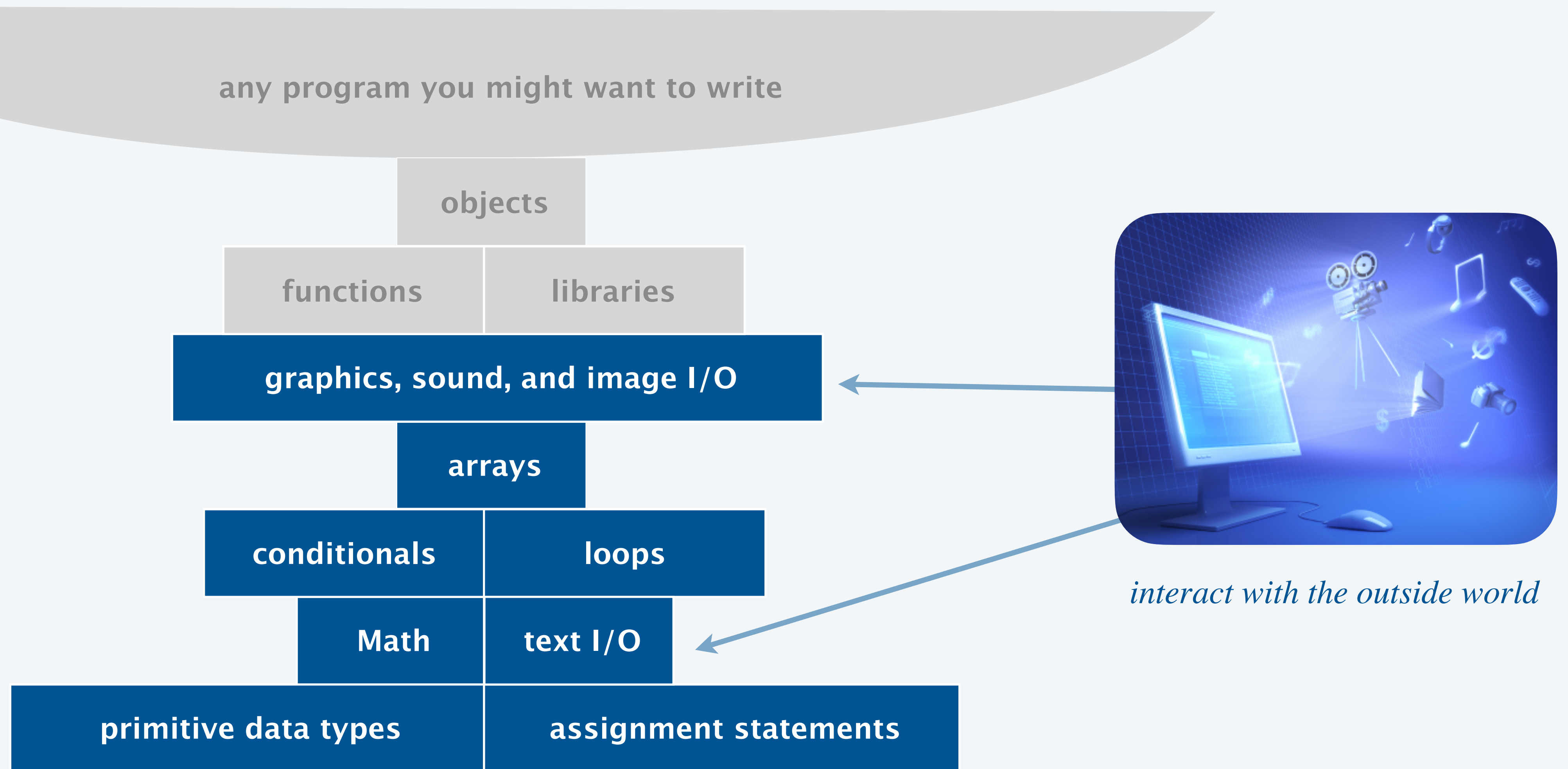


<https://introc.cs.princeton.edu>

1.5 INPUT AND OUTPUT

- ▶ *standard input and output*
- ▶ *redirection and piping*
- ▶ *standard drawing*
- ▶ *animation*

Basic building blocks for programming



Input and output

Goal. Write Java programs that interact with the outside world via input and output devices.

Input devices.



keyboard



trackpad



storage



network



webcam



microphone

Output devices.



video display



earbuds



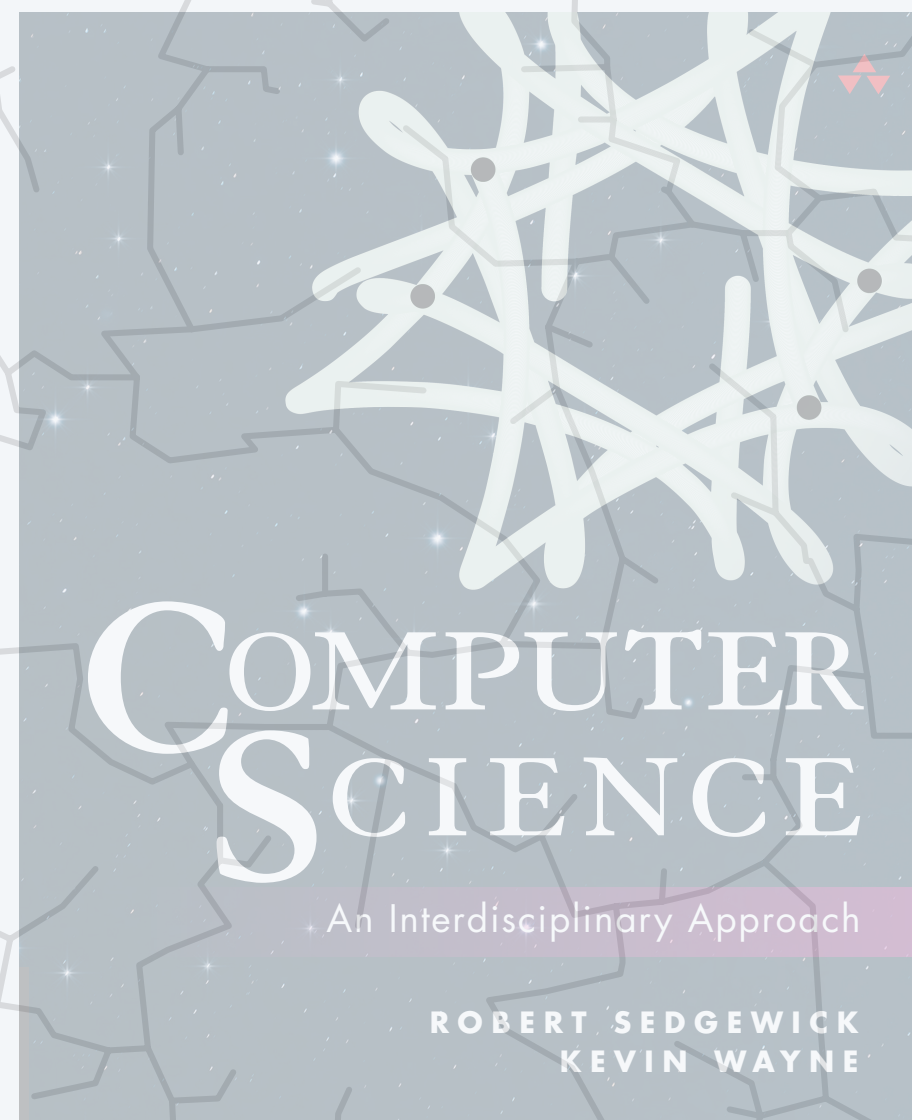
storage



network



braille display



<https://introc.cs.princeton.edu>

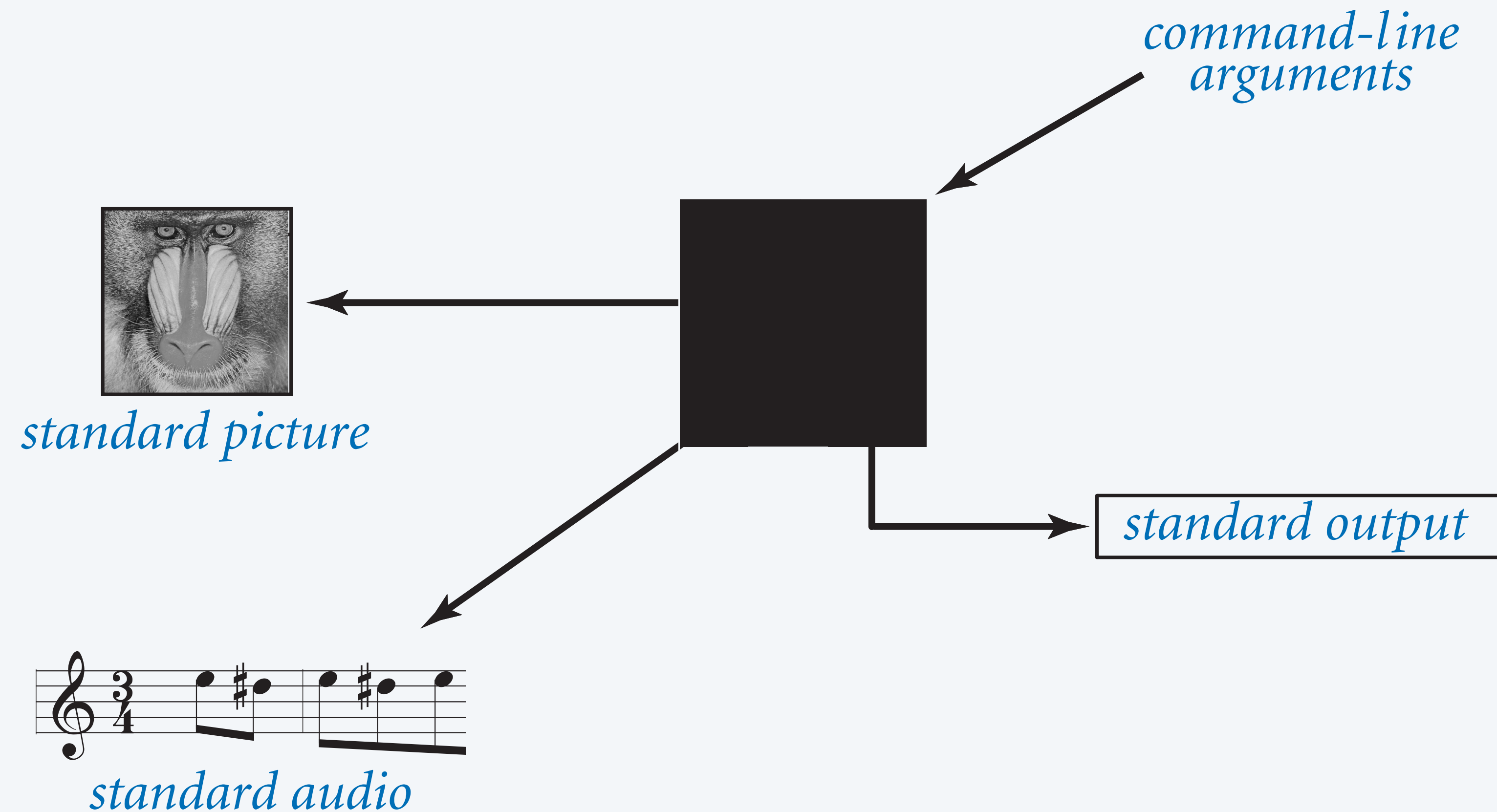
1.5 INPUT AND OUTPUT

- ▶ *standard input and output*
- ▶ *redirection and piping*
- ▶ *standard drawing*
- ▶ *animation*

Input-output abstractions (so far)

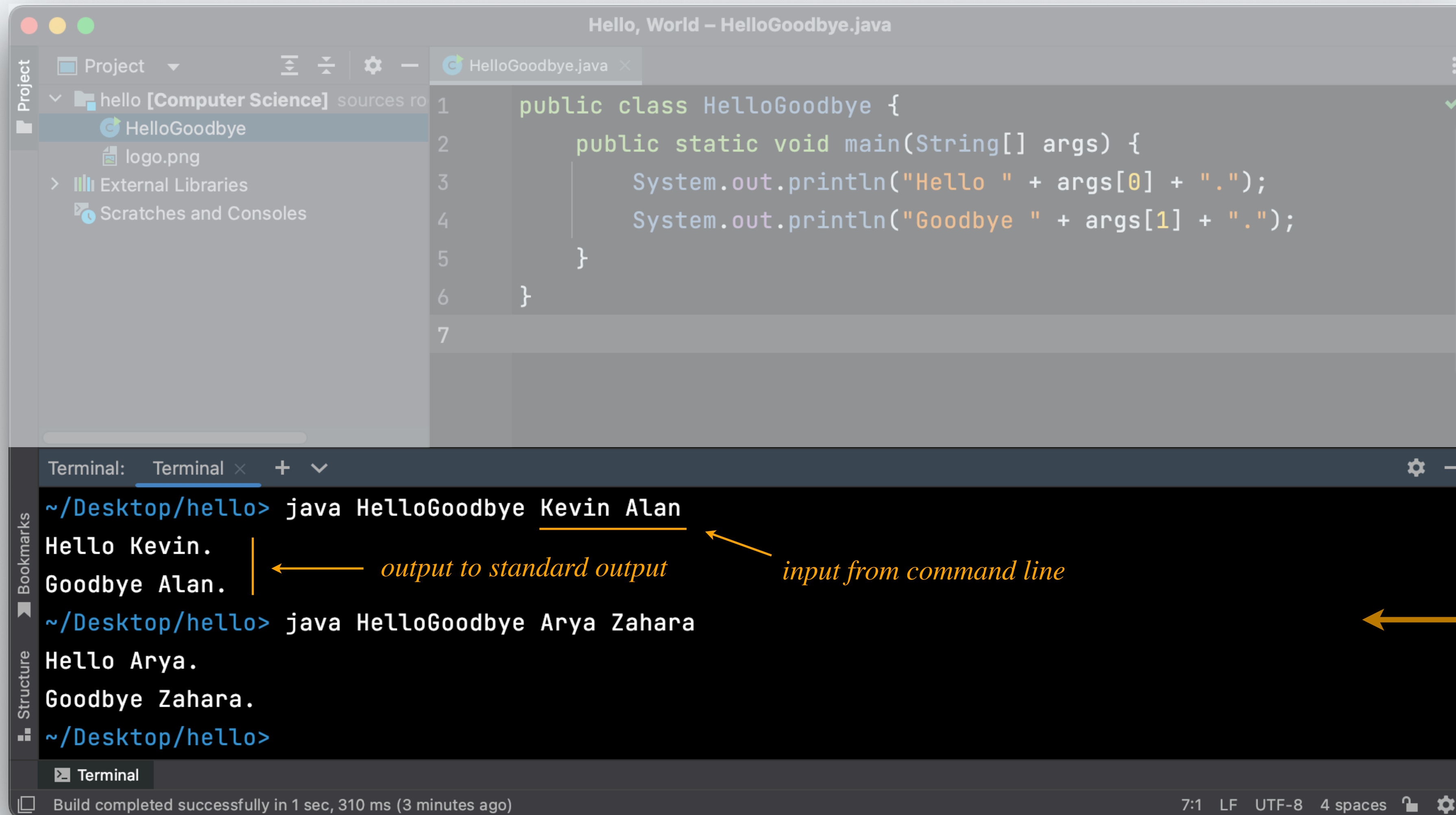
Our approach.

- Define input and output abstractions.
- Use operating system (OS) functionality to connect our Java programs to physical devices.



Review: terminal

Terminal. A text-based interface for interacting with programs, files, and devices.



The screenshot shows an IDE window titled "Hello, World - HelloGoodbye.java". The editor displays the following Java code:

```
1 public class HelloGoodbye {
2     public static void main(String[] args) {
3         System.out.println("Hello " + args[0] + ".");
4         System.out.println("Goodbye " + args[1] + ".");
5     }
6 }
7
```

Below the editor is a terminal window with the following output:

```
~/Desktop/hello> java HelloGoodbye Kevin Alan
Hello Kevin.
Goodbye Alan.
~/Desktop/hello> java HelloGoodbye Arya Zahara
Hello Arya.
Goodbye Zahara.
~/Desktop/hello>
```

Annotations in the terminal window:

- An arrow points from the text "output to standard output" to the first two lines of output ("Hello Kevin." and "Goodbye Alan.").
- An arrow points from the text "input from command line" to the command "java HelloGoodbye Kevin Alan".
- An arrow points from the text "VT-100 terminal emulator" to the terminal window.



VT-100 terminal emulator

Review: command-line arguments

Command-line arguments. Provide text input to a program.

Basic properties.

