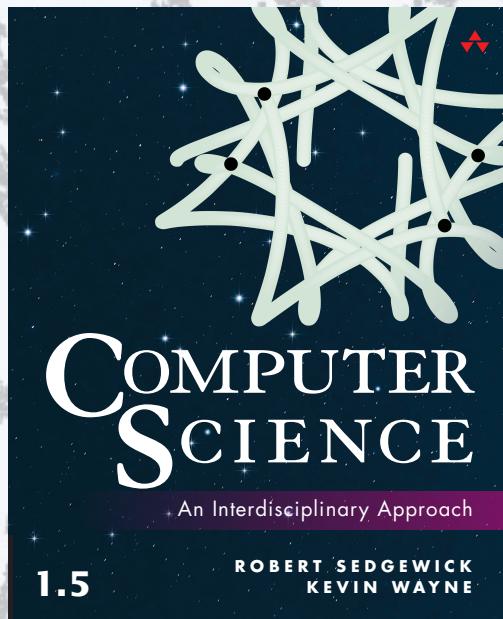


COMPUTER SCIENCE
SEGEWICK / WAYNE
PART I: PROGRAMMING IN JAVA



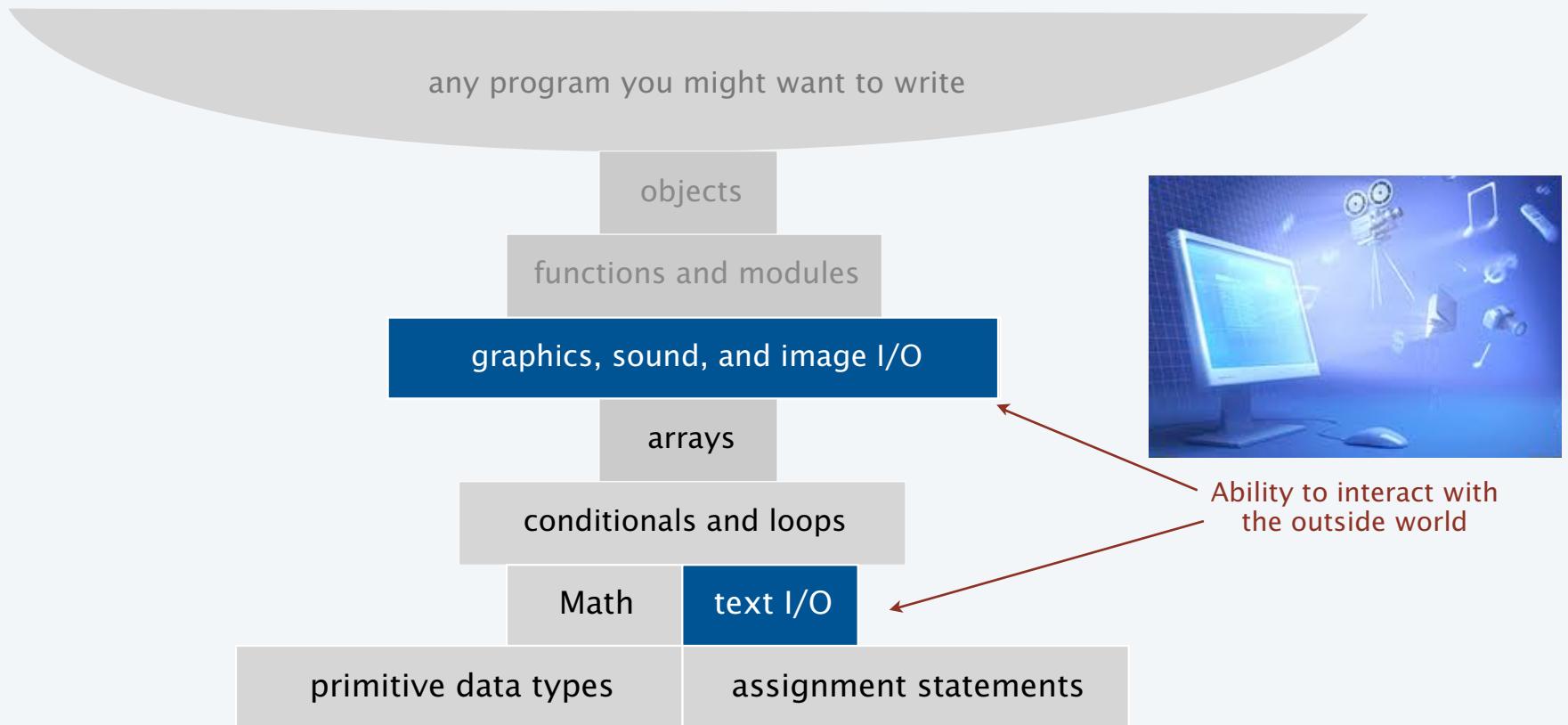
<http://introcs.cs.princeton.edu>

4. Input and Output

4. Input and Output

- Standard input and output
- Standard drawing
- Fractal drawings
- Animation

Basic building blocks for programming



Input and output

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

Typical
INPUT
devices



Keyboard



Trackpad



Storage



Network



Camera



Microphone

Typical
OUTPUT
devices



Display



Storage



Network



Printer



Speakers

Our approach.

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

Abstraction

plays an *essential* role in understanding computation.

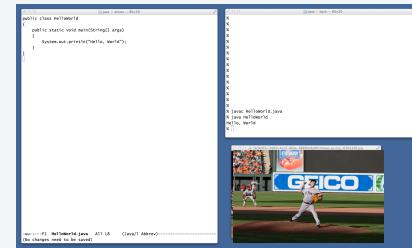
Interested in thinking more deeply about this concept?

An *abstraction* is something that exists only as an idea.

Consider taking a philosophy course.

Example: "Printing" is the idea of a program producing text as output.

Good abstractions *simplify* our view of the world, by *unifying* diverse real-world artifacts.



This lecture. Abstractions for delivering input to or receiving output from our programs.

Quick review

Terminal. An abstraction for providing input and output to a program.

The image shows a Mac OS X desktop environment. On the left, a Java code editor displays the following code:

```
public class DrawCards
{
    public static void main(String[] args)
    {
        int N = Integer.parseInt(args[0]);
        String[] rank = {"2", "3", "4", "5", "6", "7", "8", "9",
                         "10", "J", "Q", "K", "A"};
        String[] suit = { "♣", "♦", "♥", "♠" };
        String[] deck = new String[52];
        for (int i = 0; i < 13; i++)
            for (int j = 0; j < 4; j++)
                deck[i + 13*j] = rank[i] + suit[j];
        for (int i = 0; i < N; i++)
            System.out.print(deck[(int) (Math.random() * 52)]);
        System.out.println();
    }
}
```

The terminal window on the right shows the execution of the program:

```
% java DrawCards 10
7♠ 2♥ Q♦ A♣ Q♣ 2♦ Q♥ 6♦ 5♥ 10♦
```

Annotations explain the process:

