2.2 Libraries and Clients

- random number library
- designing libraries
- sound synthesis
- synthesizer library
Basic building blocks for programming

any program you might want to write

objects

functions  libraries

arrays

conditionals  loops

Math  text I/O

primitive data types  assignment statements

build reusable libraries
2.2 Libraries and Clients

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

```java
int getRandomNumber() {
    return 4; // chosen by fair dice roll.
    // guaranteed to be random.
}
```

https://introcs.cs.princeton.edu

https://xkcd.com/221/
Goal. Design a library to generate pseudo-random numbers.

```java
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]
    };
```

[Image with text: Standard random library]

Robert Sedgewick and Kevin Wayne
Standard random implementation: random numbers from various distributions

```java
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)
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;
    }

    ...
}

Standard random implementation: random numbers from a Gaussian distribution

can call a method without knowing how it is implemented

μ σ
Standard random implementation: shuffling the elements in an array

```java
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);
        }
    }

    :
}
```

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

calls a private method
(in the same 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

```java
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();
    }
}
```

StdRandom client 2

```java
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);
        }
    }
}
```

```bash
~/cos26/libraries> java-introcs Shuffle A B C D E E A D B C

~/cos26/libraries> java-introcs Shuffle A B C D E C A E B D

~/cos26/libraries> java-introcs Shuffle 2C 2D 2H ... AS 4S 2D AC 9H QH 8C ... JS 4H 2S

~/cos26/libraries> java-introcs RandomPoints 100000
```
2.2 **Libraries and Clients**

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

<table>
<thead>
<tr>
<th>library</th>
<th>description</th>
<th>example method call</th>
<th>source</th>
<th>logo</th>
</tr>
</thead>
<tbody>
<tr>
<td>StdRandom</td>
<td>generate random numbers</td>
<td>StdRandom.uniformInt(6)</td>
<td>textbook</td>
<td><img src="image" alt="logo" /></td>
</tr>
<tr>
<td>StdDraw</td>
<td>draw geometric shapes</td>
<td>StdDraw.circle(0.5, 0.5, 0.25)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Math</td>
<td>compute mathematical functions</td>
<td>Math.sqrt(2.0)</td>
<td>Java system</td>
<td><img src="image" alt="logo" /></td>
</tr>
<tr>
<td>java.util.Arrays</td>
<td>manipulate arrays</td>
<td>Arrays.sort(a)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Gaussian</td>
<td>compute Gaussian pdf and cdf</td>
<td>Gaussian.pdf(3.0)</td>
<td>user-defined</td>
<td><img src="image" alt="logo" /></td>
</tr>
<tr>
<td>SayNumber</td>
<td>speak numbers</td>
<td>SayNumber.sayInteger(126)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>:</td>
<td>:</td>
<td></td>
<td>:</td>
</tr>
</tbody>
</table>
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.
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.
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.

```
~/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
```

adds stdlib.jar to Java classpath

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

...
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 programatically.

```java
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
}
```

```
~/.cos126/libraries> java-introcs StdRandom 5
85.06009 false 8 8.88418
22.97440 true 40 9.18536
19.46492 false 28 8.89026
53.62835 true 90 8.90420
85.72239 false 5 8.78333
...
```

executes main() defined in this class
looks plausible (between 0 and 99)
Method header comments

**Best practice.** Every method should include a comment before the method header.

- Describe its purpose.
- Use names of parameter variables in description.
- Identify parameters, return value, and exceptions using Javadoc tags.

```java
/**
 * Returns a random integer uniformly in [0, n).
 * @param n number of possible integers
 * @return a random integer uniformly between 0 (inclusive) and n (exclusive)
 */
public static int uniformInt(int n) {
    return (int) (Math.random() * n);
}
```
Class StdRandom

public final class StdRandom
extends Object

Overview. The StdRandom class provides static methods for generating random number from various discrete and continuous distributions, including uniform, Bernoulli, geometric, Gaussian, exponential, Pareto, Poisson, and Cauchy. It also provides method for shuffling an array or subarray and generating random permutations.

Conventions. By convention, all intervals are half open. For example, uniformDouble(-1.0, 1.0) returns a random number between -1.0 (inclusive) and 1.0 (exclusive). Similarly, shuffle(a, lo, hi) shuffles the hi - lo elements in the array a[], starting at index lo (inclusive) and ending at index hi (exclusive).

Performance. The methods all take constant expected time, except those that involve arrays. The shuffle method takes time linear in the subarray to be shuffled; the discrete methods take time linear in the length of the argument array.

Additional information. For additional documentation, see Section 2.2 of Computer Science: An Interdisciplinary Approach by Robert Sedgewick and Kevin Wayne.

Author:
Robert Sedgewick, Kevin Wayne
2.2 Libraries and Clients

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

\[ y(t) = \sin(2\pi \cdot 2048 \cdot t), \quad 0 \leq t \leq T \]
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.

Review: audio sampling

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

\[ y(t) = \sin (2\pi \cdot 2048 \cdot t), \quad 0 \leq t \leq T \]

\[ a(t) = \sin (2\pi \cdot 2048 \cdot t), \quad t = \frac{0}{44100}, \frac{1}{44100}, \frac{2}{44100}, \ldots \]
Review: standard audio API

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

