete-time signal.

- A sample is a signal value at specific point in time.
- Take samples at evenly spaced points.

model sound with an array of real numbers between −1 and +1 (using 44,100 samples per second)



$$y(t) = \sin(2\pi \cdot 2048 \cdot t), \quad 0 \le t \le T$$

$$a(t) = \sin(2\pi \cdot 2048 \cdot t), \quad t = \frac{0}{44100}, \frac{1}{44100}, \frac{2}{44100}, \ldots$$

Review: standard audio API

StdAudio. Our library for playing, reading, and saving digital audio.

`public class StdAudio`

`static int`	`SAMPLE_RATE`	44,100 (*CD quality audio*)
`static void`	`play(String filename)`	*play the audio file*
`static void`	`playInBackground(String filename)`	*play the audio file in the background*
`static void`	`play(double sample)`	*play the sample*
`static void`	`play(double[] samples)`	*play the samples*
`static double[]`	`read(String filename)`	*read the samples from an audio file*
`static void`	`save(String filename, double[] samples)`	*save the samples to an audio file*
	⋮	⋮

Sine wave implementation

implementation

```
public class Synth {

   public static int numberOfSamples(double duration) {
      return (int) (StdAudio.SAMPLE_RATE * duration);
   }

   private static double sine(double freq, double t) {
      return Math.sin(2 * Math.PI * freq * t);
   }

   public static double[] sineWave(double freq, double amplitude, double duration) {
      int n = numberOfSamples(duration);
      double[] a = new double[n];
      for (int i = 0; i < n; i++) {
         double t = 1.0 * i / StdAudio.SAMPLE_RATE;
         a[i] = amplitude * sine(freq, t);
      }
      return a;
   }

}
```

utility method

for internal use only
(`private` helper methods)

sample at n equally spaced points

$$a(t) = A \sin\left(2\pi \cdot f \cdot t\right), \quad t = \frac{0}{44100}, \frac{1}{44100}, \frac{2}{44100}, \ldots$$

client

```
double[] a = Synth.sineWave(2048.0, 0.5, 3.0);
StdAudio.play(a);
```

Libraries and clients: quiz 2

What sound will the following code fragment produce?

```
double freq = 17400.0;
double amplitude = 0.5;
double duration = 10.0;
double[] a = Synth.sineWave(freq, amplitude, duration);
StdAudio.play(a);
```

A. Extremely high-pitched sound.

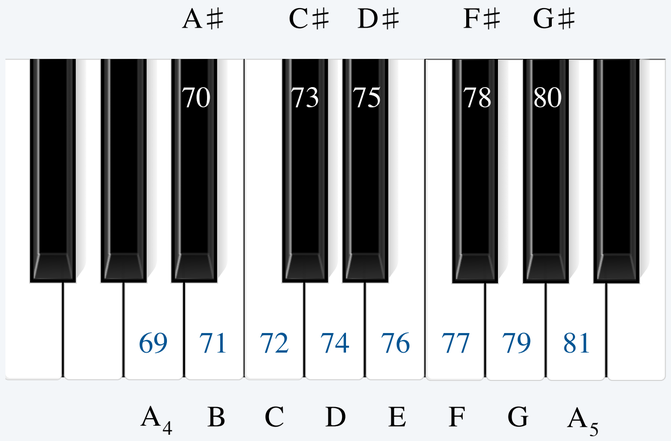
B. Inaudible.

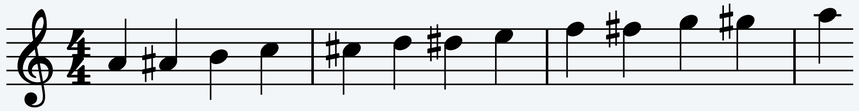
C. Ultrasonic weapon.

D. All of the above.

Crash course in Western music

- Concert A is 440 Hz.
- An octave is the interval between a note and one with twice its frequency.
- Octave is divided into 12 notes on a logarithmic scale. ⟵ *"twelve-tone equal temperament"*





note	MIDI (m)	frequency (Hz) $(440 \times 2^{(m-69)/12})$	sine wave
A_4	69	440	
A♯ / B♭	70	466.16	
B	71	493.88	
C	72	523.25	
C♯ / D♭	73	554.37	
D	74	587.33	
D♯ / E♭	75	622.25	
E	76	659.26	
F	77	698.46	
F♯ / G♭	78	739.99	
G	79	783.99	
G♯ / A♭	80	830.61	
A_5	81	880	

Libraries and clients: quiz 3

Which of the following converts from MIDI note number to frequency?