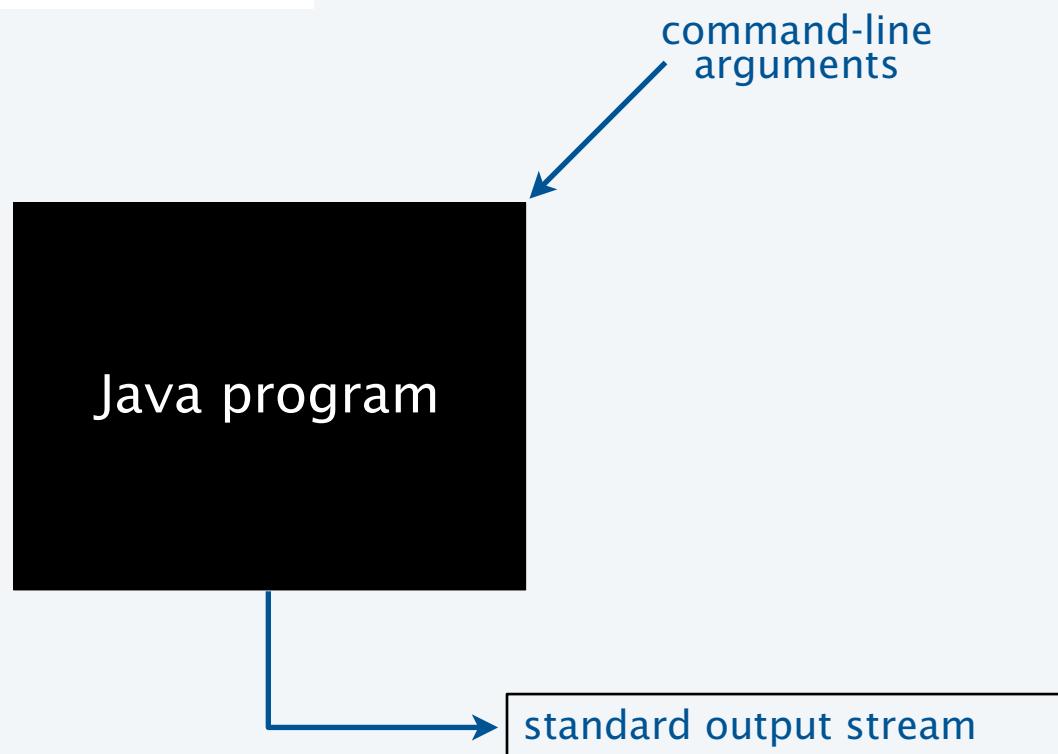
- "Input from command line" points to the command `% java DrawCards 10`.
- "Output to standard output stream" points to the card sequence `7♠ 2♥ Q♦ A♣ Q♣ 2♦ Q♥ 6♦ 5♥ 10♦`.
- A small image of a vintage computer monitor labeled "Virtual VT-100 terminal" is connected by an arrow to the terminal window, indicating where the output is displayed.

The bottom terminal window shows the command being run again:

```
% java DrawCards 5
K♣ J♦ A♣ 10♦ 5♦
%
% java DrawCards 10
7♠ 2♥ Q♦ A♣ Q♣ 2♦ Q♥ 6♦ 5♥ 10♦
%
% java DrawCards 20
8♠ 9♦ A♥ 2♣ K♦ 7♦ J♥ A♥ 6♦ 5♣ 8♥ Q♦ Q♣ J♣ 8♣ 7♣ 3♦ J♥ 10♥ 4♥
%
%
```

Input-output abstraction (so far)

A mental model of what a Java program does.



Review: command-line input

Command-line input. An abstraction for providing arguments (strings) to a program.

Basic properties

- Strings you type after the program name are available as `args[0]`, `args[1]`, ... at *run* time.
- Arguments are available when the program *begins* execution.
- Need to call system conversion methods to convert the strings to other types of data.

```
public class RandomInt
{
    public static void main(String[] args)
    {
        int N = Integer.parseInt(args[0]);
        double r = Math.random();
        int t = (int) (r * N);
        System.out.println(t);
    }
}
```

```
% java RandomInt 6
3

% java RandomInt 10000
3184
```

Review: standard output

Infinity. An abstraction describing something having no limit.

Standard output stream. An abstraction for an infinite output sequence.

Basic properties

- Strings from `System.out.println()` are added to the end of the standard output stream.
- Standard output stream is sent to terminal application by default.