- Arguments provided to a program by typing after program name.
- Arguments provided to program *before* execution.
- Java: string arguments available in *main()* as *args[0]*, *args[1]*, ...

```
public class HelloGoodbye {  
    public static void main(String[] args) {  
        System.out.print("Hello ");  
        System.out.println(args[0] + ".");  
        System.out.print("Goodbye ");  
        System.out.println(args[1] + ".");  
    }  
}
```

```
~/cos126/io> java HelloGoodbye Kevin Alan  
Hello Kevin.  
Goodbye Alan.  
                                ↑  
                                command-line arguments  
  
~/cos126/io> java HelloGoodbye Arya Zahara  
Hello Arya.  
Goodbye Zahara.  
                                ↑           ↑  
                                args[0]   args[1]  
  
~/cos126/io> java HelloGoodbye Aðalbjörg 'Zu Sheng'  
Hello Aðalbjörg.  
Goodbye Zu Sheng.  
                                ↑  
                                use quotes
```

Review: standard output

Standard output stream. An abstraction for an output sequence of text.

Basic properties.

- The call `System.out.println()` appends text to standard output.
- By default, standard output is sent to the terminal/display.
- No limit on amount of output.

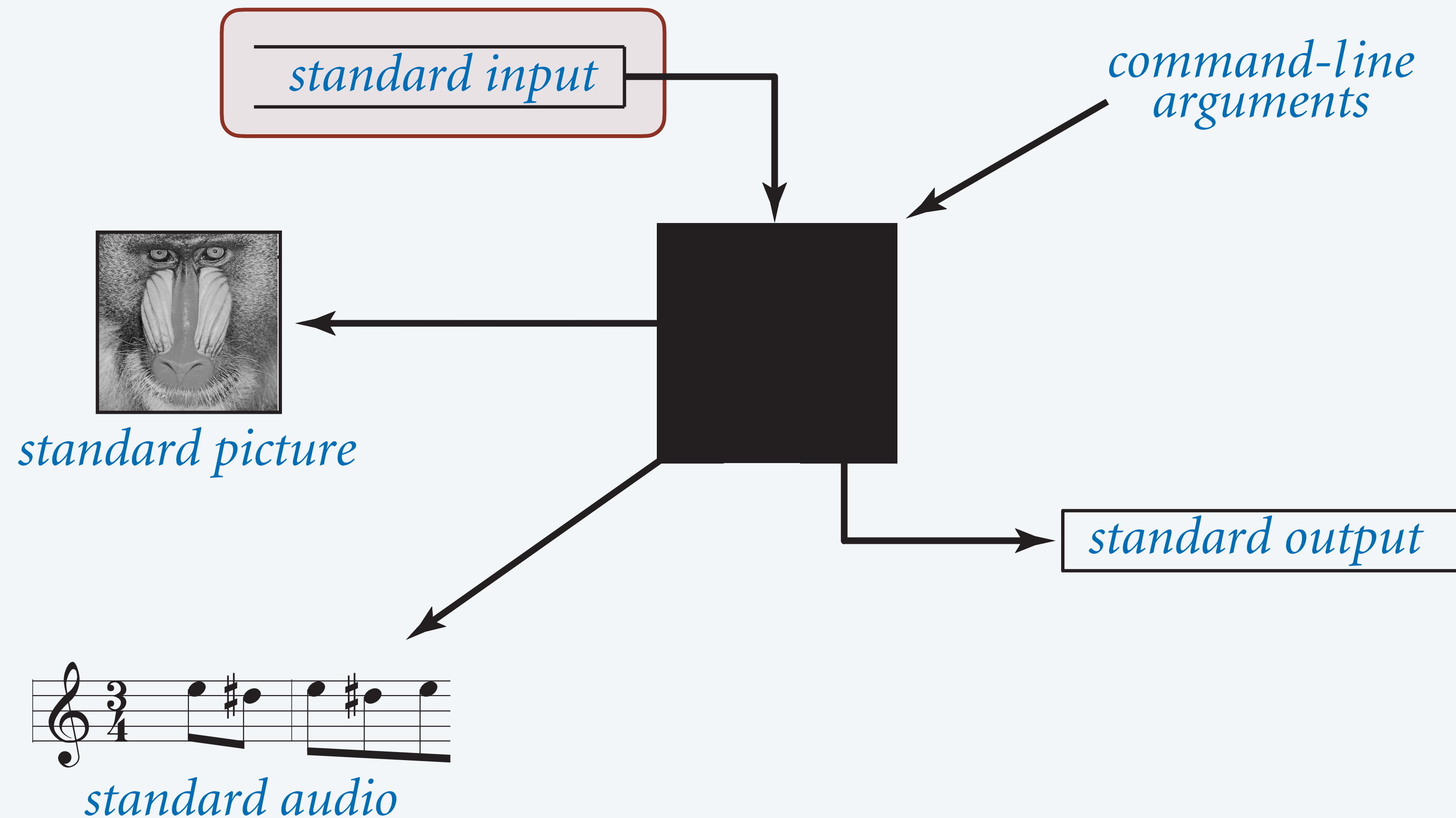
```
public class RandomUniform {  
    public static void main(String[] args) {  
        int n = Integer.parseInt(args[0]);  
        for (int i = 0; i < n; i++) {  
            System.out.println(Math.random());  
        }  
    }  
}
```

```
~/cos126/io> java RandomUniform 4  
0.9320744627218469  
0.4279508713950715  
0.08994615071160994  
0.6579792663546435  
  
~/cos126/io> java RandomUniform 100000  
0.09474882292442943  
0.2832974030384712  
0.1833964252856476  
0.2952177517730442  
0.8035985765979008  
...
```

← produces lots of output

Input-output abstractions (standard input)

Next step. Add a text **input** stream.

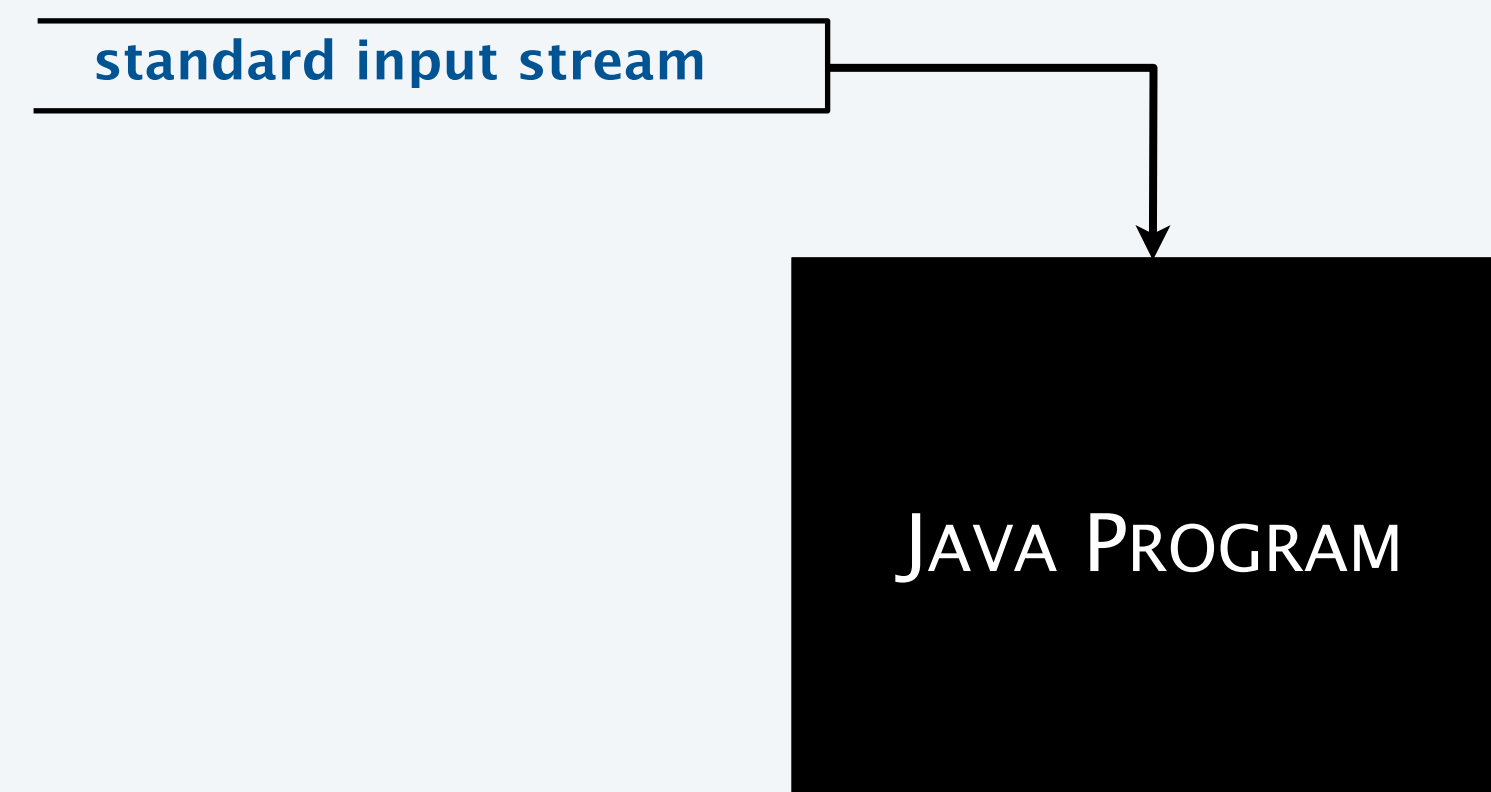


Standard input

Standard input stream. An abstraction for an input sequence of text.

Advantages over command-line arguments:

- No limit on the amount of input.
- Conversion to primitive types is explicitly handled.
- Can provide input interactively, *while* the program is executing.



StdIn. Our library for reading strings and numbers from standard input.

← available with `javac-introcs`
and `java-introcs` commands

<code>public class StdIn</code>	description
<code>static boolean isEmpty()</code>	<i>true if no more values, false otherwise</i>
<code>static int readInt()</code>	<i>read a value of type int</i>
<code>static double readDouble()</code>	<i>read a value of type double</i>
<code>static boolean readBoolean()</code>	<i>read a value of type boolean</i>
<code>static String readString()</code>	<i>read a value of type String</i>
<code>⋮</code>	<code>⋮</code>

← *read next token (sequence of non-whitespace characters)
and parse as specified type*

StdOut. Our library for printing strings and numbers to standard output.

available with `javac-introcs`
and `java-introcs` commands

<code>public class StdOut</code>	description
<code>static void print(String s)</code>	<i>print s on the output stream</i>
<code>static void println()</code>	<i>print a newline on the output stream</i>
<code>static void println(String s)</code>	<i>print s, then a newline on the stream</i>
<code>static void printf(String f, ...)</code>	<i>print formatted output</i>
<code>⋮</code>	<code>⋮</code>

Q. How different from `System.out.println()` ?

A. Mostly the same, but output is independent of OS and locale. ← *we'll use StdOut from now on*

Standard input warmup

Interactive user input. User can provide input **while** the program is running.

```
public class AddTwoInts {  
    public static void main(String[] args) {  
        StdOut.print("Type the first integer: ");  
        int a = StdIn.readInt();  
        StdOut.print("Type the second integer: ");  
        int b = StdIn.readInt();  
        int sum = a + b;  
        StdOut.println("Their sum is " + sum);  
    }  
}
```

Remark 1. By default, comes from keyboard/terminal.

Remark 2. Input and output can be interleaved.

Remark 3. Run-time exception if user supplies incompatible input.

```
~/cos126/io> java-introcs AddTwoInts  
Type the first integer: 1  
Type the second integer: 2  
Their sum is 3  
  
~/cos126/io> java-introcs AddTwoInts  
Type the first integer: 100  
Type the second integer: 26  
Their sum is 126  
  
~/cos126/io> java-introcs AddTwoInts  
Type the first integer: 100  
Type the second integer: twenty-six  
java.util.InputMismatchException: attempts  
to read an 'int' value from standard input,  
but the next token is "twenty-six"  
↑  
sequence of non-  
whitespace characters
```

Average the numbers on the standard input stream

Goal. Read a stream of numbers (from standard input) and print their average (to standard output).

```
public class Average {
    public static void main(String[] args) {
        double sum = 0.0; // cumulative total
        int n = 0; // number of values

        while (!StdIn.isEmpty()) {
            double x = StdIn.readDouble();
            sum = sum + x;
            n++;
        }

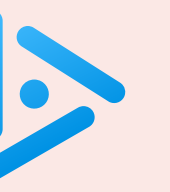
        StdOut.printf("%.4f\n", sum / n);
    }
}
```

formatted printing *4 digits after decimal point* *newline*

```
~/cos126/io> java-introcs Average
1.0
2.0
4.0
2.0
<Ctrl-D> ← signifies end of standard input
              (<Enter><Ctrl-Z><Enter> on Windows)
2.2500

~/cos126/io> java-introcs Average
10.0 5.0 6.0 3.0
7.0      32.0 ← values separated by whitespace
<Ctrl-D>
10.5000
```

Remark. No limit on amount of input. ← *“streaming algorithm” (avoids storing data)*

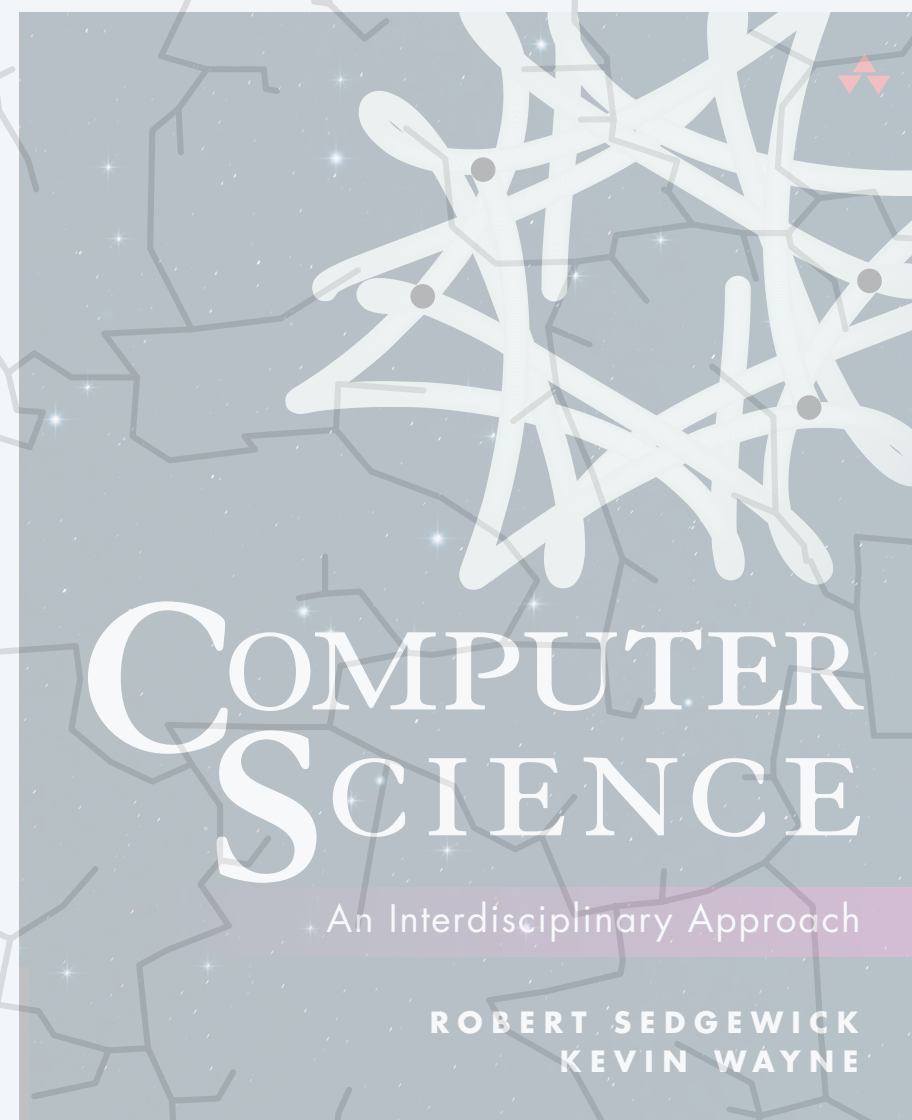


What does the following program do with the given input?

- A. Prints "X", "Y", and "Z".
- B. Run-time exception.
- C. Both A and B.
- D. Neither A nor B.

```
public class Mystery {  
    public static void main(String[] args) {  
        int n = args.length;  
        for (int i = 0; i < n; i++) {  
            String s = StdIn.readString();  
            StdOut.println(s);  
        }  
    }  
}
```

```
~/cos126/io> java-introcs Mystery A B C D E  
X Y Z  
<Ctrl-D>
```

<https://introc.cs.princeton.edu>

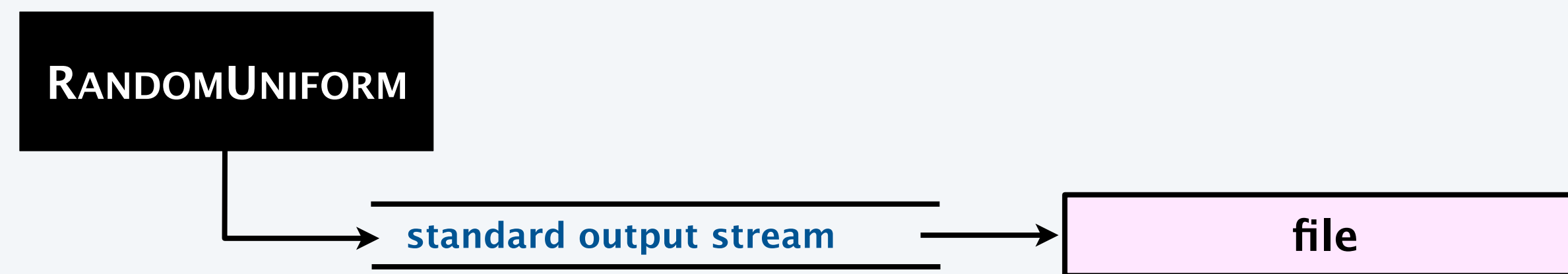
1.5 INPUT AND OUTPUT

- ▶ *standard input and output*
- ▶ *redirection and piping*
- ▶ *standard drawing*
- ▶ *animation*

Redirecting standard output

Terminal. By default, standard output is sent to the terminal/display.