A.

```
private static double midiToFrequency(int midi) {
   return 440 * Math.pow(2, (midi - 69) / 12);
}
```

B.

```
private static double midiToFrequency(int midi) {
   return 440.0 * 2.0 ^ ((midi - 69.0) / 12.0);
}
```

$$frequency = 440 \times 2^{(midi - 69) / 12}$$

C. Both A and B.

D. Neither A nor B.

MIDI-number-to-frequency conversion

Goal. Add methods (and constants) to library that many clients might want to use.

Musical Instrument Digital Interface (MIDI). Digital music standard.

Class constant.

- Declare and initialize "variable" outside of any method, using *final* and *static* modifiers.
- Access modifier can be *public* or *private*.
- Java naming convention: use *SCREAMING_SNAKE_CASE*.

```
public class Synth {
   public static final double CONCERT_A = 440.0;

   private static double midiToFrequency(int midi) {
      return CONCERT_A * Math.pow(2, (midi - 69) / 12.0);
   }

   ...
}
```

class constant (`static` *variable*)

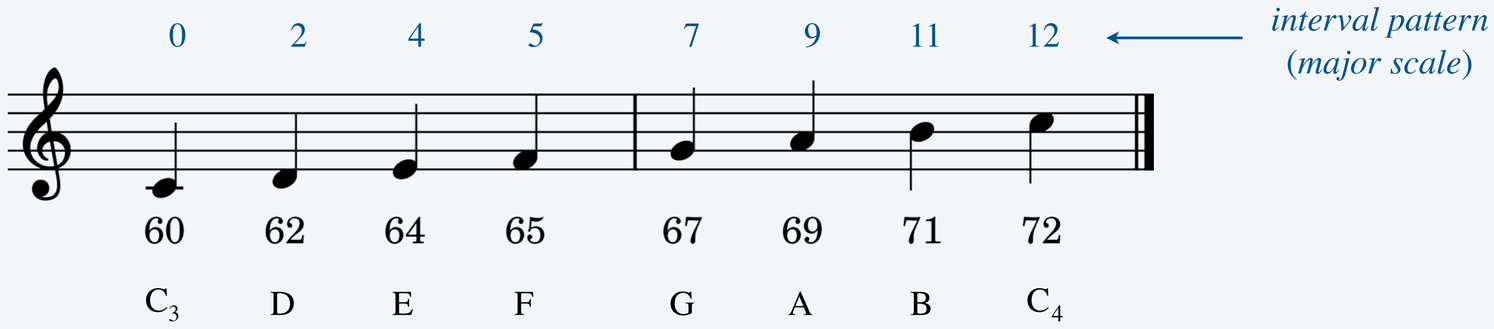
$$frequency = 440 \times 2^{(midi - 69) / 12}$$

implementation

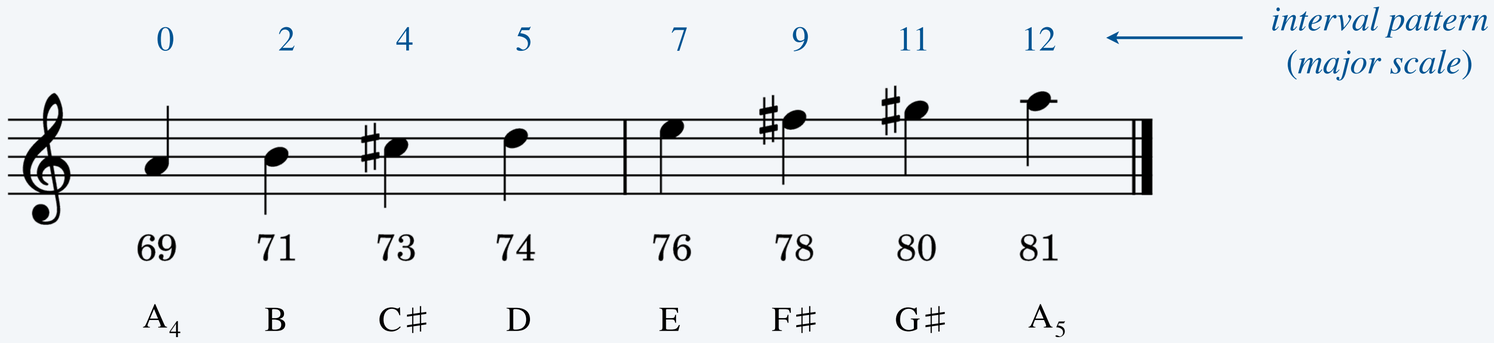
Musical scales

Major scale. Sequence of 8 notes in a specific interval pattern, starting with a root note and ending with the same note one octave higher.