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

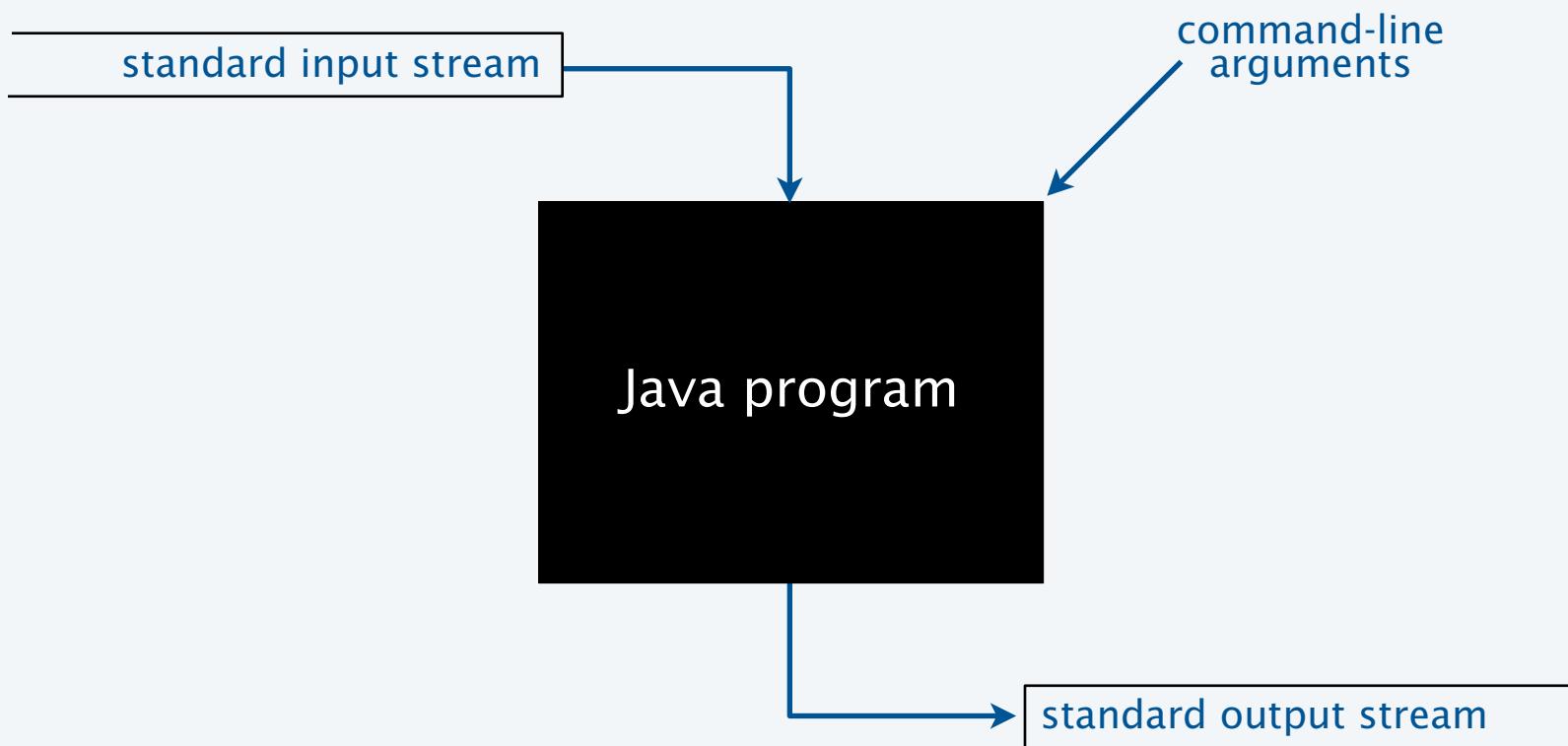
```
% java RandomSeq 4
0.9320744627218469
0.4279508713950715
0.08994615071160994
0.6579792663546435
```

No limit on amount
of output → ...

```
% java RandomSeq 1000000
0.09474882292442943
0.2832974030384712
0.1833964252856476
0.2952177517730442
0.8035985765979008
0.7469424300071382
0.5835267075283997
0.3455279612587455
...
```

Improved input-output abstraction

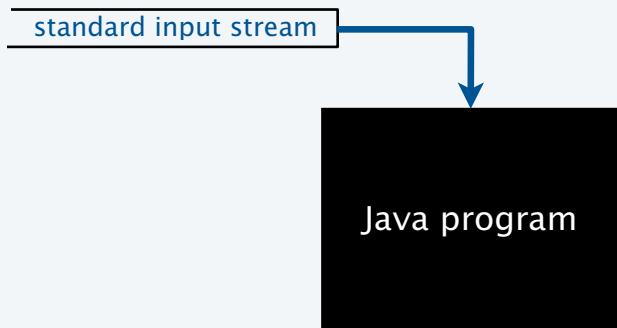
Add an infinite *input* stream.