```java
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

```java
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;
    }

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

Sample at `n` equally spaced points:

\[ a(t) = A \sin \left( 2\pi f t \right), \quad t = \frac{0}{44100}, \frac{1}{44100}, \frac{2}{44100}, \ldots \]
What sound will the following code fragment produce?

```java
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.
- **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.

\[
\text{note} \quad | \quad \text{MIDI} \ (m) \quad | \quad \text{frequency (Hz)} \ (440 \times 2^{(m-69)/12}) \quad | \quad \text{sine wave}
\]

<table>
<thead>
<tr>
<th>note</th>
<th>MIDI \ (m)</th>
<th>frequency (Hz)</th>
<th>sine wave</th>
</tr>
</thead>
<tbody>
<tr>
<td>A₄</td>
<td>69</td>
<td>440</td>
<td></td>
</tr>
<tr>
<td>A₆ / B♭</td>
<td>70</td>
<td>466.16</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>71</td>
<td>493.88</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>72</td>
<td>523.25</td>
<td></td>
</tr>
<tr>
<td>C₆ / D♭</td>
<td>73</td>
<td>554.37</td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>74</td>
<td>587.33</td>
<td></td>
</tr>
<tr>
<td>D₆ / E♭</td>
<td>75</td>
<td>622.25</td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>76</td>
<td>659.26</td>
<td></td>
</tr>
<tr>
<td>F</td>
<td>77</td>
<td>698.46</td>
<td></td>
</tr>
<tr>
<td>F₆ / G♭</td>
<td>78</td>
<td>739.99</td>
<td></td>
</tr>
<tr>
<td>G</td>
<td>79</td>
<td>783.99</td>
<td></td>
</tr>
<tr>
<td>G₆ / A♭</td>
<td>80</td>
<td>830.61</td>
<td></td>
</tr>
<tr>
<td>A₅</td>
<td>81</td>
<td>880</td>
<td></td>
</tr>
</tbody>
</table>
Which of the following converts from MIDI note number to frequency?

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

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

C. Both A and B.

D. Neither A nor B.
Goal. Add methods (and constants) to library that many clients might want to use.


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.

```java
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);
    }
}
```

implementation

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

```java
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);
        }
    }
}
```

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

~`/cos126/libraries> java-intros 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
```

![MIDI note numbers and durations](image)

```
~/cos126/libraries> java-introcs PlayThatTune < MajorScaleC.txt
[plays C major scale]
```
**Goal.** Read in a sequence of MIDI note numbers and durations from standard input, and play the synthesized results to standard audio.

```java
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);
        }
    }
}
```

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

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

~/cos126/libraries> java-intros PlayThatTune < FurElise.txt [plays beginning of Fur Elise]
```
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, ...).
- ...
**Synthesizer API**

*Synth.* A library for synthesizing sound.

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

\[
a(t) = \text{sgn} \left( \sin \left( 2\pi \cdot 440 \cdot t \right) \right), \quad 0 \leq t \leq T
\]

\[
\text{sgn}(x) = \begin{cases} 
-1 & \text{if } x < 0 \\
0 & \text{if } x = 0 \\
+1 & \text{if } x > 0
\end{cases}
\]

**implementation**

```java
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 $f$.

\[
a(t) = 2 \left( 440t - \left\lfloor 440t + \frac{1}{2} \right\rfloor \right), \quad 0 \leq t \leq T
\]

**Implementation**

```java
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() */
}
```
Sound envelope. Defines how a sound changes over time.

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

\[ a(t) = \sin(2\pi \cdot 30 \cdot t) \]

\[ a(t) = \sin(2\pi \cdot 30 \cdot t) \cdot 2^{-10t} \]
Exponential fade