Ex 1. C major scale.



Ex 2. A major scale.



Musical scales

Major scale. Sequence of 8 notes in a specific interval pattern, starting with a root note and ending with the same note one octave higher.

```
public class MajorScale {
   public static void main(String[] args) {
      int root = Integer.parseInt(args[0]);
      double duration = 0.5;
      double amplitude = 0.5;
      int[] pattern = { 0, 2, 4, 5, 7, 9, 11, 12 };
      for (int i = 0; i < pattern.length; i++) {
         int midi = root + pattern[i];
         double freq = Synth.midiToFrequency(midi);
         double[] a = Synth.sineWave(freq, amplitude, duration);
         StdAudio.play(a);
      }
   }
}
```

client

interval pattern (major scale)

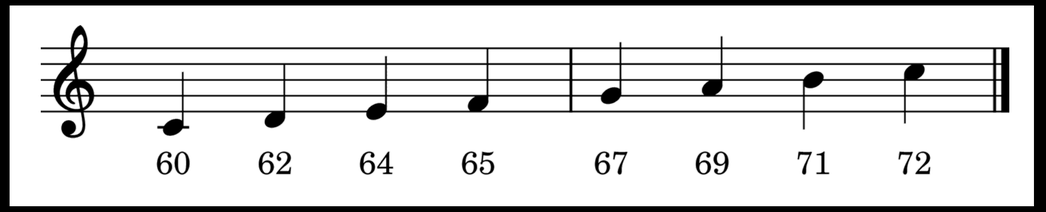
```
~/cos126/libraries> java-introcs MajorScale 60
[plays A major scale]

~/cos126/libraries> java-introcs MajorScale 69
[plays C major scale]
```

Play that tune

Goal. Read in a sequence of MIDI note numbers and durations from standard input, and play the synthesized results to standard audio.

```
~/cos126/libraries> more MajorScaleC.txt
60 0.5
62 0.5
64 0.5
65 0.5
67 0.5
69 0.5
71 0.5
72 0.5
```

duration (seconds)



MIDI note number

```
~/cos126/libraries> java-introcs PlayThatTune < MajorScaleC.txt
```

[plays C major scale]

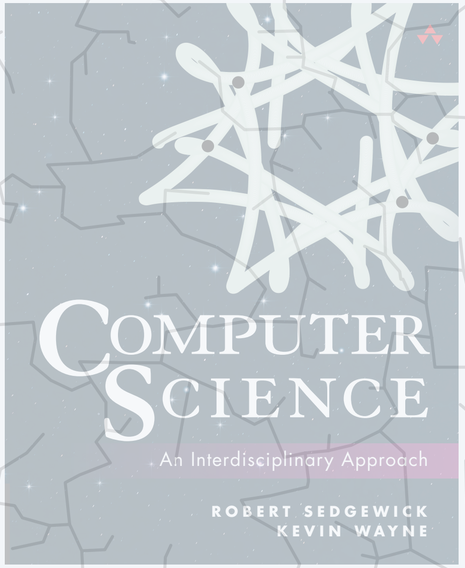
Play that tune

Goal. Read in a sequence of MIDI note numbers and durations from standard input, and play the synthesized results to standard audio.

client

```
public class PlayThatTune {
   public static void main(String[] args) {
      double amplitude = 0.5;
      while (!StdIn.isEmpty()) {
         int midi = StdIn.readInt();
         double duration = StdIn.readDouble();
         double freq = Synth.midiToFrequency(midi);
         double[] a = Synth.sineWave(freq, amplitude, duration);
         StdAudio.play(a);
      }
   }
}
```

```
~/cos126/libraries> java-introcs PlayThatTune < Arpeggio.txt
[plays arpeggio]

~/cos126/libraries> java-introcs PlayThatTune < LooneyTunes.txt
[plays Looney Tunes theme]

~/cos126/libraries> java-introcs PlayThatTune < FurElise.txt
[plays beginning of Fur Elise]
```



https://introcs.cs.princeton.edu

2.2 Libraries and Clients

- *random number library*
- *designing libraries*
- *sound synthesis*
- *synthesizer library*

Digital synthesizers

Digital synth. Electronic musical instrument that generates audio signals digitally.