Standard input

Infinity. An abstraction describing something having no limit.

Standard input stream. An abstraction for an infinite *input* sequence.



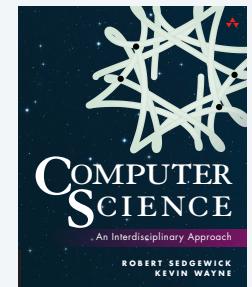
Advantages over command-line input

- Can provide new arguments *while* the program is executing.
- No limit on the amount of data we can input to a program.
- Conversion to primitive types is explicitly handled (stay tuned).

StdIn library

Developed for this course, but broadly useful

- Implement abstractions invented for UNIX in the 1970s.
- Available for download at booksite.
- Included in introscs software you downloaded at the beginning of the course.



public class StdIn	
boolean isEmpty()	<i>true iff no more values</i>
int readInt()	<i>read a value of type int</i>
double readDouble()	<i>read a value of type double</i>
long readLong()	<i>read a value of type long</i>
boolean readBoolean()	<i>read a value of type boolean</i>
char readChar()	<i>read a value of type char</i>
String readString()	<i>read a value of type String</i>
String readAll()	<i>read the rest of the text</i>

standard input stream

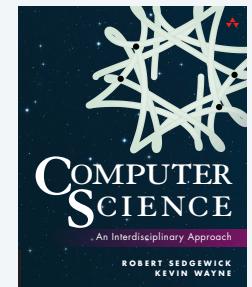


Java program

StdOut library

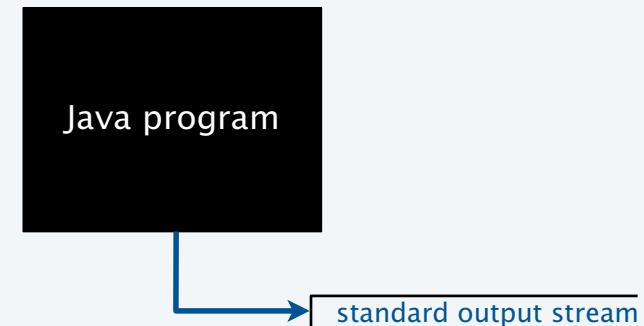
Developed for this course, but broadly useful

- Implement abstractions invented for UNIX in the 1970s.
- Available for download at booksite.
- Included in introscs software you downloaded at the beginning of the course.



```
public class StdOut
```

void print(String s)	<i>put s on the output stream</i>
void println()	<i>put a newline on the output stream</i>
void println(String s)	<i>put s, then a newline on the stream</i>
void printf(String f, ...)	<i>formatted output</i>



Q. These are the same as `System.out`. Why not just use `System.out`?

- A. We provide a consistent set of simple I/O abstractions in one place.
- A. We can make output *independent* of system, language, and locale.

use StdOut
from now on

StdIn/StdOut warmup

Interactive input

- Prompt user to type inputs on standard input stream.
- Mix input stream with output stream.

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

```
% java AddTwo
Type the first integer: 1
Type the second integer: 2
Their sum is 3
```

StdIn application: average the numbers on the standard input stream

Average

- Read a stream of numbers.
- Compute their average.

Q. How do I specify the end of the stream?

- A. <Ctrl-d> (standard for decades).
- A. <Ctrl-z> (Windows).

```
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.println(sum / n);
    }
}
```

Key points

- No limit on the size of the input stream.
- Input and output can be interleaved.

```
% java Average
10.0 5.0 6.0
3.0 7.0 32.0
<Ctrl-d>
10.5
```

Summary: prototypical applications of standard output and standard input

StdOut: Generate a stream of random numbers

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

StdIn: Compute the average of a stream of numbers

```
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.println(sum / n);
    }
}
```

Both streams are *infinite* (no limit on their size).