Redirecting standard output. Send standard output to a **file** (instead of terminal/display).



```
~/cos126/io> java-introcs RandomUniform 1000000 > data.txt
[no output]

~/cos126/io> more data.txt
0.09474882292442943
0.2832974030384712
0.1833964252856476
0.2952177517730442
0.8035985765979008
...
```

display content of a file (arrow pointing to the output of the 'more' command)

redirect standard output (arrow pointing to the '>' symbol in the first command)

filename (arrow pointing to 'data.txt' in the first command)

Redirecting standard input

Terminal. By default, standard input comes from the terminal/keyboard.

Redirecting standard input. Read standard input from a **file** (instead of terminal/keyboard).



```
~/cos126/io> more data.txt
0.09474882292442943
0.2832974030384712
0.1833964252856476
0.2952177517730442
0.8035985765979008
...

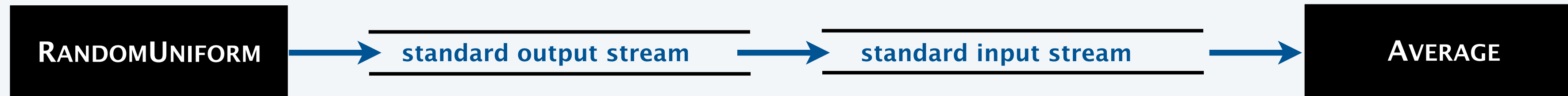
~/cos126/io> java-introcs Average < data.txt
0.4948
```

redirect
standard input *filename*

The terminal output shows the execution of the 'more' command on 'data.txt', displaying several floating-point numbers. Below this, the 'java-introcs Average < data.txt' command is shown, which outputs the average value '0.4948'. Two orange arrows point from the text 'redirect standard input' to the '<' symbol in the command, and another orange arrow points from 'filename' to 'data.txt'.

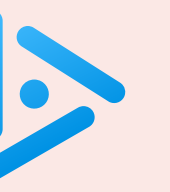
Piping

Piping. Connect standard output of one program to standard input of another program.



```
~/cos126/io> java-introcs RandomUniform 1000000 | java-introcs Average  
0.4998  
                                     ↑  
                                     pipe operator  
~/cos126/io> java-introcs RandomUniform 1000000 | java-introcs Average  
0.5002
```

Remark. No limit within programs on amount of data to process.



The macOS command `say` reads text from standard input and synthesizes it as audible speech. Which of the following commands will speak "Hello, World" ?

A.

```
~/cos126/io> say  
HelloWorld.java  
<Ctrl-D>
```

B.

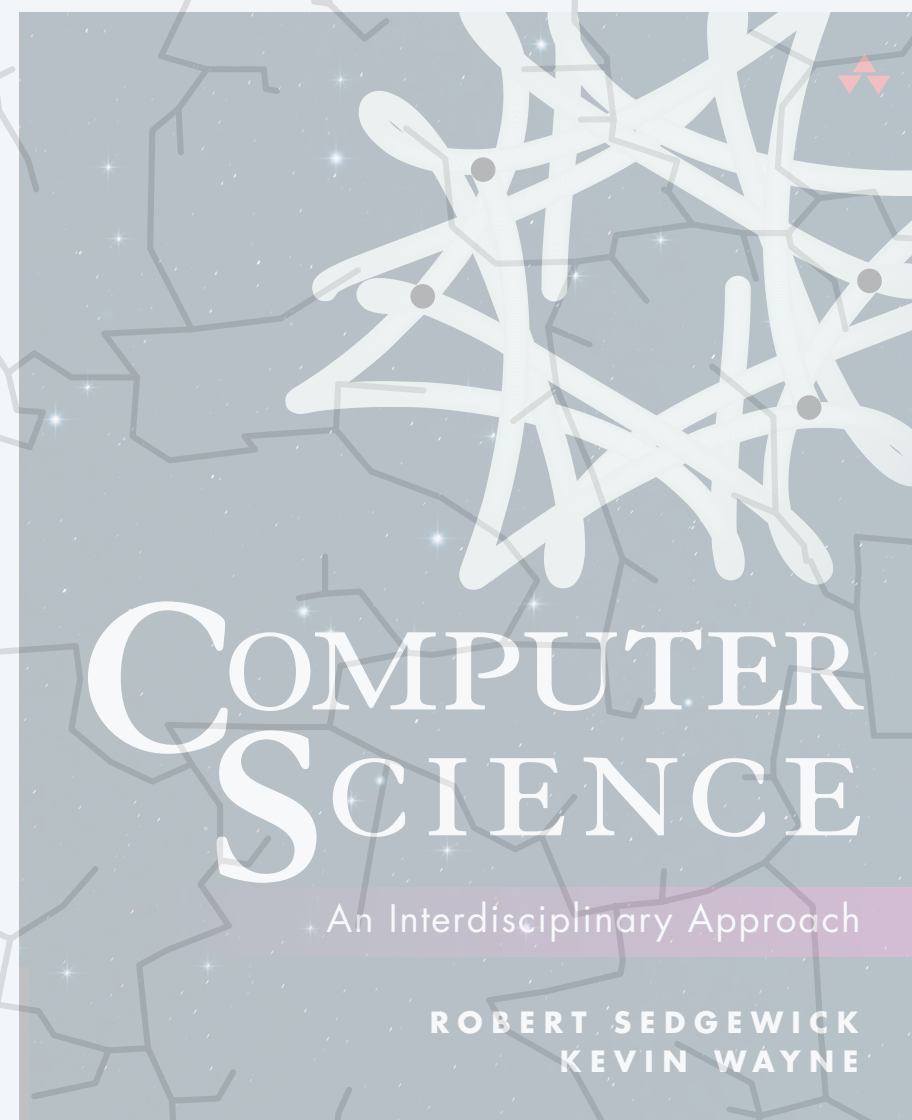
```
~/cos126/io> say > HelloWorld.java  
<Ctrl-D>
```

C.

```
~/cos126/io> java HelloWorld | say
```

D.

```
~/cos126/io> say < HelloWorld.java
```

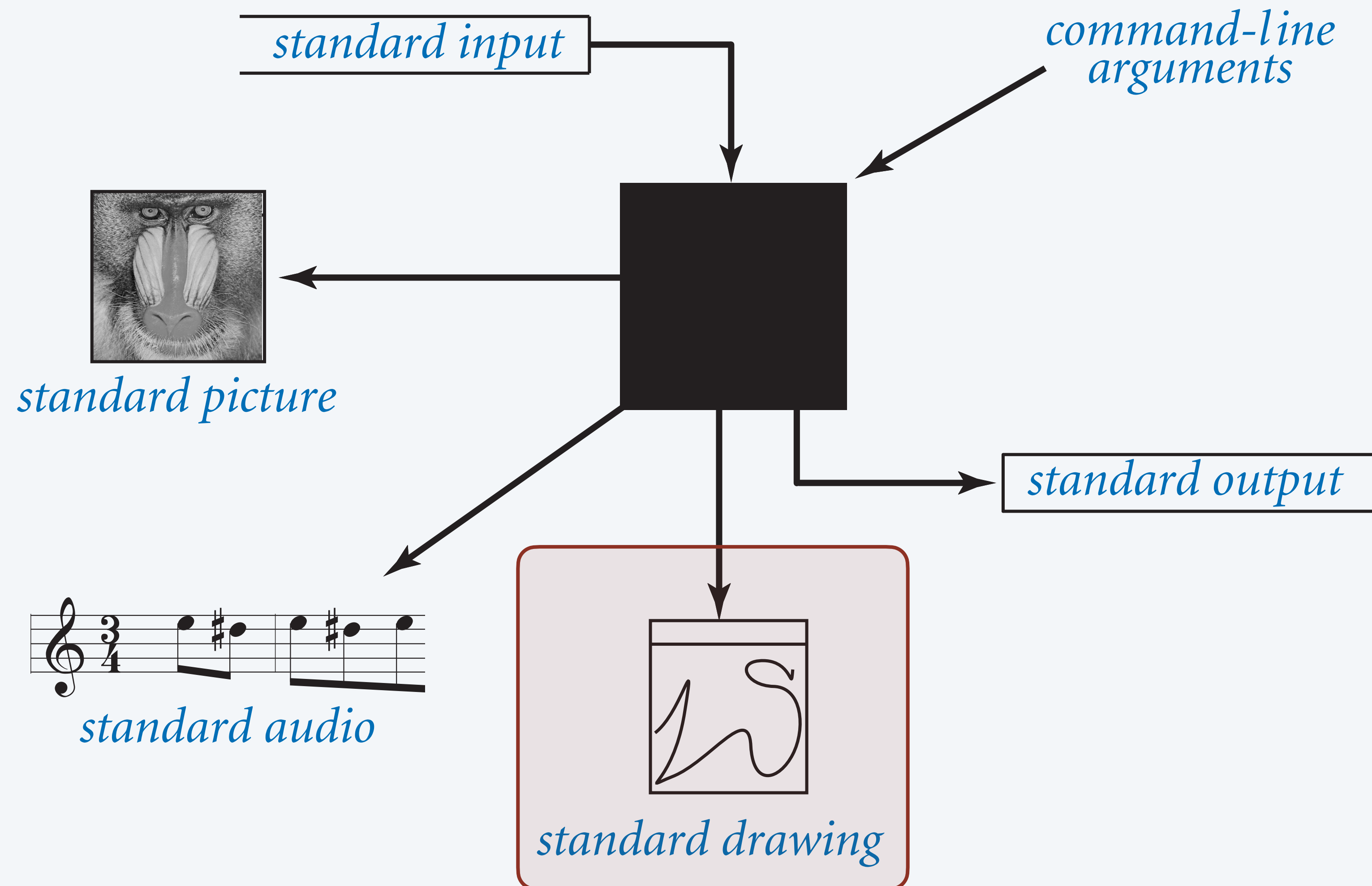
<https://introc.cs.princeton.edu>

1.5 INPUT AND OUTPUT

- ▶ *standard input and output*
- ▶ *redirection and piping*
- ▶ *standard drawing*
- ▶ *animation*

Input-output abstractions (standard drawing)

Next step. Add the ability to create a **drawing**.



StdDraw. Our library for drawing and animating **geometric shapes** in a graphical window.

↑
*use StdPicture library
to manipulate images*

← *available with javac-introcs
and java-introcs commands*

```
public class StdDraw
```

description

```
static void line(double x0, double y0, double x1, double y1)
```

draw line segment between (x_0, y_0) and (x_1, y_1)

```
static void point(double x, double y)
```

draw point (x, y)

```
static void circle(double x, double y, double r)
```

draw circle of radius r centered at (x, y)

```
static void square(double x, double y, double r)
```

draw square of half-width r centered at (x, y)

```
static void polygon(double[] x, double[] y)
```

draw polygon connecting points (x_i, y_i)

```
static void text(double x, double y, String text)
```

draw text, centered at (x, y)

```
static void picture(double x, double y, String filename)
```

draw GIF, JPG or PNG image, centered at (x, y)

⋮

⋮

StdDraw. Our library for drawing and animating **geometric shapes** in a graphical window.

<code>public class StdDraw</code>	description	default value
<code>static void setCanvasSize(int width, int height)</code>	<i>set the canvas size to width-by-height</i>	512-by-512
<code>static void setXscale(double x0, double x1)</code>	<i>set x-range to $[x_0, x_1]$</i>	[0, 1]
<code>static void setYscale(double y0, double y1)</code>	<i>set y-range to $[y_0, y_1]$</i>	[0, 1]
<code>static void setPenRadius(double radius)</code>	<i>set the pen radius to radius</i>	0.002
<code>static void setPenColor(Color color)</code>	<i>set the pen color to color</i>	<i>black</i>
<code>⋮</code>	<code>⋮</code>	

StdDraw.BLACK, StdDraw.WHITE,
StdDraw.GRAY, StdDraw.RED,
StdDraw.GREEN, StdDraw.BLUE,
StdDraw.PRINCETON_ORANGE, ...

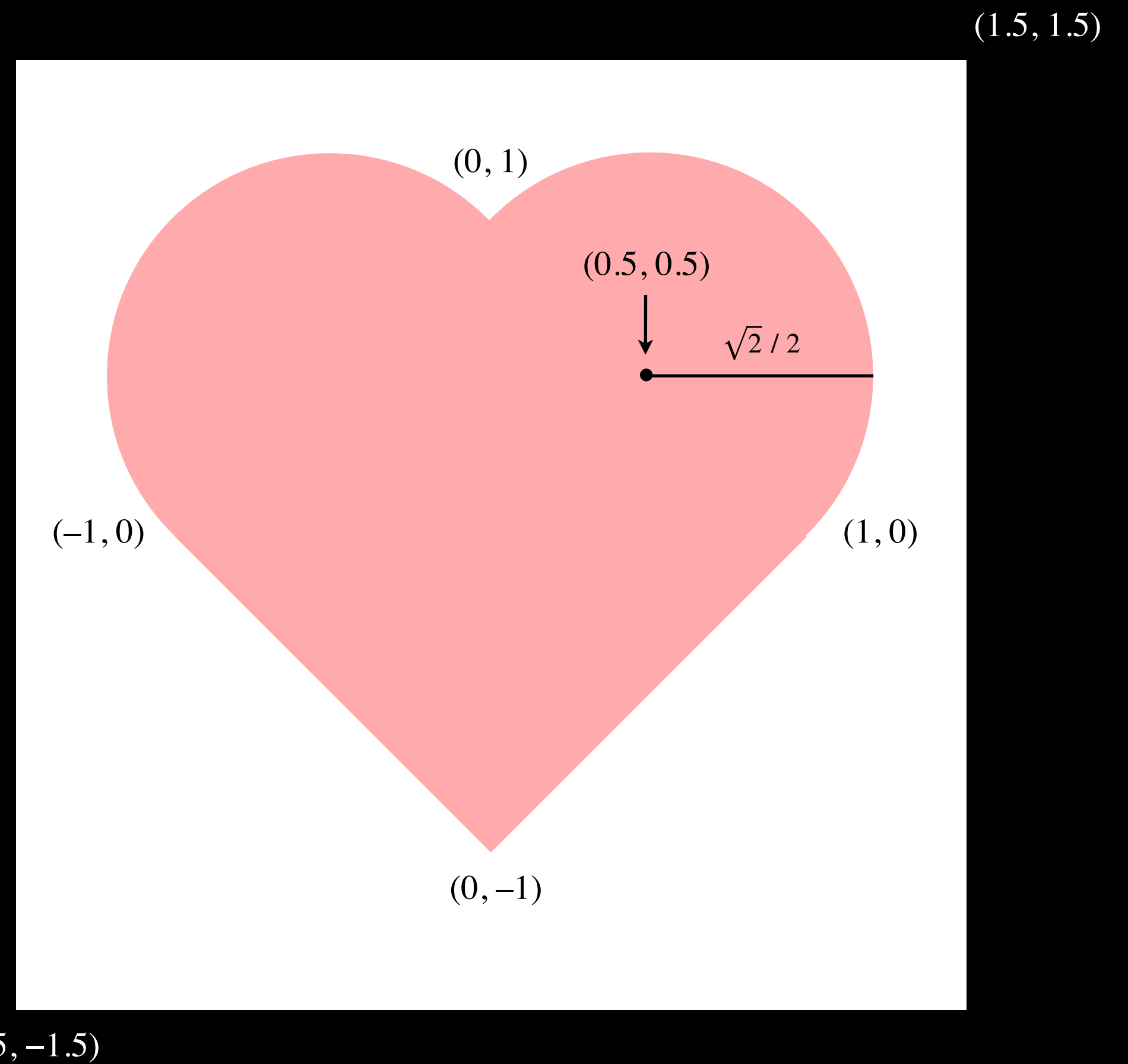
Your first drawing



Goal. Draw filled diamond and two filled circles.

```
public class Heart {  
    public static void main(String[] args) {  
        StdDraw.setXscale(-1.5, +1.5);  
        StdDraw.setYscale(-1.5, +1.5);  
        StdDraw.setPenColor(StdDraw.PINK);  
  
        // draw filled diamond  
        double[] xs = { -1, 0, 1, 0 };  
        double[] ys = { 0, -1, 0, 1 };  
        StdDraw.filledPolygon(xs, ys);  
  
        // draw two filled circles  
        double radius = Math.sqrt(2) / 2;  
        StdDraw.filledCircle(+0.5, 0.5, radius);  
        StdDraw.filledCircle(-0.5, 0.5, radius);  
    }  
}
```

```
~/cos126/io> java-introcs Heart
```



trace of drawing

Data visualization

Goal. Read points (from standard input) and plot.

```
public class PlotPoints {  
    public static void main(String[] args) {  
  
        double xmin = StdIn.readDouble();  
        double ymin = StdIn.readDouble();  
        double xmax = StdIn.readDouble();  
        double ymax = StdIn.readDouble();  
  
        StdDraw.setXscale(xmin, xmax);  
        StdDraw.setYscale(ymin, ymax);  
  
        while (!StdIn.isEmpty()) {  
            double x = StdIn.readDouble();  
            double y = StdIn.readDouble();  
            StdDraw.point(x, y);  
        }  
    }  
}
```