- Sound effects.
- Film and television soundtracks.
- Diverse genres of music (rock, jazz, pop, disco, hip-hop, electronic music, ...).
- ...

R2-D2
(Star Wars)

Axel F
(Harold Faltemeyer)

Synthesizer API

Synth. A library for synthesizing sound.

```
public class Synth
```

`static int CONCERT_A`	440.0 (*concert A pitch in Hz*)
`static int numberOfSamples(double duration)`	*number of audio samples*
`static double midiToFrequency(int midi)`	*frequency of MIDI note number*

utility methods

`static double sineWave(double freq, double amplitude, double duration)`	*sine wave*
`static double squareWave(double freq, double amplitude, double duration)`	*square wave*
`static double sawWave(double freq, double amplitude, double duration)`	*saw wave*
`static double supersawWave(double freq, double amplitude, double duration)`	*supersaw wave*
`static double whiteNoise( double amplitude, double duration)`	*white noise*

create sound waves

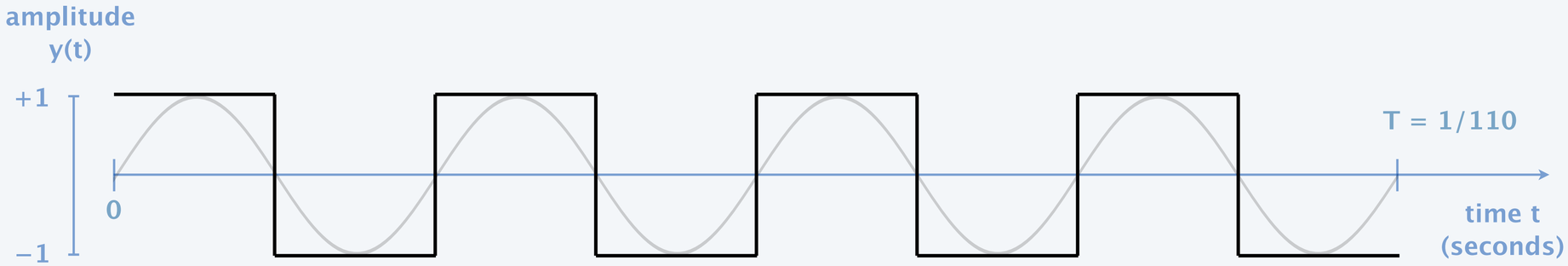
`static double[] superpose(double[] a, double[] b)`	*add the two waves*
`static double[] modulate(double[] a, double[] b)`	*multiply the two waves*
`static double[] fadeIn(double[] a, double lambda)`	*exponential fade in*
`static double[] fadeOut(double[] a, double lambda)`	*exponential fade out*

manipulate sound waves

⋮ ⋮

Square waves

Square wave. Alternates between +1 and −1 with frequency f, half the time at each value.



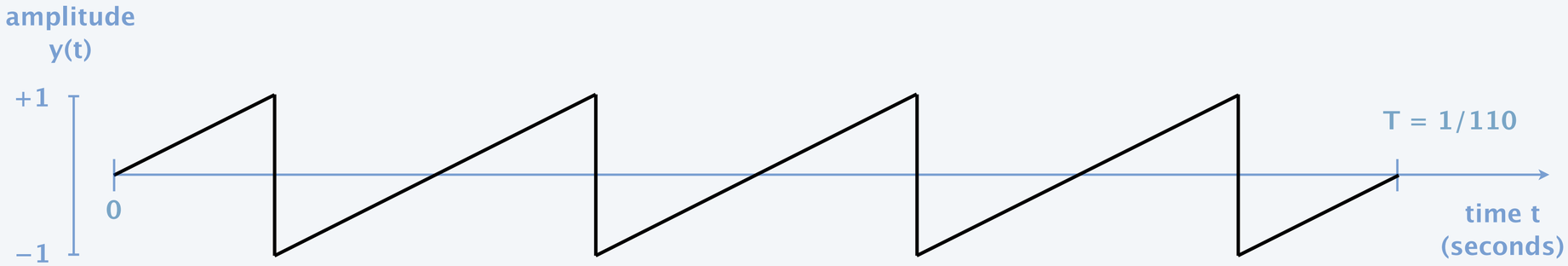
$$a(t) = sgn\left(\sin(2\pi \cdot 440 \cdot t)\right), \quad 0 \le t \le T \qquad sgn(x) = \begin{cases} -1 & \text{if } x < 0 \\ 0 & \text{if } x = 0 \\ +1 & \text{if } x > 0 \end{cases}$$