Q. Do I always have to type in my input data and print my output?

A. No! Keep data and results in *files* on your computer, or use *piping* to connect programs.

Redirection: keep data in files on your computer

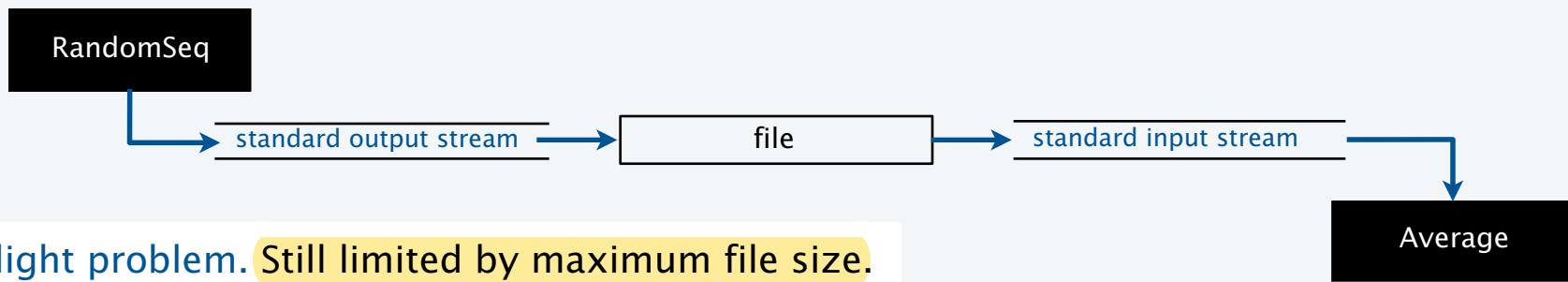
Redirect standard output to a file

```
% java RandomSeq 1000000 > data.txt  
% more data.txt  
0.09474882292442943  
0.2832974030384712  
0.1833964252856476  
0.2952177517730442  
0.8035985765979008  
0.7469424300071382  
0.5835267075283997  
0.3455279612587455  
...
```

Redirect from a file to standard input

```
% java Average < data.txt  
0.4947655567740991
```

"take standard input from"



Piping: entirely avoid saving data

Q. There's no room for a huge file on my computer. Now what?

A. No problem! Use *piping*.

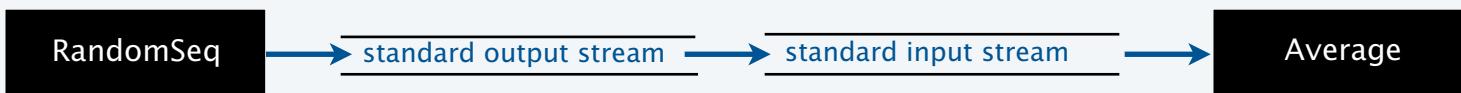
Piping. Connect standard output of one program to standard input of another.

```
% java RandomSeq 1000000 | java Average  
0.4997970473016028
```



```
% java RandomSeq 1000000 | java Average  
0.5002071875644842
```

set up a pipe



Critical point. No limit *within programs* on the amount of data they can handle.

It is the job of the *system* to collect data on standard output and provide it to standard input.

Streaming algorithms

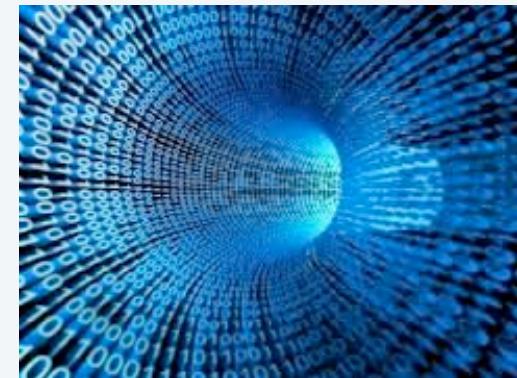
Early computing

- Amount of available memory was much smaller than amount of data to be processed.
- *But* dramatic increases happened every year.
- Redirection and piping enabled programs to handle much more data than computers could store.