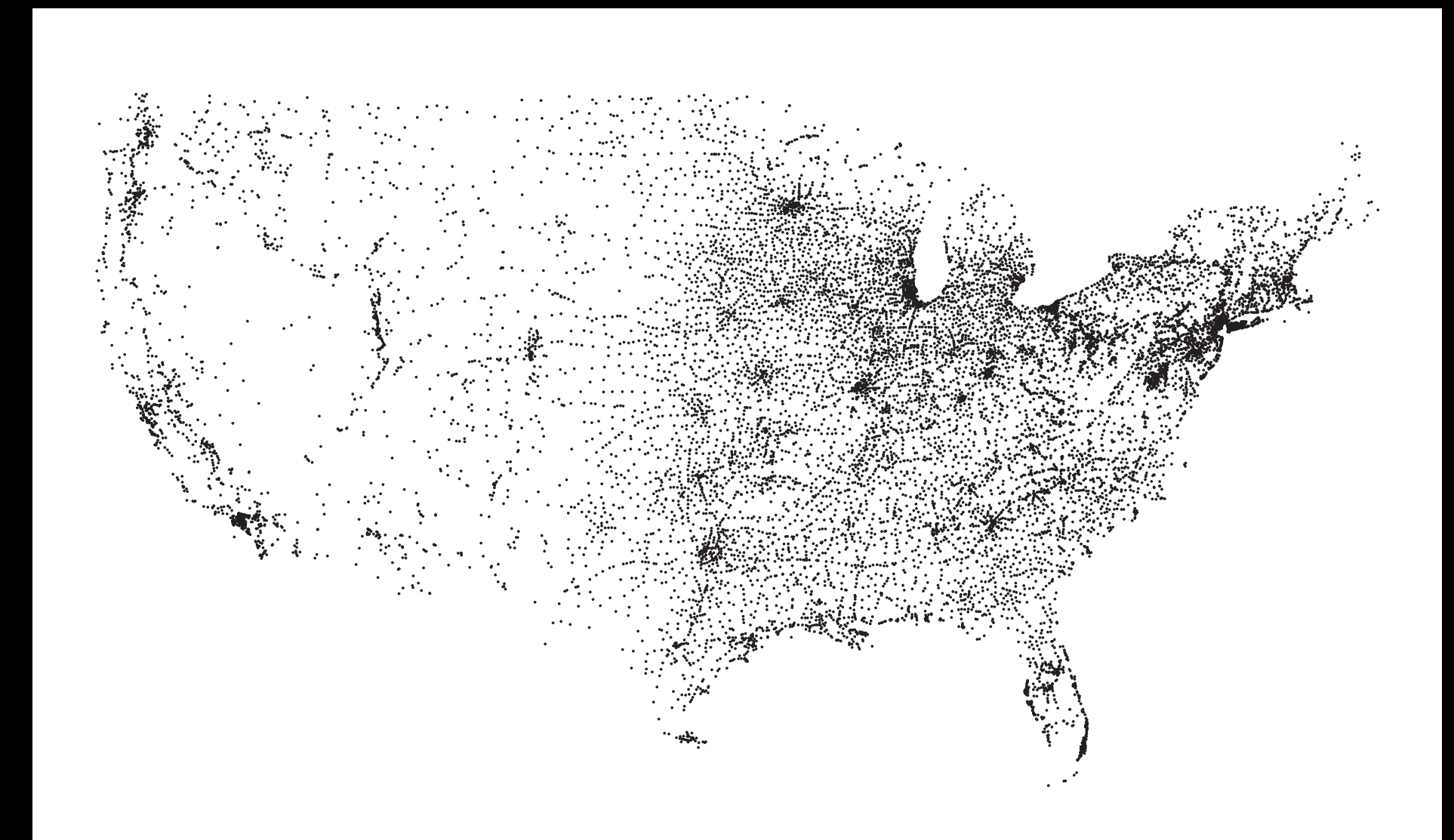
← *bounding box*

← *set x- and y-scale*

← *read points
and plot*

```
~/cos126/io> more USA.txt  
669905.0 247205.0 1244962.0 700000.0 ← bounding box  
1097038.8890 245552.7780  
1103961.1110 247133.3330  
1104677.7780 247205.5560  
...  
~/cos126/io> java-introcs PlotPoints < USA.txt
```

← *sequence of points
(13,509 USA cities)*



(699905, 247205)

Data visualization

Goal. Read points (from standard input) and plot.

```
public class PlotPoints {
    public static void main(String[] args) {

        double xmin = StdIn.readDouble();
        double ymin = StdIn.readDouble();
        double xmax = StdIn.readDouble();
        double ymax = StdIn.readDouble();

        StdDraw.setXscale(xmin, xmax);
        StdDraw.setYscale(ymin, ymax);

        while (!StdIn.isEmpty()) {
            double x = StdIn.readDouble();
            double y = StdIn.readDouble();
            StdDraw.point(x, y);
        }
    }
}
```

← *bounding box*

← *set x- and y-scale*

← *read points
and plot*

```
~/cos126/io> more StarryNight.txt
669905.0 247205.0 1244962.0 700000.0 ← bounding box
1097038.8890 245552.7780
1103961.1110 247133.3330 ← sequence of points
1104677.7780 247205.5560 (223,534 dots)
...

~/cos126/io> java-introcs PlotPoints < StarryNight.txt
```



Plotting a function

Goal. Plot $y = \sin(4x) + \sin(20x)$ in the interval $0 \leq x \leq \pi$.

Method. Take $n + 1$ samples, evenly spaced in interval.

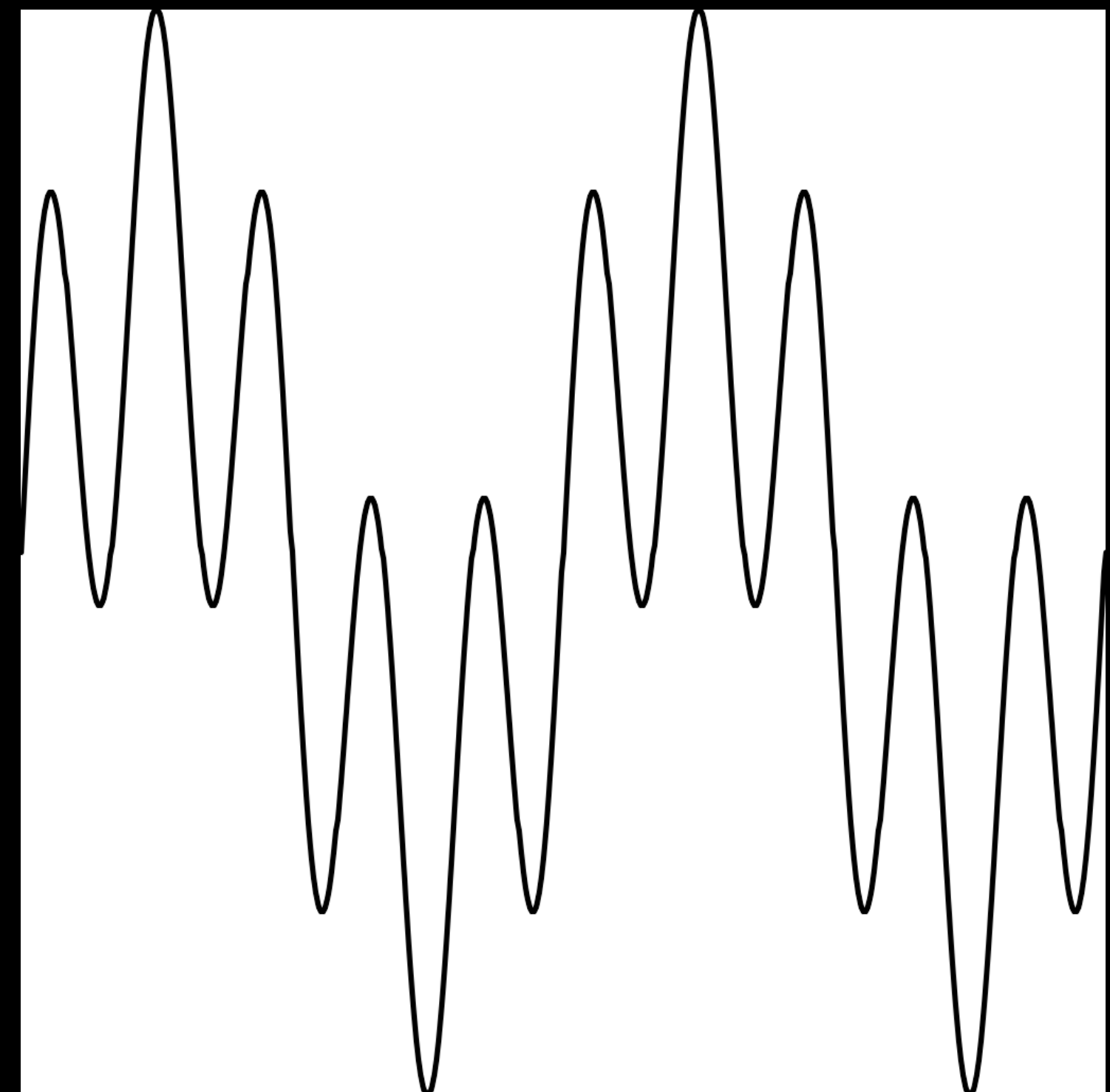
how many samples is enough?

```
public class PlotFunction {
    public static void main(String[] args) {
        int n = Integer.parseInt(args[0]);

        double[] x = new double[n+1];
        double[] y = new double[n+1];
        for (int i = 0; i <= n; i++) {
            x[i] = Math.PI * i / n;
            y[i] = Math.sin(4*x[i]) + Math.sin(20*x[i]);
        }

        StdDraw.setXscale(0, Math.PI);
        StdDraw.setYscale(-2.0, +2.0);
        for (int i = 0; i < n; i++)
            StdDraw.line(x[i], y[i], x[i+1], y[i+1]);
    }
}
```

```
~/cos126/io> java-introcs PlotFunction 1000
```

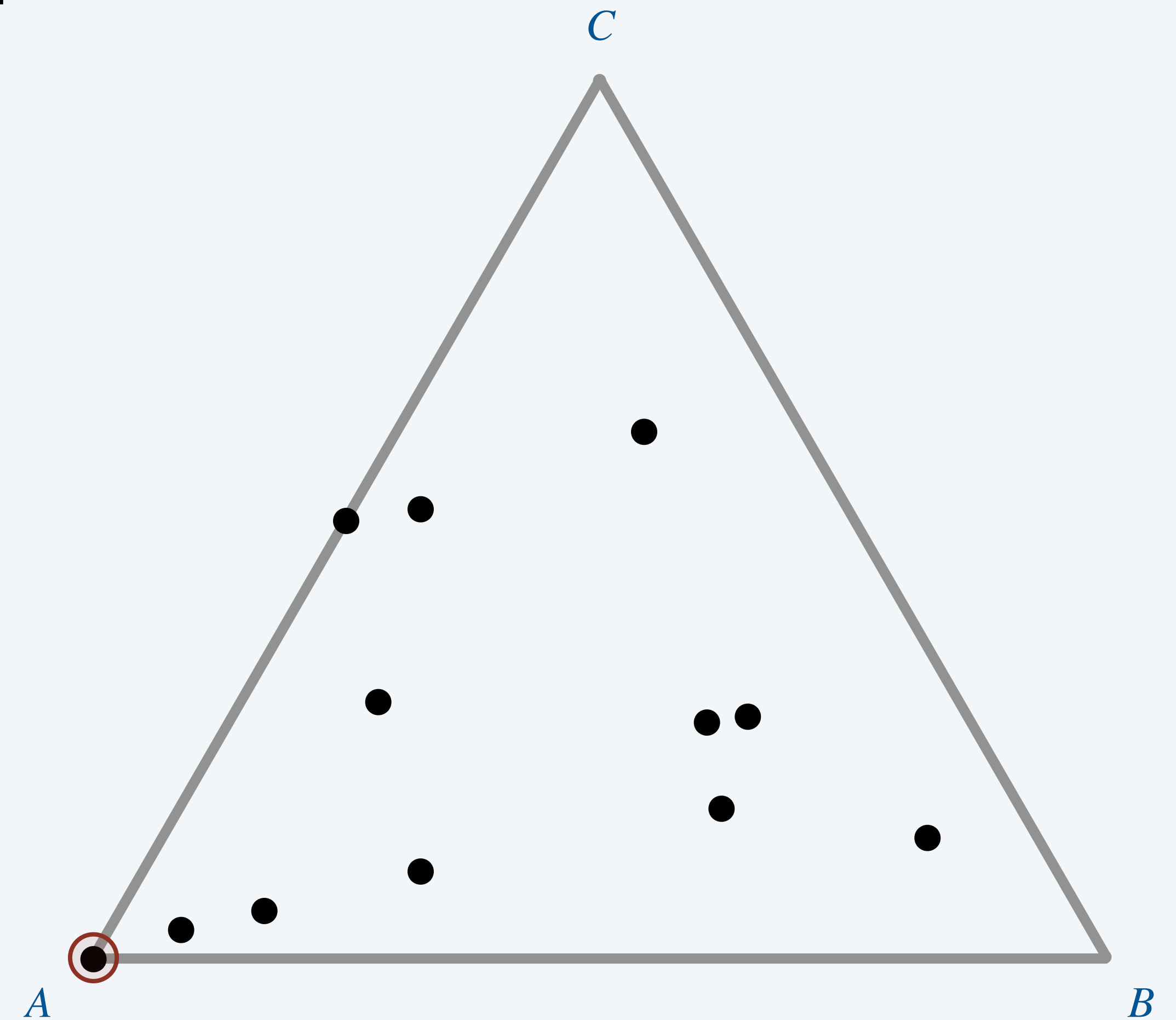


The chaos game

Chaos game. Draw an equilateral triangle; make one vertex the current point.

- Pick a vertex uniformly at random.
- Draw a point halfway between that vertex and the current point.
- Repeat.

<i>i</i>	vertex
0	<i>C</i>
1	<i>B</i>
2	<i>C</i>
3	<i>A</i>
4	<i>B</i>
5	<i>A</i>
6	<i>A</i>
7	<i>A</i>
8	<i>C</i>
9	<i>B</i>
10	<i>B</i>
...	...