implementation

```
private static double square(double freq, double t) {
   return Math.signum(sine(freq, t));
}

public static double[] squareWave(double freq, double amplitude, double duration) {
   /* similar to sineWave() */
}
```

Sawtooth waves

Sawtooth wave. Rises from -1 to $+1$ linearly, then drops back to -1, and repeats with frequency f.



$$a(t) = 2\left(440\,t - \left\lfloor 440\,t + \tfrac{1}{2}\right\rfloor\right), \quad 0 \le t \le T$$

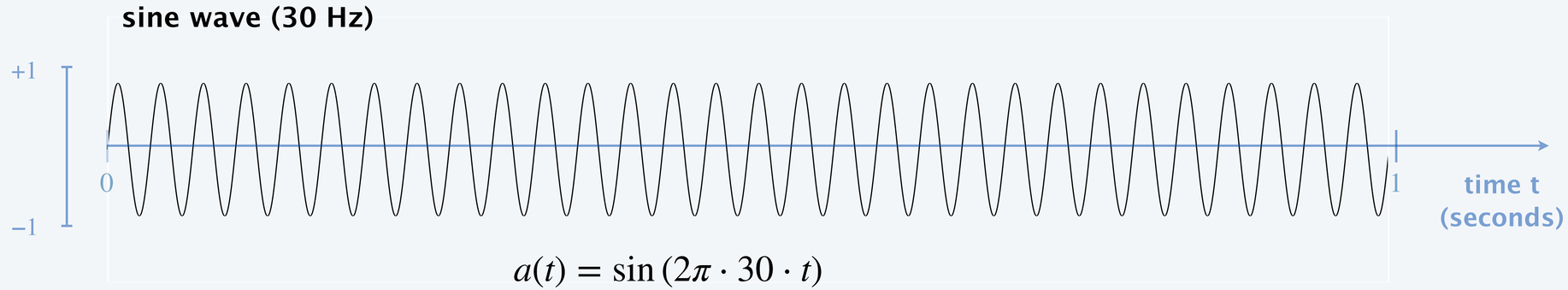
implementation

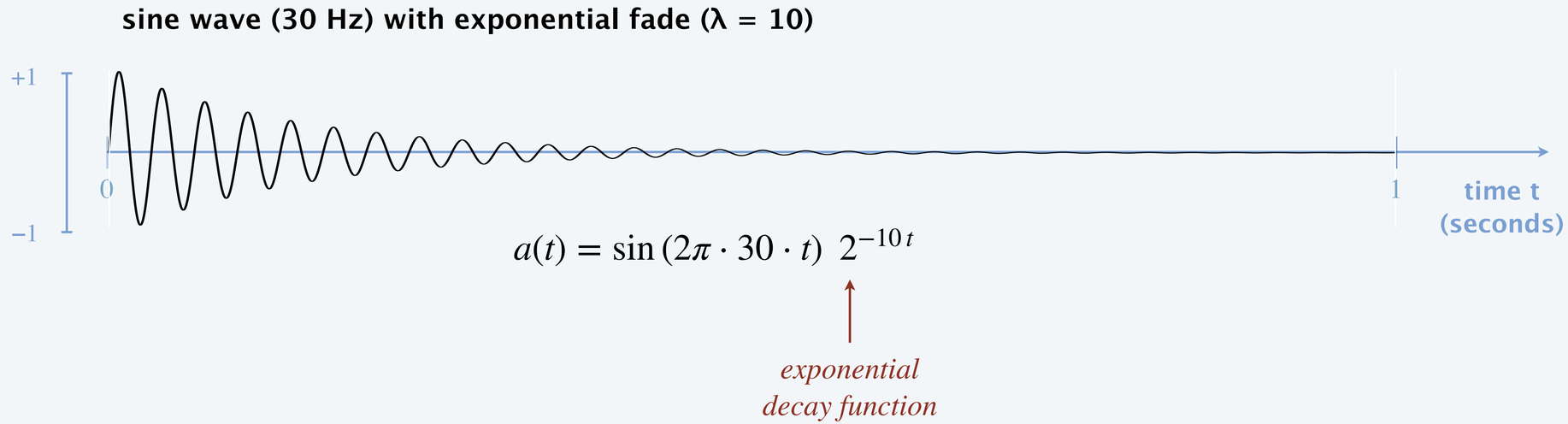
```
private static double saw(double freq, double t) {
   return 2 * (freq*t - Math.floor(freq*t + 0.5));
}

public static double[] sawWave(double freq, double amplitude, double duration) {
   /* similar to sineWave() */
}
```