Modern computing

- Amount of available memory *is* much smaller than amount of data to be processed.
- Dramatic increases *still* happen every year.
- *Streaming algorithms* enable our programs to handle much more data than our computers can store.



Lesson. Avoid limits *within your program* whenever possible.



COMPUTER SCIENCE

SEGEWICK / WAYNE

PART I: PROGRAMMING IN JAVA

Image sources

<http://www.digitalreins.com/wp-content/uploads/2013/05/Binary-code.jpg>

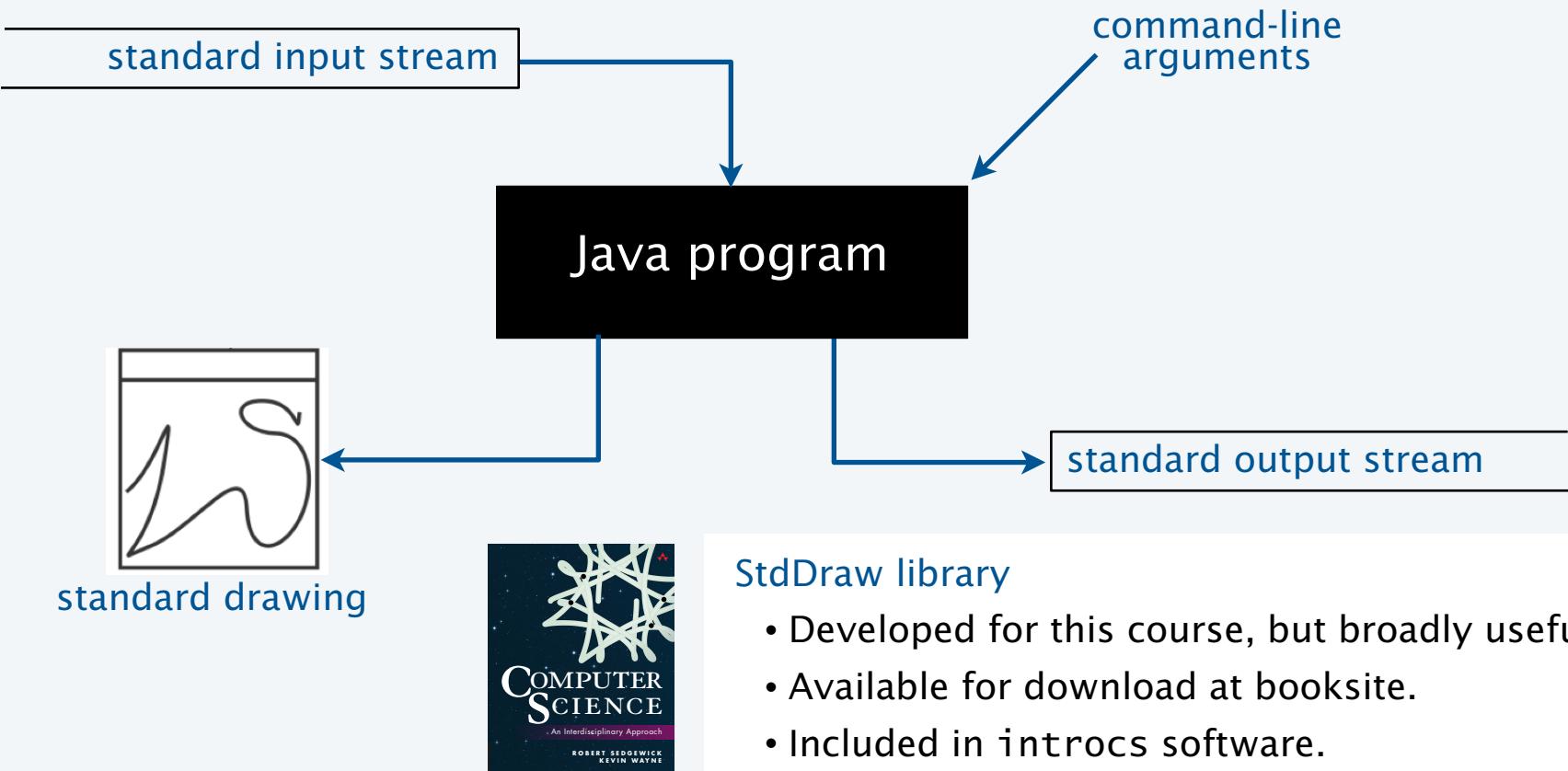
http://en.wikipedia.org/wiki/Punched_tape#mediaviewer/File:Harwell-dekatron-witch-10.jpg