Q. What figure emerges?

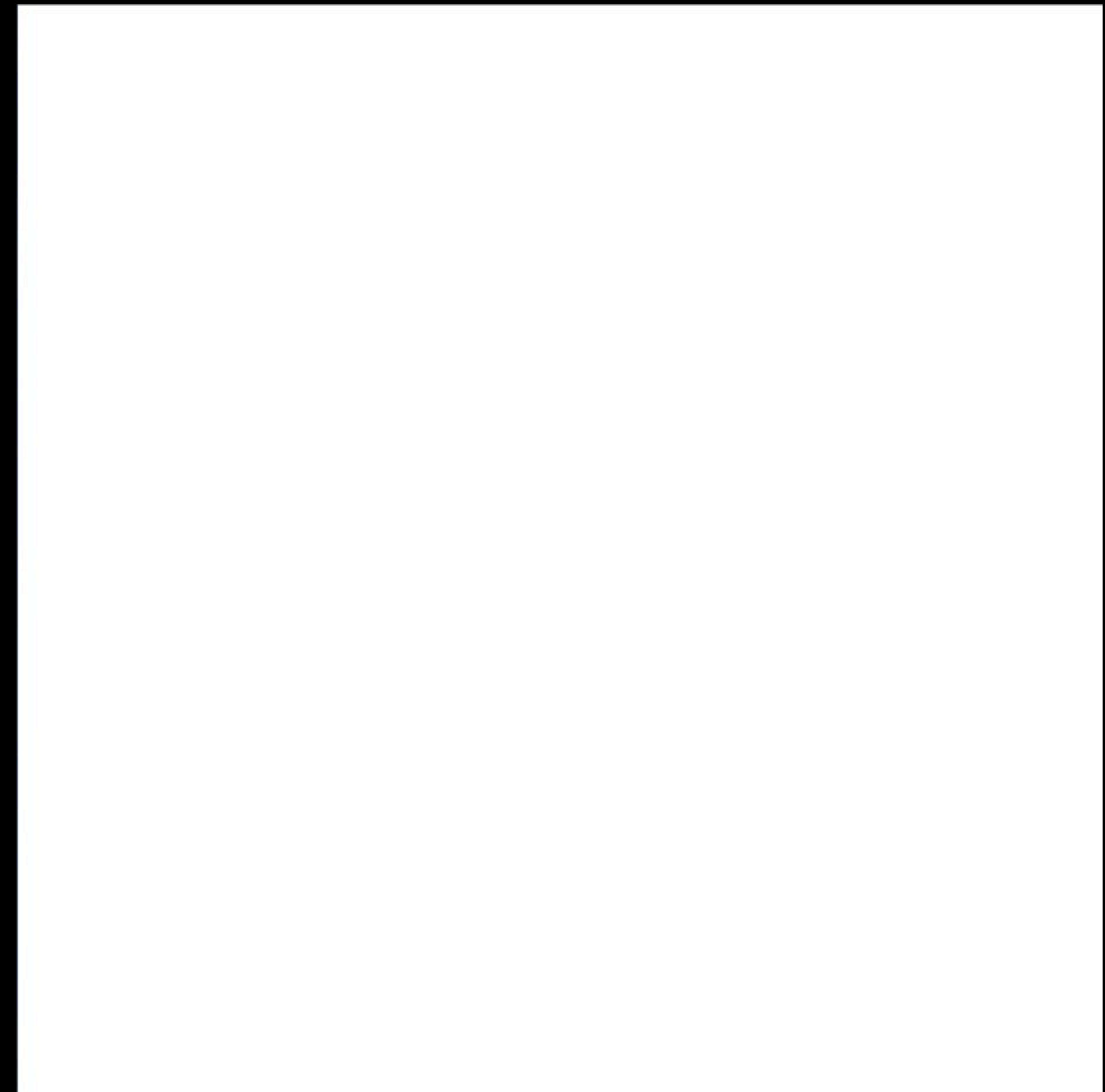
The chaos game: implementation

```
public class ChaosGame {
    public static void main(String[] args) {
        int trials = Integer.parseInt(args[0]);
        double c = Math.sqrt(3) / 2;
        double[] cx = { 0.0, 1.0, 0.5 };
        double[] cy = { 0.0, 0.0, c };
        StdDraw.setPenRadius(0.01);
        double x = 0.0, y = 0.0;
        for (int t = 1; t <= trials; t++) {
            int r = (int) (Math.random() * 3);
            x = (x + cx[r]) / 2.0;
            y = (y + cy[r]) / 2.0;
            StdDraw.point(x, y);
        }
    }
}
```

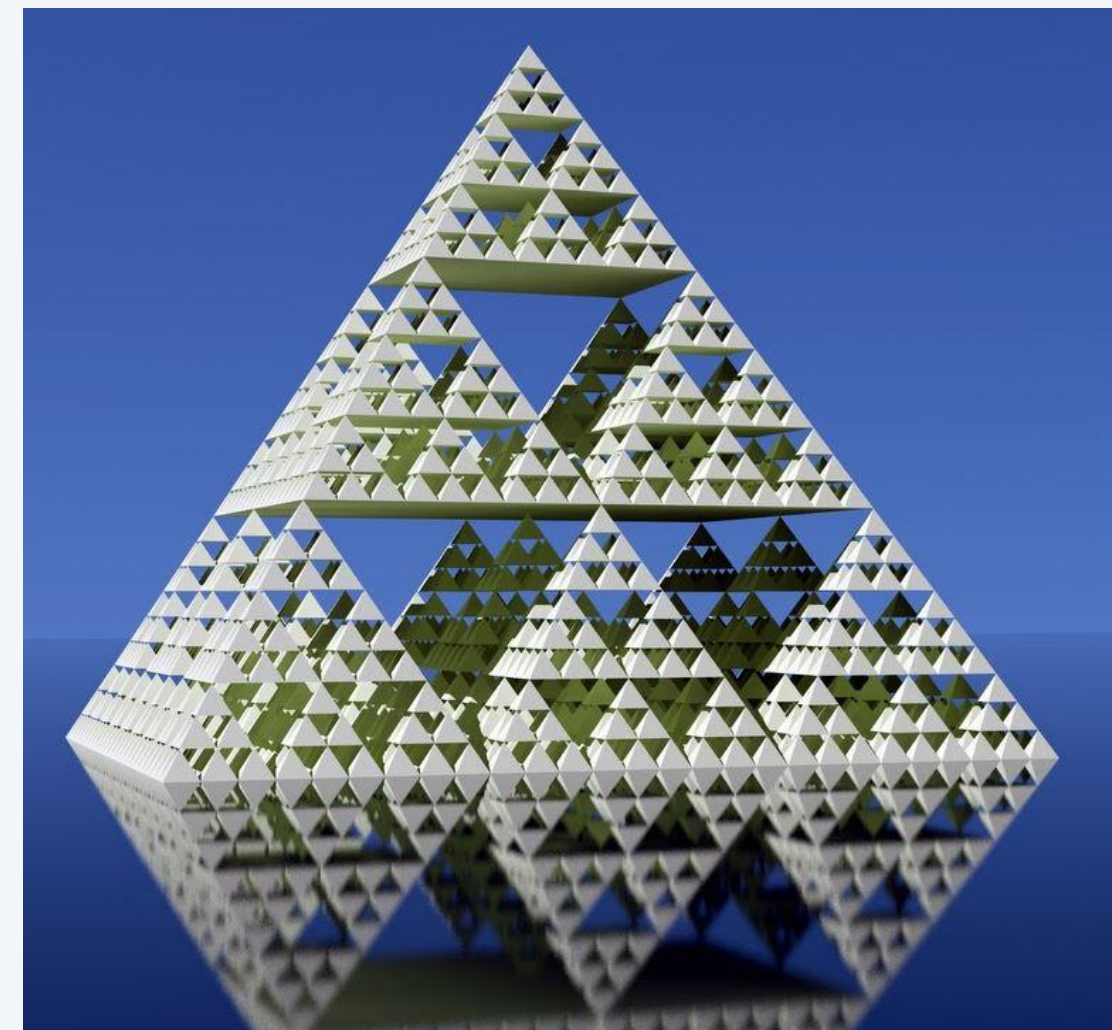
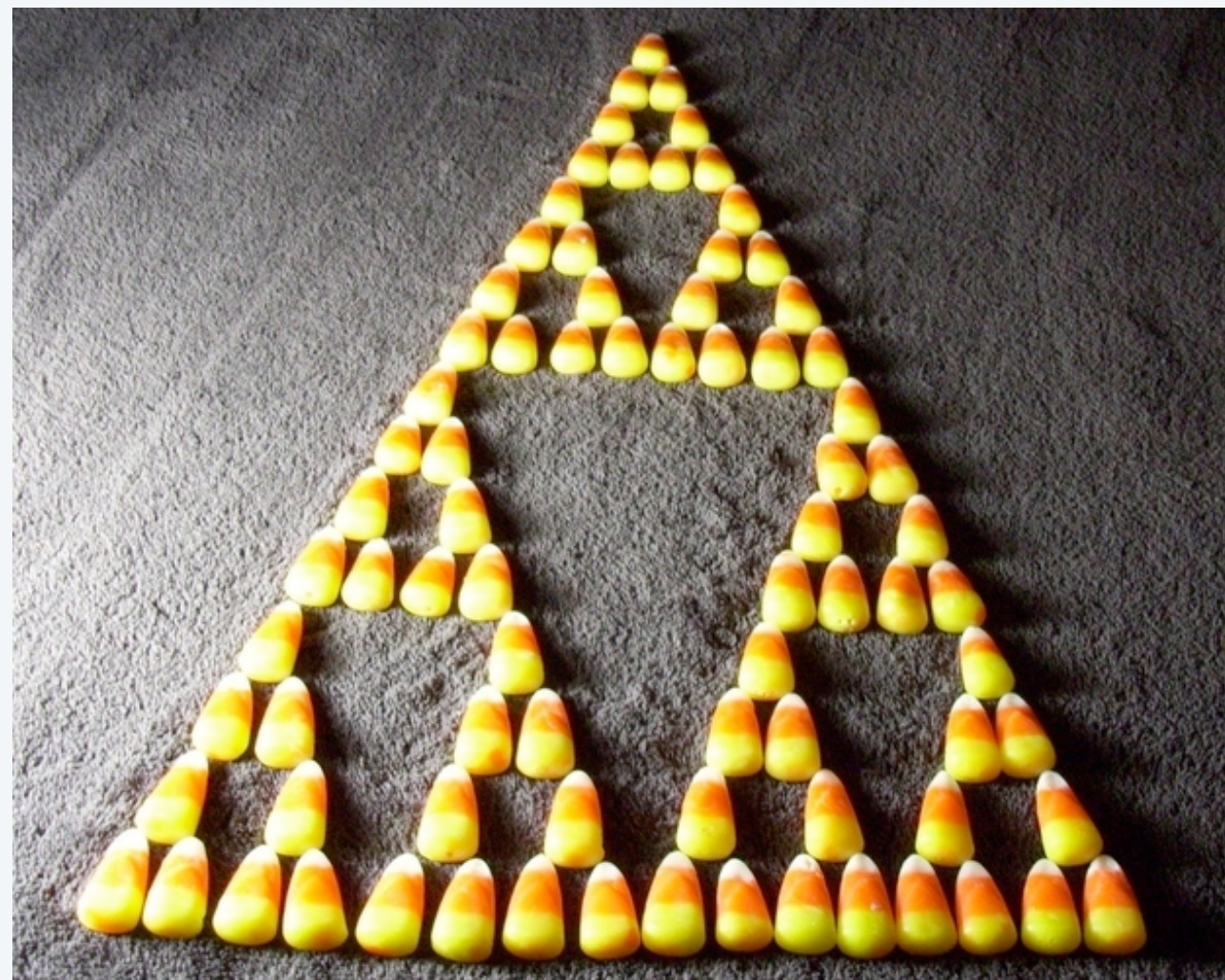
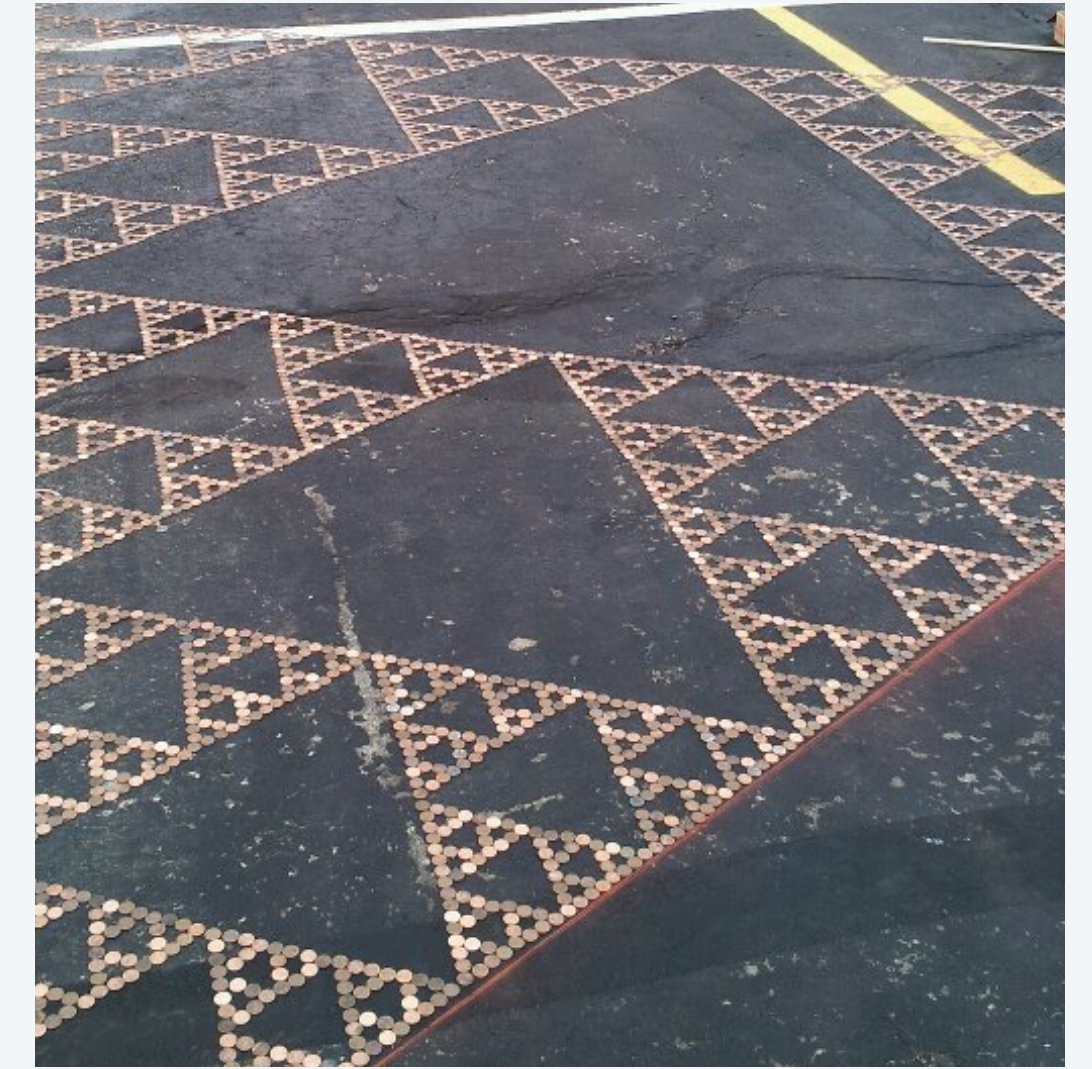
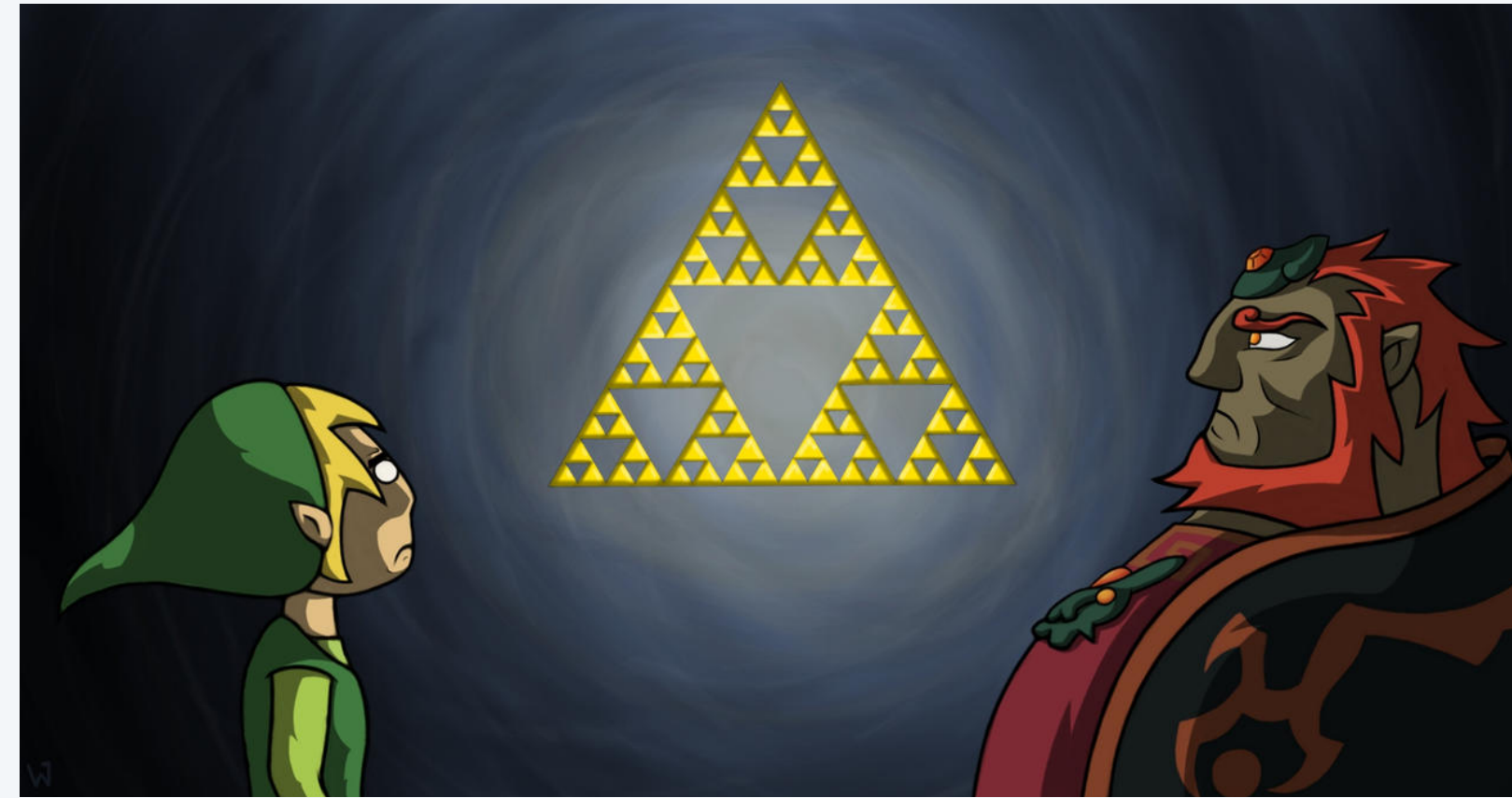
vertices of triangle