Exponential fade

Sound envelope. Defines how a sound changes over time.

Exponential fade. A sound envelope whose amplitude decays according to exponential function.





Exponential fade

implementation

```
public class Synth {

   public static double[] fadeOut(double[] a, double lambda) {
      int n = a.length;
      double[] result = new double[n];
      for (int i = 0; i < n; i++) {
         double t = 1.0 * i / StdAudio.SAMPLE_RATE;
         result[i] = a[i] *  Math.pow(2.0, -lambda * t);
      }
      return result;
   }

}
```

client

```
while (true) {
   double[] a = Synth.sineWave(440.0, 0.5, 1.0);
   double[] b = Synth.fadeOut(a, 10.0);
   StdAudio.play(b);
}
```

client

```
while (true) {
   double[] a = Synth.squareWave(55.0, 0.25, 1.0);
   double[] b = Synth.fadeOut(a, 5.0);
   StdAudio.play(b);
}
```

Libraries and clients: quiz 4

What sound does *StdAudio.play*(*mystery*(5.0)) **produce?**

A. 5 seconds of concert A (440 Hz).

B. 5 seconds of a random frequency.

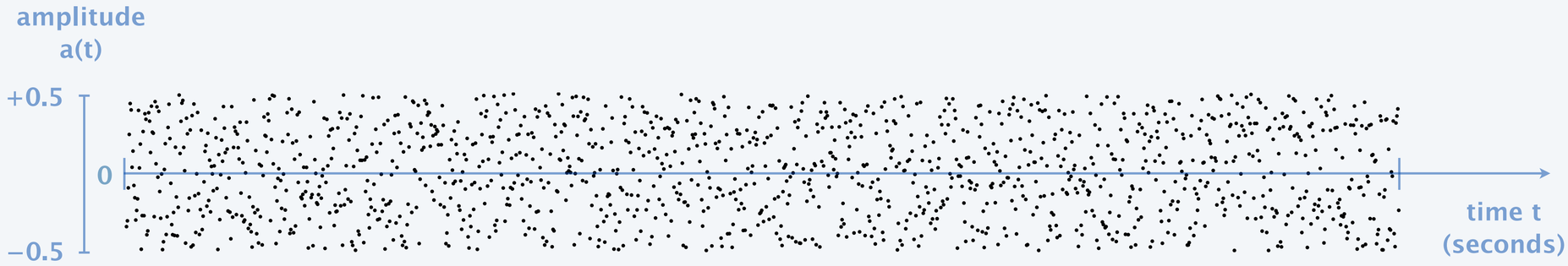
C. 5 seconds of silence.

D. 5 seconds of static.

```
public static double[] mystery(double duration) {
   int n = numberOfSamples(duration);
   double[] a = new double[n];
   for (int i = 0; i < n; i++) {
      a[i] = StdRandom.uniformDouble(-0.5, 0.5);
   }
   return a;
}
```

White noise

White noise. Samples are uniformly random values.



implementation

```
public static double[] whiteNoise(double amplitude, double duration) {
   int n = numberOfSamples(duration);
   double[] a = new double[n];
   for (int i = 0; i < n; i++) {
      a[i] = StdRandom.uniformDouble(-amplitude, +amplitude);
   }
   return a;
}
```