```java
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;
    }
}
```

Implementation

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

double[] a = Synth.squareWave(55.0, 0.25, 1.0);
double[] b = Synth.fadeOut(a, 5.0);
StdAudio.play(b);
```
Libraries and clients: quiz 3

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.

```java
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.uniform(-0.5, 0.5);
    }
    return a;
}
```
White noise

White noise. Samples are uniformly random values.

```
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;
}
```

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.

```c
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);
```

concert A with harmonics
Superposition

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

Ex 1. Harmonics.
Ex 2. Chord.

```java
double duration = 5.0;
double[] a = Synth.sineWave(440.00, 0.33, duration);
double[] c = Synth.sineWave(554.37, 0.33, duration);
double[] e = Synth.sineWave(659.26, 0.33, duration);
double[] chord = Synth.superpose(a, c, e);
StdAudio.play(chord);
```

A major chord
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;  "detuned" frequencies

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);
**Goal.** Play that tune, but with a supersaw.

```java
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);
        }
    }
}
```

```bash
~/cos126/libraries> java-introcs SlayThatTune < Arpeggio.txt
(plays arpeggio)
```

```bash
~/cos126/libraries> java-introcs SlayThatTune < AxelF.txt
(plays beginning of Axel F)
```
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) { ... }
}
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.
<table>
<thead>
<tr>
<th>media</th>
<th>source</th>
<th>license</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zhongshuge bookstore</td>
<td>Feng Shao / X+Living</td>
<td></td>
</tr>
<tr>
<td>Random Number</td>
<td>xkcd</td>
<td>CC BY-NC 2.5</td>
</tr>
<tr>
<td>Coin Toss</td>
<td>Adobe Stock</td>
<td>education license</td>
</tr>
<tr>
<td>Ten-Sided Die</td>
<td>Adobe Stock</td>
<td>education license</td>
</tr>
<tr>
<td>Normal Distribution</td>
<td>Adobe Stock</td>
<td>education license</td>
</tr>
<tr>
<td>Shuffle Icon</td>
<td>Adobe Stock</td>
<td>education license</td>
</tr>
<tr>
<td>Client Avatars</td>
<td>Adobe Stock</td>
<td>education license</td>
</tr>
<tr>
<td>Cloud Coding Icon</td>
<td>Adobe Stock</td>
<td>education license</td>
</tr>
<tr>
<td>Contract Icon</td>
<td>Adobe Stock</td>
<td>education license</td>
</tr>
<tr>
<td>Implementation Icon</td>
<td>Adobe Stock</td>
<td>education license</td>
</tr>
<tr>
<td>Family 1 Watching TV</td>
<td>Adobe Stock</td>
<td>education license</td>
</tr>
<tr>
<td>Family 2 Watching TV</td>
<td>Adobe Stock</td>
<td>education license</td>
</tr>
<tr>
<td>Family 3 Watching TV</td>
<td>Adobe Stock</td>
<td>education license</td>
</tr>
<tr>
<td>media</td>
<td>source</td>
<td>license</td>
</tr>
<tr>
<td>------------------------------</td>
<td>-------------</td>
<td>------------------------</td>
</tr>
<tr>
<td>TV Remote Control</td>
<td>Adobe Stock</td>
<td>education license</td>
</tr>
<tr>
<td>Pink Vintage TV</td>
<td>Adobe Stock</td>
<td>education license</td>
</tr>
<tr>
<td>Bravia TV</td>
<td>Sony</td>
<td></td>
</tr>
<tr>
<td>Pharmacy Pills</td>
<td>Adobe Stock</td>
<td>education license</td>
</tr>
<tr>
<td>Private Sign on a Door</td>
<td>Adobe Stock</td>
<td>education license</td>
</tr>
<tr>
<td>Sound Waves and the Ear</td>
<td>Wikimedia</td>
<td>CC BY 4.0</td>
</tr>
<tr>
<td>Piano Keys</td>
<td>Adobe Stock</td>
<td>education license</td>
</tr>
<tr>
<td>Yamaha DX7</td>
<td>Wikimedia</td>
<td>public domain</td>
</tr>
<tr>
<td>R2–D2 Sound Effects</td>
<td>Star Wars</td>
<td></td>
</tr>
<tr>
<td>Axel F</td>
<td>Harold Faltemeyer</td>
<td></td>
</tr>
<tr>
<td>Piano Keys</td>
<td>Adobe Stock</td>
<td>education license</td>
</tr>
<tr>
<td>Human Hands with Puzzle</td>
<td>Adobe Stock</td>
<td>education license</td>
</tr>
</tbody>
</table>