midpoint

```
~/cos126/io> java-introcs ChaosGame 10000
```



Sierpinski triangles in the wild

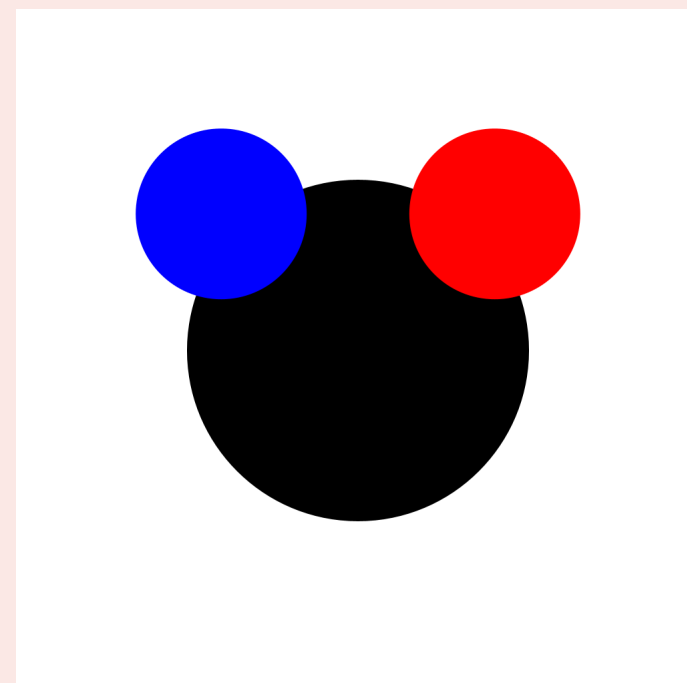




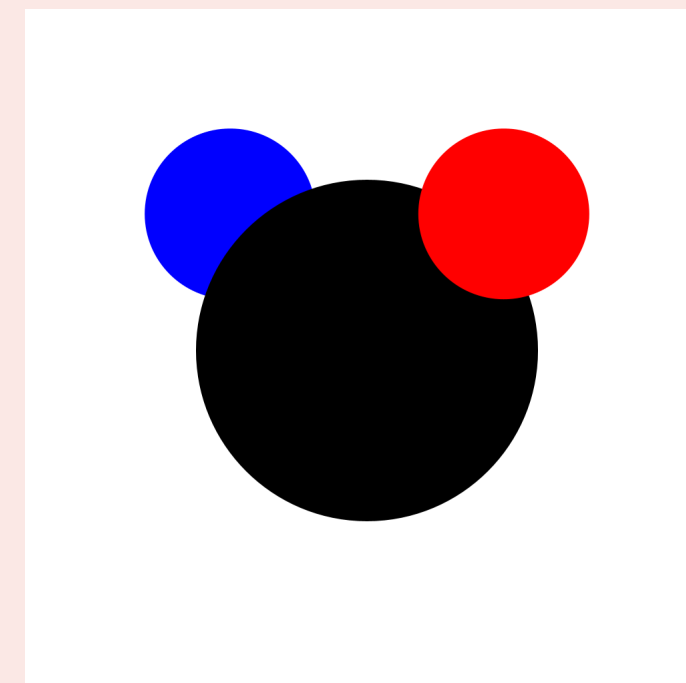
What is the result of executing the following code fragment?

```
// black circle (center)  
StdDraw.setPenColor(StdDraw.BLACK);  
StdDraw.filledCircle(0.5, 0.5, 0.25);  
  
// small blue circle (upper left)  
StdDraw.setPenColor(StdDraw.BLUE);  
StdDraw.filledCircle(0.3, 0.7, 0.125);  
  
// small red circle (upper right)  
StdDraw.setPenColor(StdDraw.RED);  
StdDraw.filledCircle(0.7, 0.7, 0.125);
```

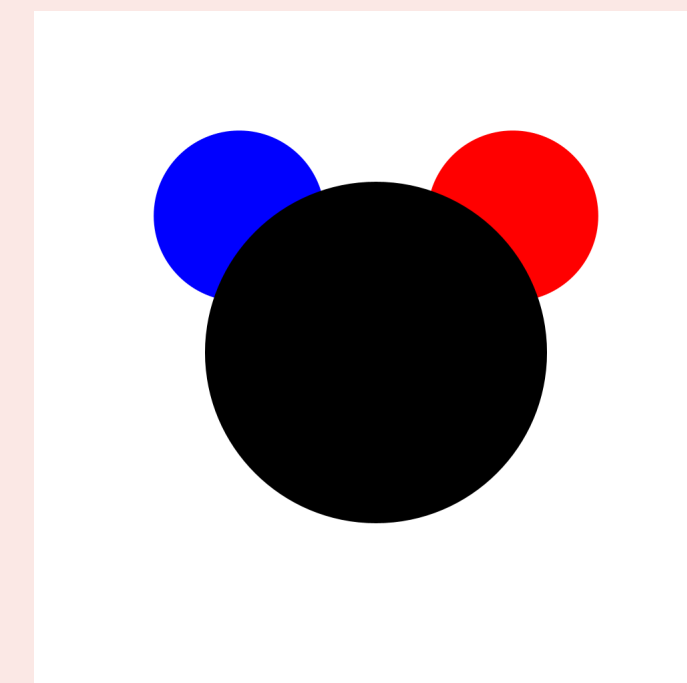
A.

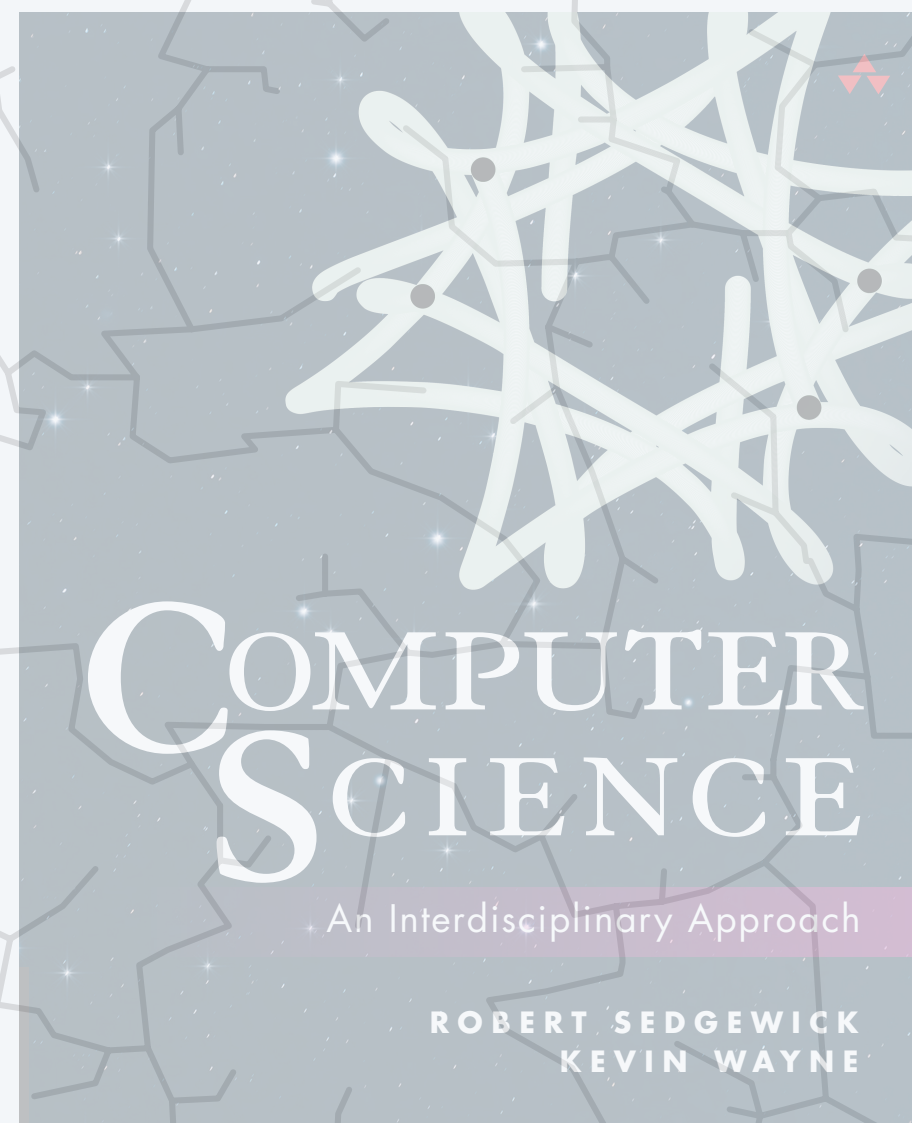


B.



C.





<https://introcs.cs.princeton.edu>

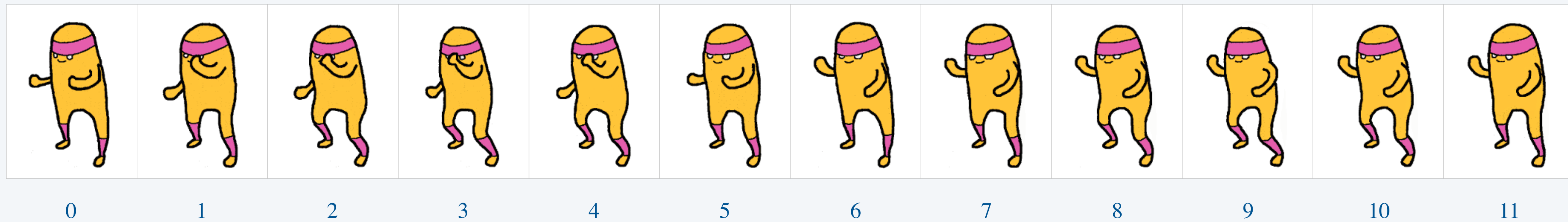
1.5 INPUT AND OUTPUT

- ▶ *standard input and output*
- ▶ *redirection and piping*
- ▶ *standard drawing*
- ▶ ***animation***

Computer animation

To create an animation, repeat the following:

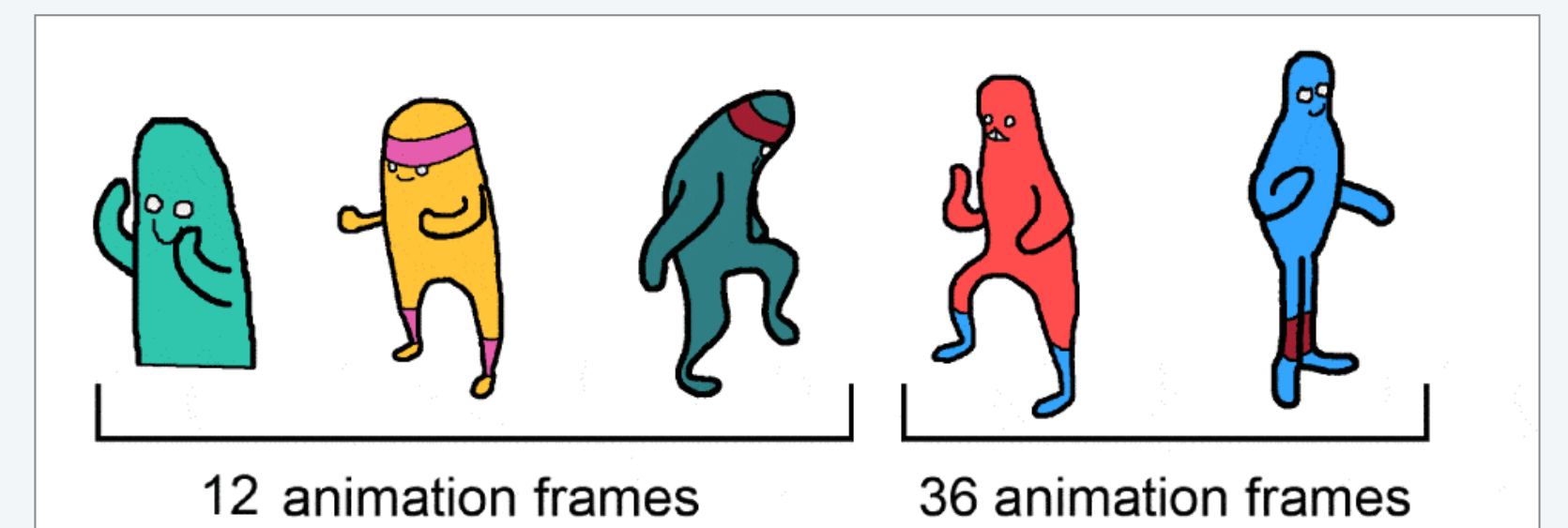
- Clear the drawing window.
- Draw next animation frame.
- Pause for a short period of time.



12 animation frames



Bottom line. Animation loop produces the illusion of motion.



Animation loop

Goal. Read animation frames from command line and display in an animation loop. ← “cel” animation

```
public class AnimationLoop {
    public static void main(String[] args) {
        int n = args.length;

        for (int i = 0; true; i++) {
            String filename = args[i % n];
            StdPicture.read(filename);
            StdPicture.show();
            StdPicture.pause(50);
        }
    }
}
```

animation
loop

cycles between
0 and $n-1$

50ms between frames
(20 frames per second)

```
~/cos126/io> ls tiger*.png
tiger00.png tiger03.png tiger06.png tiger09.png
tiger01.png tiger04.png tiger07.png tiger10.png
tiger02.png tiger05.png tiger08.png tiger11.png

~/cos126/io> java-introcs AnimationLoop tiger*.png
```



“wildcard”

StdDraw. Our library for drawing and **animating geometric shapes** in a graphical window.

<code>public class StdDraw</code>	description
<code>static void enableDoubleBuffering()</code>	<i>enable double buffering</i>
<code>static void disableDoubleBuffering()</code>	<i>disable double buffering</i>
<code>static void clear(Color color)</code>	<i>clear the background to color</i>
<code>static void show()</code>	<i>show the drawing in a window</i>
<code>static void pause(int t)</code>	<i>pause for t milliseconds</i>
<code> :</code>	<code>:</code>

Double buffering. Defer drawing shapes on screen until next call to `StdDraw.show()`.

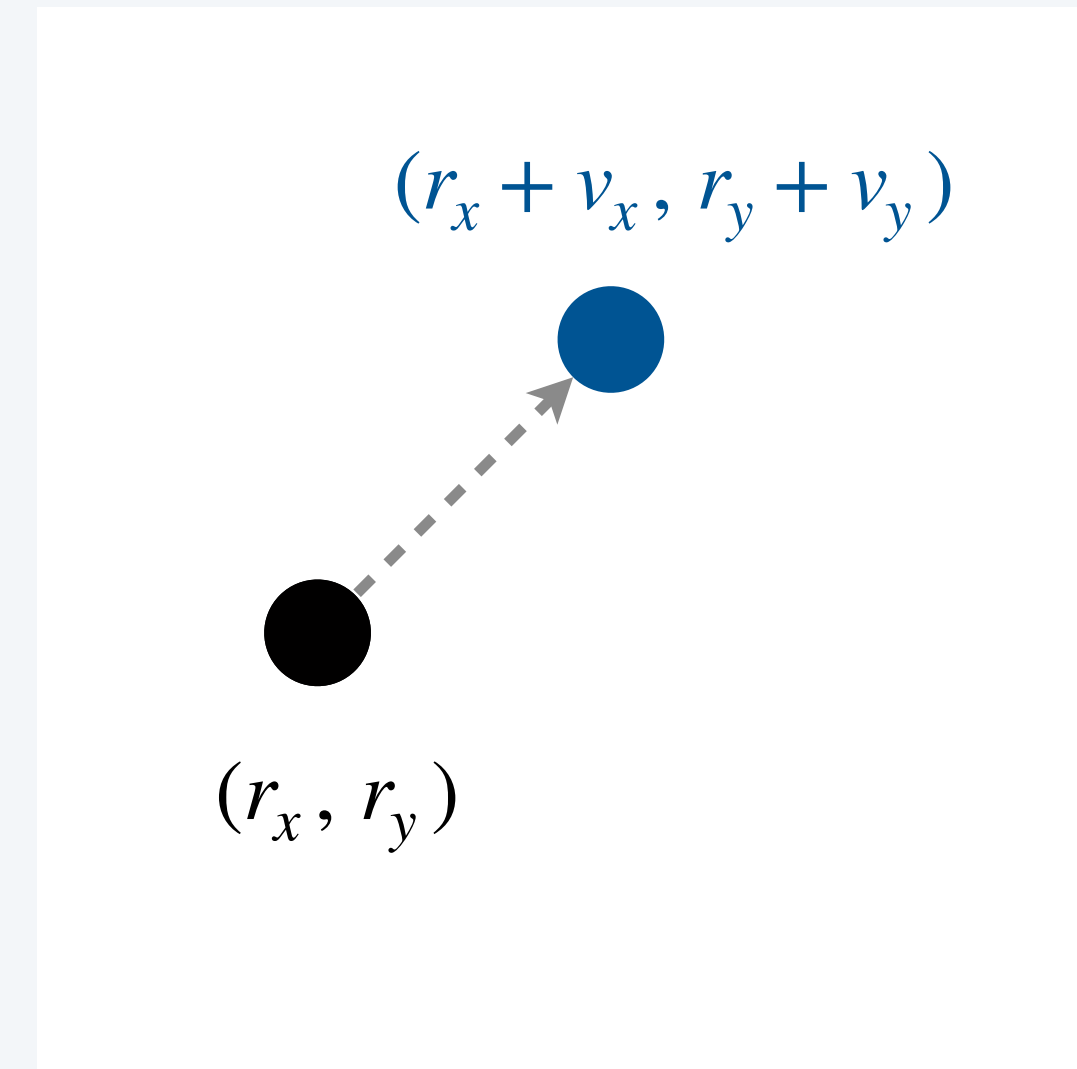
- Smoother animation.
- Faster (when drawing many shapes).