client

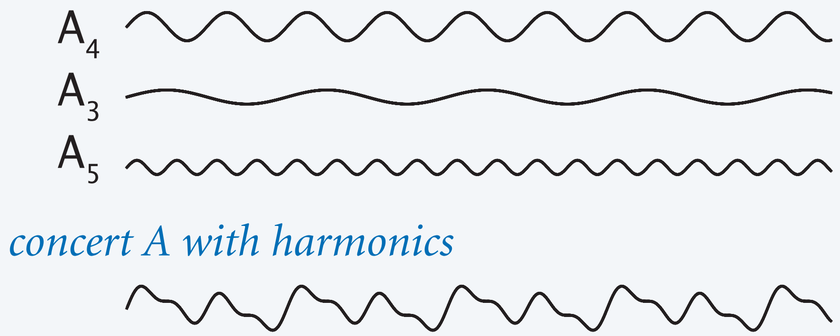
```
while (true) {
   double[] a = Synth.whiteNoise(0.5, 1.0);
   double[] b = Synth.fadeOut(a, 20.0);
   StdAudio.play(b);
}
```

Superposition

Superposition. To combine two (or more) audio signals, add the corresponding samples.

Ex 1. Harmonics.

```
double duration = 5.0;
double[] a4 = Synth.sineWave(440.0, 0.50, duration);
double[] a3 = Synth.sineWave(220.0, 0.25, duration);
double[] a5 = Synth.sineWave(880.0, 0.25, duration);
double[] harmonics = Synth.superpose(a4, a3, a5);
StdAudio.play(harmonics);
```



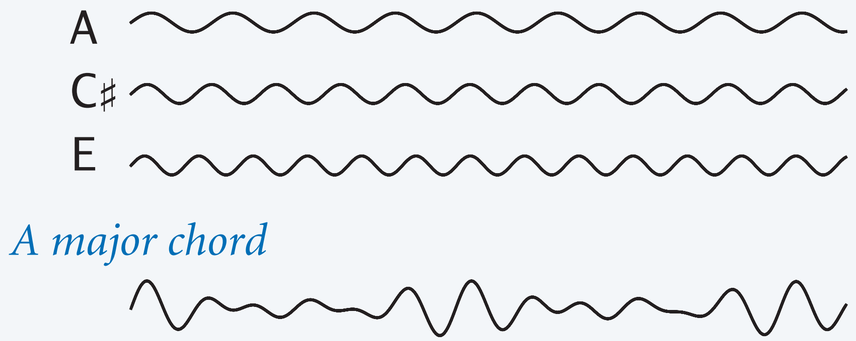
Superposition

Superposition. To combine two (or more) audio signals, add the corresponding samples.

Ex 1. Harmonics.

Ex 2. Chord.

```
double duration = 5.0;
double[] a4 = Synth.sineWave(440.00, 0.33, duration);
double[] c5 = Synth.sineWave(554.37, 0.33, duration);
double[] e5 = Synth.sineWave(659.26, 0.33, duration);
double[] chord = Synth.superpose(a4, c5, e5);
StdAudio.play(chord);
```



Superposition

Superposition. To combine two (or more) audio signals, add the corresponding samples.

Ex 1. Harmonics.

Ex 2. Chord.

Ex 3. Supersaw.

```
double freq = 220.0;
double amplitude = 0.05;
double duration = 10.0;
double[] a0 = Synth.sawWave(freq,         amplitude, duration);
double[] a1 = Synth.sawWave(freq - 0.191, amplitude, duration);
double[] a2 = Synth.sawWave(freq - 0.109, amplitude, duration);
double[] a3 = Synth.sawWave(freq - 0.037, amplitude, duration);
double[] a4 = Synth.sawWave(freq + 0.031, amplitude, duration);
double[] a5 = Synth.sawWave(freq + 0.107, amplitude, duration);
double[] a6 = Synth.sawWave(freq + 0.181, amplitude, duration);
double[] supersaw = Synth.superpose(a0, a1, a2, a3, a4, a5, a6);
StdAudio.play(supersaw);
```

"detuned" frequencies

Slay that tune

Goal. Play that tune, but with a supersaw.

client

```
public class SlayThatTune {
   public static void main(String[] args) {
      double amplitude = 0.5;
      while (!StdIn.isEmpty()) {
         int midi = StdIn.readInt();
         double duration = StdIn.readDouble();
         double freq = Synth.midiToFrequency(midi - 12);
         double[] a = Synth.supersawWave(freq, amplitude, duration);
         StdAudio.play(a);
      }
   }
}
```

transpose one octave lower

```
~/cos126/libraries> java-introcs SlayThatTune < Arpeggio.txt
[plays arpeggio]

~/cos126/libraries> java-introcs SlayThatTune < AxelF.txt
[plays beginning of Axel F]
```