← *drawing to screen is slow;
typical screen refresh rate = 60 Hz*

Moving ball

Moving ball. [with constant velocity]

- Ball has position (r_x, r_y) and velocity (v_x, v_y) .
- To move ball, update position to $(r_x + v_x, r_y + v_y)$.



To animate a moving ball, repeat the following:

- Clear the drawing window.
- Move the ball.
- Draw the ball. | \longleftarrow *next animation frame*
- Pause for a short period of time.

Moving ball

```
public class MovingBall {
    public static void main(String[] args) {
        double rx = 0.0,    ry = 0.0;
        double vx = 0.001,  vy = 0.002;
        double radius = 0.10;

        StdDraw.setXscale(-1.0, +1.0);
        StdDraw.setYscale(-1.0, +1.0);
        StdDraw.enableDoubleBuffering();

        while (true) {
            rx = rx + vx;
            ry = ry + vy;
            StdDraw.clear(StdDraw.WHITE);
            StdDraw.filledCircle(rx, ry, radius);
            StdDraw.show();
            StdDraw.pause(20);
        }
    }
}
```

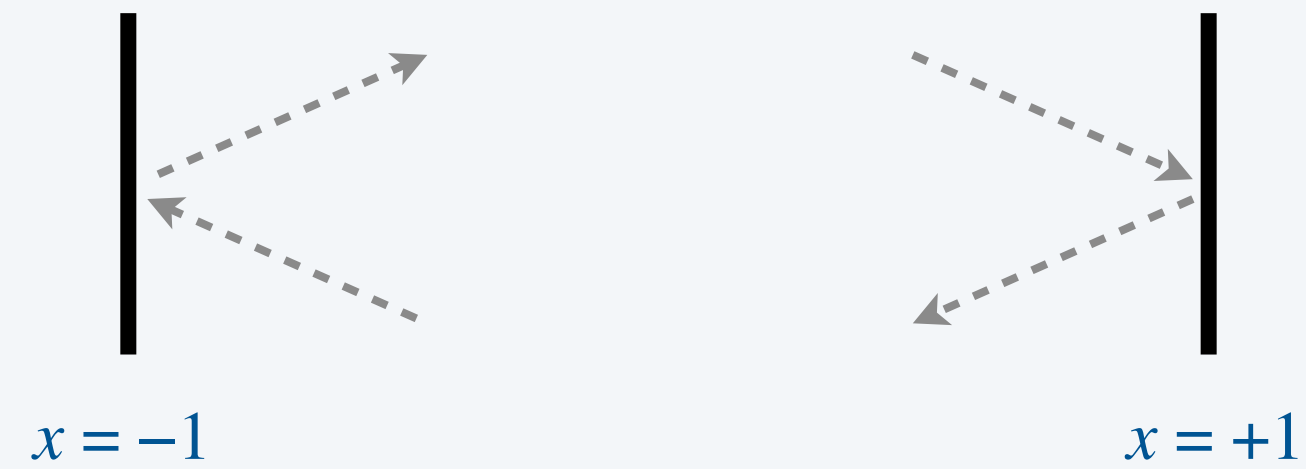
```
~/cos126/io> java-introcs MovingBall
```



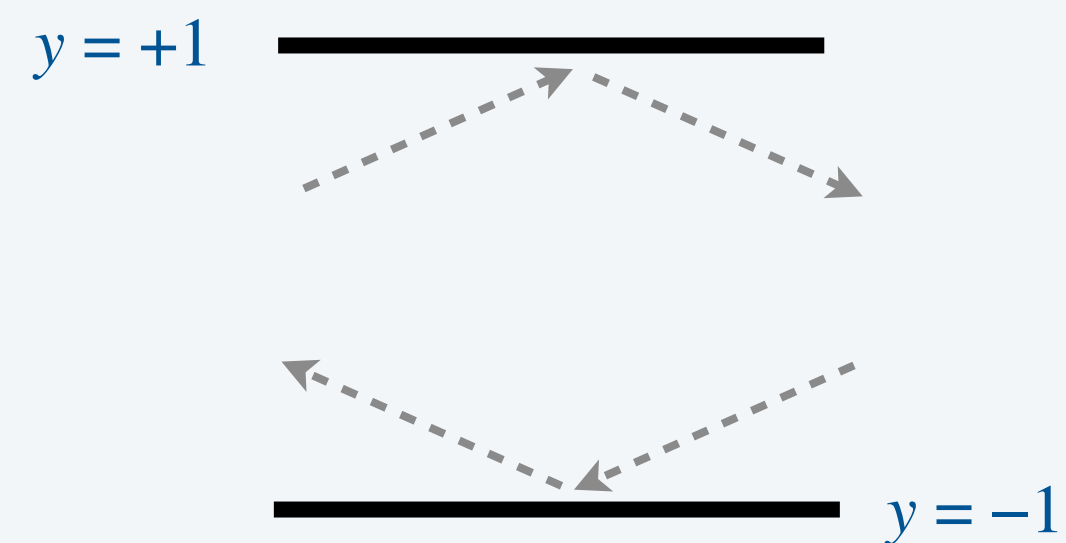
Bouncing ball

To “bounce” the ball off the walls:

- If the ball hits a vertical wall, set v_x to $-v_x$.



- If the ball hits a horizontal wall, set v_y to $-v_y$.



Physics. We're ignoring gravity, spin, friction, inelasticity, air resistance, ...

Bouncing ball

```
public class BouncingBall {
    public static void main(String[] args) {
        double rx = 0.480, ry = 0.860;
        double vx = 0.015, vy = 0.023;
        double radius = 0.1;

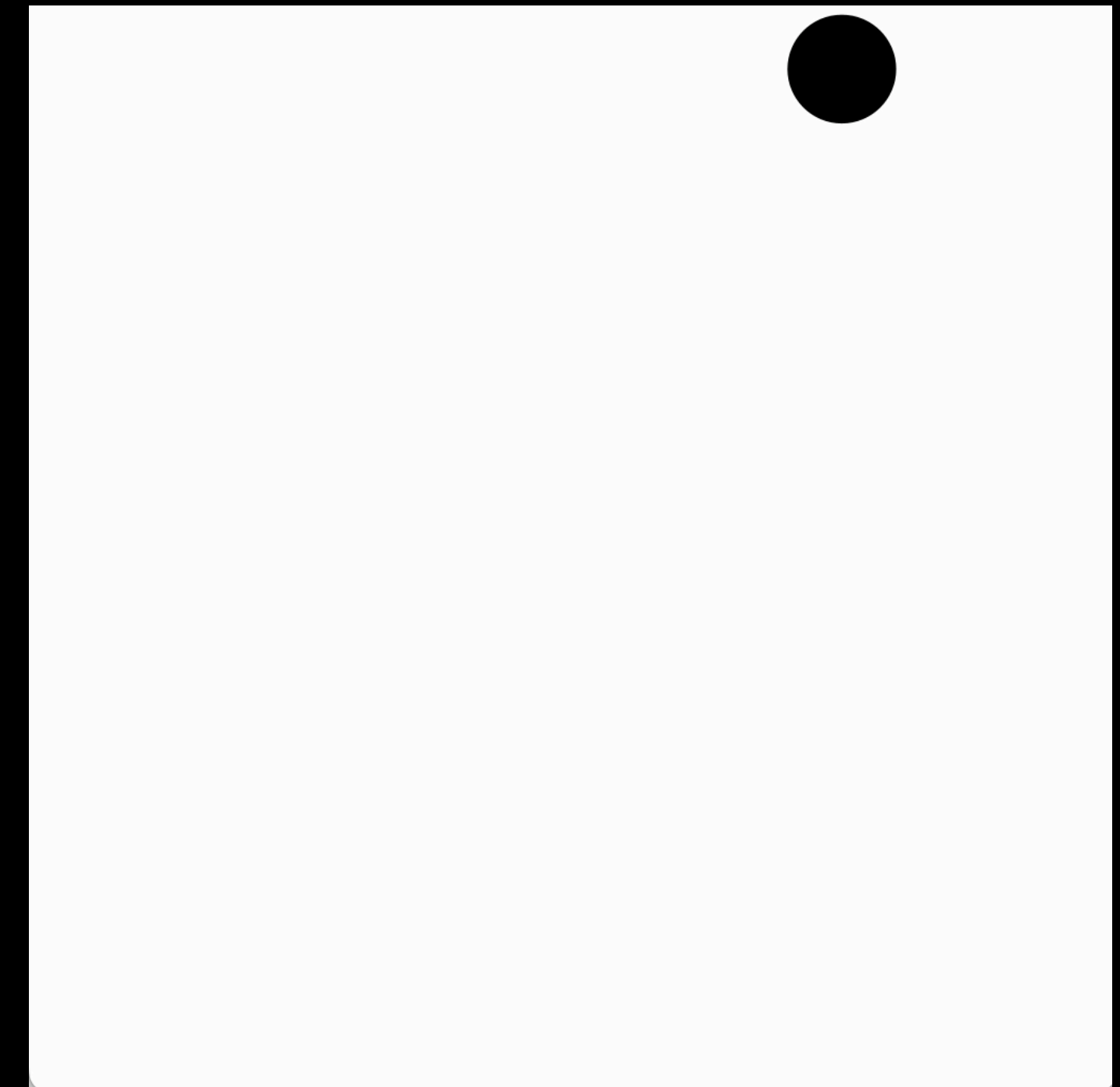
        StdDraw.setXscale(-1.0, +1.0);
        StdDraw.setYscale(-1.0, +1.0);
        StdDraw.enableDoubleBuffering();

        while (true) {
            rx = rx + vx;
            ry = ry + vy;
            if (Math.abs(rx) + radius >= 1.0) vx = -vx;
            if (Math.abs(ry) + radius >= 1.0) vy = -vy;
            StdDraw.clear(StdDraw.WHITE);
            StdDraw.filledCircle(rx, ry, radius);
            StdDraw.show();
            StdDraw.pause(20);
        }
    }
}
```

*bounce
off walls*

↓

```
~/cos126/io> java-introcs BouncingBall
```



StdAudio. Our library for processing digital audio. ←

*available with javac-introcs
and java-introcs commands*

<code>public class StdAudio</code>	description
<code>static int SAMPLE_RATE</code>	44100 (<i>CD quality audio</i>)
<code>static void play(double sample)</code>	<i>play the sample</i>
<code>static void play(double[] sample)</code>	<i>play the samples</i>
<code>static void play(String filename)</code>	<i>play the audio file (do not execute subsequent code until done playing)</i>
<code>static void playInBackground(String filename)</code>	<i>play the audio file in a background thread (execute subsequent code while playing)</i>
<code>static double[] read(String filename)</code>	<i>read the samples from an audio file</i>
<code>⋮</code>	<code>⋮</code>

Deluxe bouncing ball



```
while (true) {
    rx = rx + vx;
    ry = ry + vy;


    // bounce off vertical walls
    if (Math.abs(rx) + radius > 1.0) {
        vx = -vx;
        StdAudio.playInBackground("BallTap.wav");
    }

    // bounce off horizontal walls
    if (Math.abs(ry) + radius > 1.0) {
        vy = -vy;
        StdAudio.playInBackground("BlockHit.wav");
    }

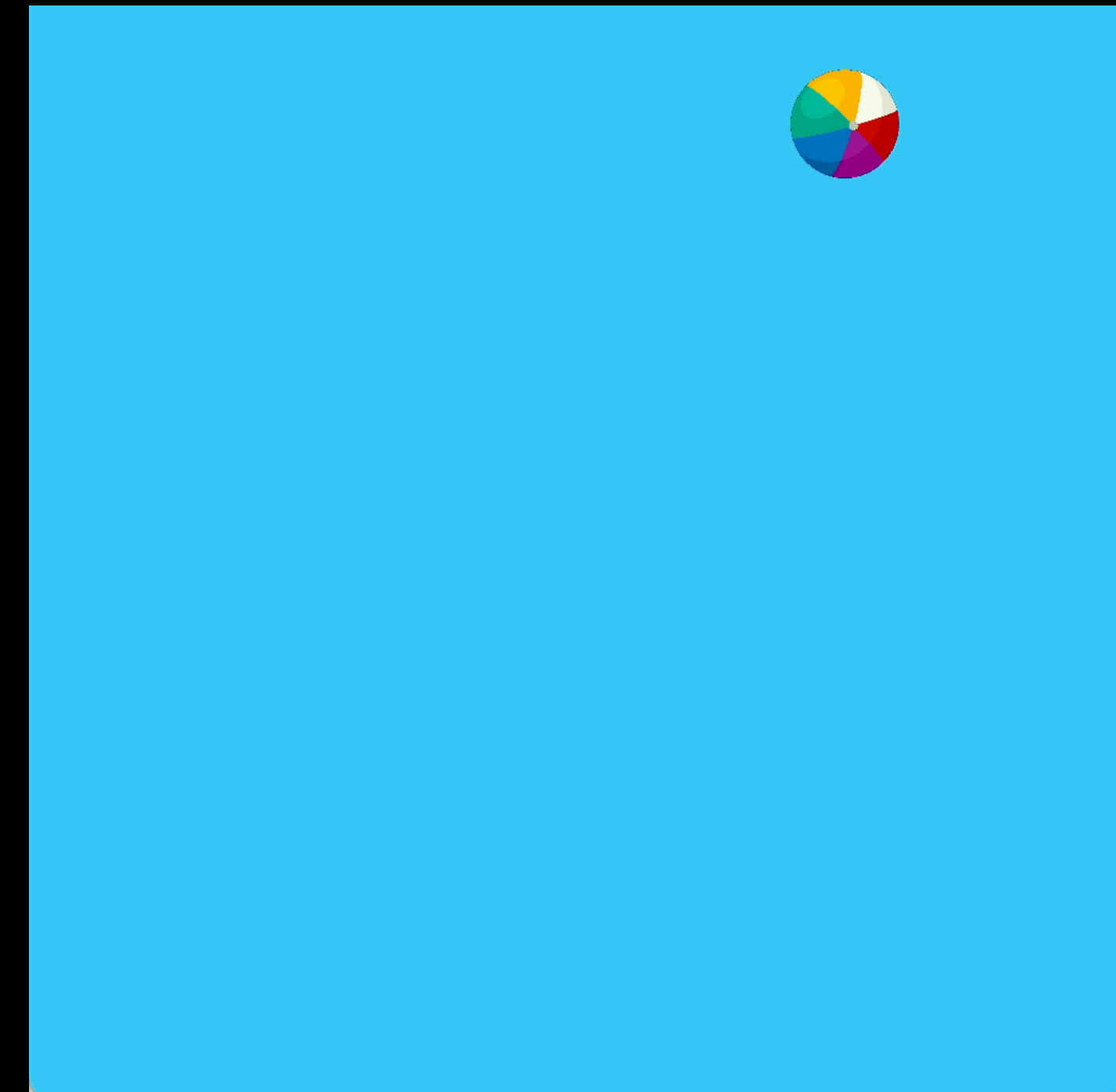
    StdDraw.clear(StdDraw.BOOK_LIGHT_BLUE);
    StdDraw.picture(rx, ry, "ball.png", 2*radius, 2*radius);
    StdDraw.show();
    StdDraw.pause(20);
}
```

plays sound effect

*draws picture
(resized to specified width and height)*



```
~/cos126/io> java-introcs DeluxeBouncingBall
```

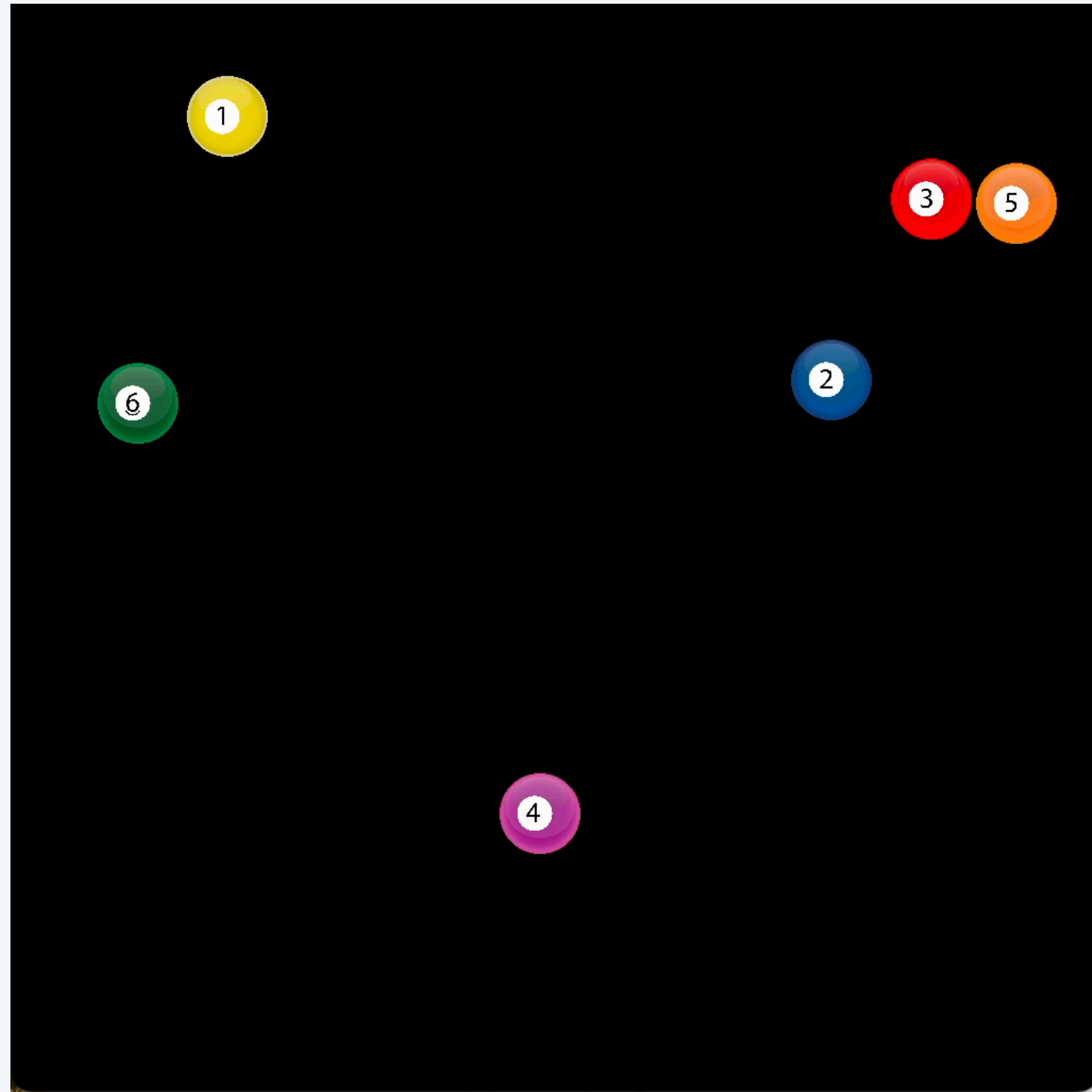




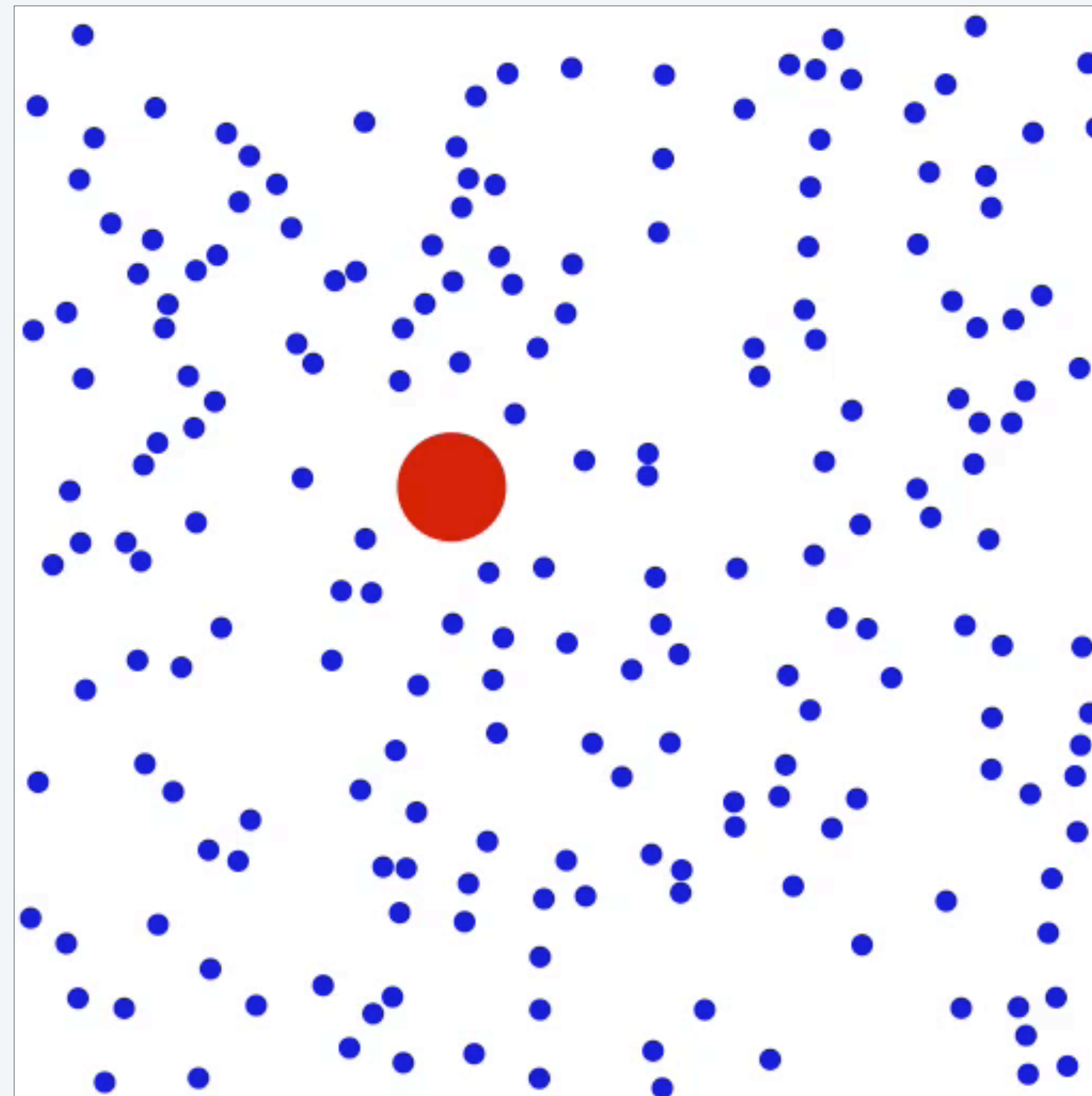
What happens if we clear the screen outside the animation loop (instead of inside it)?

- A. Works exactly as before.
- B. Light blue only.
- C. Display a trace of the ball's entire path.
- D. Compile-time error.

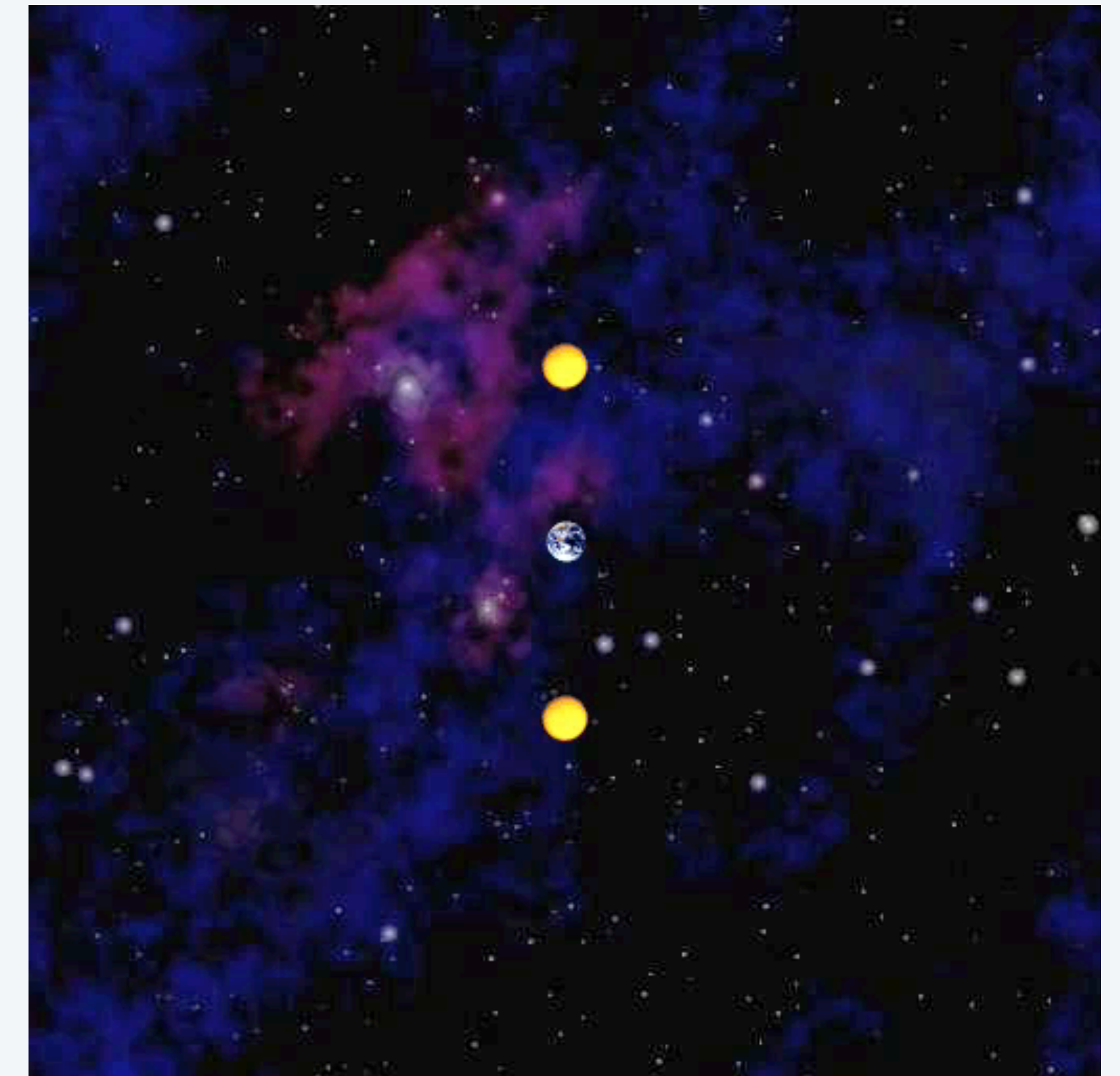
```
StdDraw.clear(StdDraw.BOOK_LIGHT_BLUE);  
  
while (true) {  
    rx = rx + vx;  
    ry = ry + vy;  
    if (Math.abs(rx) + radius > 1.0) vx = -vx;  
    if (Math.abs(ry) + radius > 1.0) vy = -vy;  
    StdDraw.clear(StdDraw.BOOK_LIGHT_BLUE);  
    StdDraw.picture(rx, ry, "ball.png", 2*radius, 2*radius);  
    StdDraw.show();  
    StdDraw.pause(20);  
}
```

multiple balls



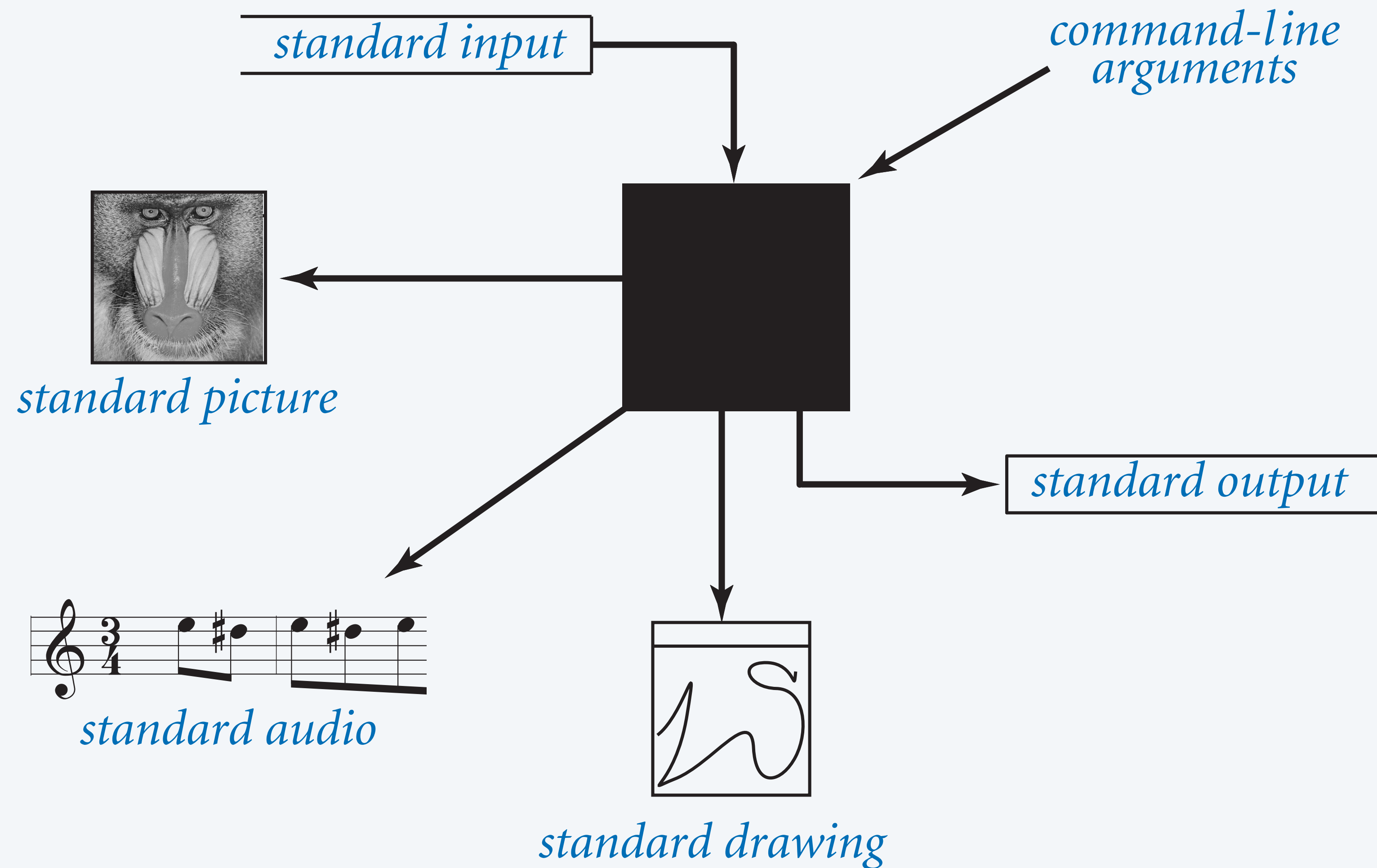
elastic collisions



gravity

Input-output abstractions

Summary. Input and output for text, pictures, drawings, and audio.



Credits

media	source	license
<i>Computer Monitor</i>	<u>iStock</u>	<u>standard license</u>
<i>DEC VT100 Terminal</i>	<u>Wikimedia</u>	<u>CC BY-SA 4.0</u>
<i>Mandrill</i>	<u>USC SIPI Image Database</u>	
<i>Starry Night Stipple</i>	Julia Ying '26	by author
<i>Sierpinski Coca Cola</i>	<u>Paul Bourke</u>	
<i>The Legend of Sierpinski</i>	<u>Sheilakh</u>	
<i>Sierpinski Pennies</i>	<u>Pinterest</u>	
<i>Sierpinski Candy Corn</i>	<u>Pinterest</u>	
<i>Sierpinski Pyramid</i>	<u>Wikimedia</u>	
<i>Sierpinski Cookie</i>	unknown	

Credits

media	source	license
<i>Dancing Characters</i>	<u>Mark Knight</u>	
<i>Tiger Animation Frames</i>	<u>Adobe Stock</u>	<u>education license</u>
<i>Beach Ball</i>	<u>Open Clip Art</u>	<u>public domain</u>
<i>Sound Effects</i>	<u>Mixkit</u>	<u>Mixkit free license</u>
<i>Pool Balls</i>	<u>Openclipart</u>	<u>public domain</u>