Synth library

implementation

```
public class Synth {
   public static final double CONCERT_A = 440.0;

   public static int    numberOfSamples(double duration) { ... }
   public static double midiToFrequency(int midi)        { ... }

   private static double   sine(double freq, double t)  { ... }
   private static double square(double freq, double t)  { ... }
   private static double    saw(double freq, double t)  { ... }

   public static double[]     sineWave(double freq, double amplitude, double duration) { ... }
   public static double[]   squareWave(double freq, double amplitude, double duration) { ... }
   public static double[]      sawWave(double freq, double amplitude, double duration) { ... }
   public static double[] supersawWave(double freq, double amplitude, double duration) { ... }
   public static double[]   whiteNoise(             double amplitude, double duration) { ... }

   public static double[] superpose(double[] a, double[] b)          { ... }
   public static double[]  modulate(double[] a, double[] b)          { ... }
   public static double[]    fadeIn(double[] a, double lambda)       { ... }
   public static double[]   fadeOut(double[] a, double lambda)       { ... }

   public static void main(String[] args) { ... }

}
```

utility methods

private helper methods

create sound waves

manipulate sound waves

Summary

API. Defines method headers and behavior for a library.

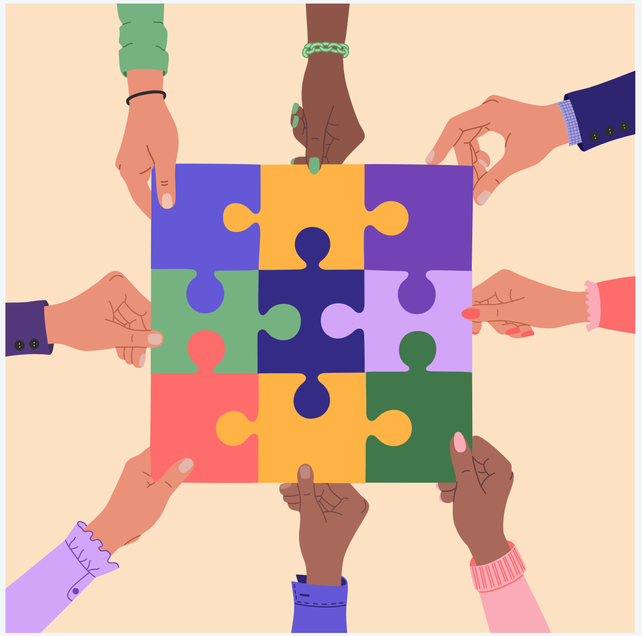
Client. Program that calls a library's methods.

Implementation. Program that implements the library's functionality.

Encapsulation. Separating clients from implementation details by hiding information.

Benefits.

- Reusable libraries.
- Independent development of small programs.
- Collaboration with a team of programmers.



Sound synthesis. You can write programs to synthesize sound.

Credits

media	source	license
Zhongshuge bookstore	Feng Shao / X+Living	
Random Number	xkcd	CC BY-NC 2.5
Coin Toss	Adobe Stock	education license
Ten-Sided Die	Adobe Stock	education license
Normal Distribution	Adobe Stock	education license
Shuffle Icon	Adobe Stock	education license
Client Avatars	Adobe Stock	education license
Rock, Paper, Scissors	Adobe Stock	education license
Cloud Coding Icon	Adobe Stock	education license
Contract Icon	Adobe Stock	education license
Implementation Icon	Adobe Stock	education license
Family 1 *Watching TV*	Adobe Stock	education license
Family 2 *Watching TV*	Adobe Stock	education license
Family 3 *Watching TV*	Adobe Stock	education license

Credits

media	source	license
TV Remote Control	Adobe Stock	education license
Pink Vintage TV	Adobe Stock	education license
Bravia TV	Sony	
Pharmacy Pills	Adobe Stock	education license
Private Sign on a Door	Adobe Stock	education license
Sound Waves and the Ear	Wikimedia	CC BY 4.0
Piano Keys	Adobe Stock	education license
Mosquito Alarm	Wikipedia	public domain
Yamaha DX7	Wikimedia	public domain
R2–D2 Sound Effects	Star Wars	
Axel F	Harold Faltemeyer	
Piano Keys	Adobe Stock	education license
Human Hands with Puzzle	Adobe Stock	education license