4. Input and Output

- Standard input and output
- **Standard drawing**
- Fractal drawings
- Animation

Further improvements to our I/O abstraction

Add the ability to create a *drawing*.



StdDraw library

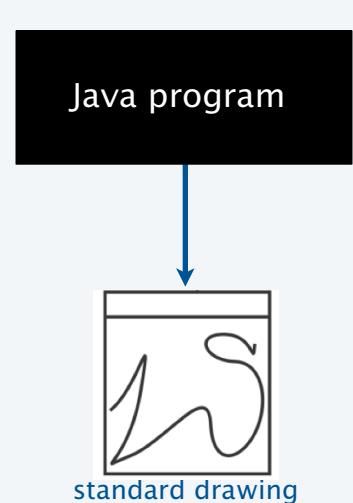
```
public class StdDraw

    void line(double x0, double y0, double x1, double y1)
    void point(double x, double y)
    void text(double x, double y, String s)
    void circle(double x, double y, double r)
    void square(double x, double y, double r)
    void polygon(double x, double y, double r)

    void picture(double x, double y, String filename) place .gif, .jpg or .png file
    void setPenRadius(double r)
    void setPenColor(Color c)

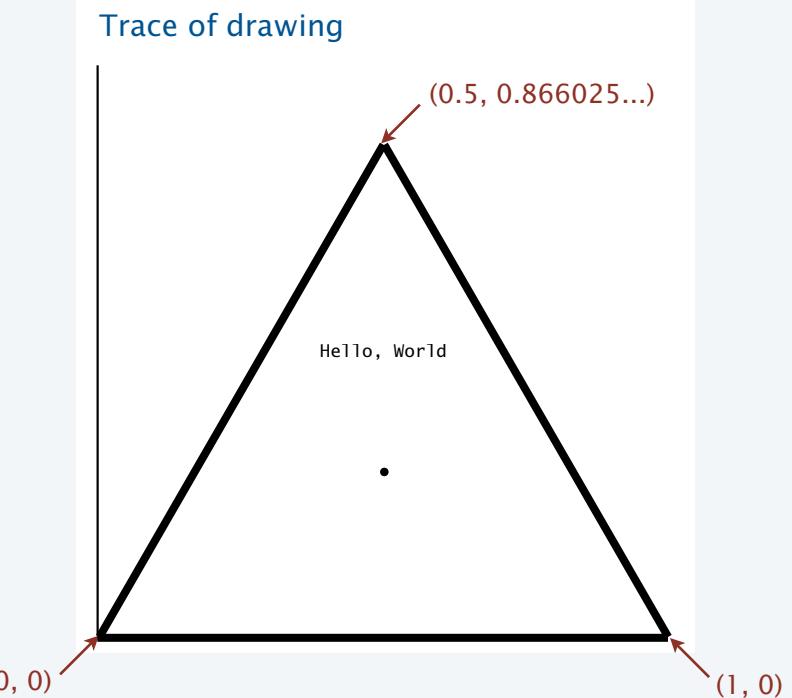
    void setXscale(double x0, double x1) reset x range to [x0, x1]
    void setYscale(double y0, double y1) reset y range to [y0, y1]
    void show(int dt) show all; pause dt millisecs
```

also filledCircle(), filledSquare(),
and filledPolygon()

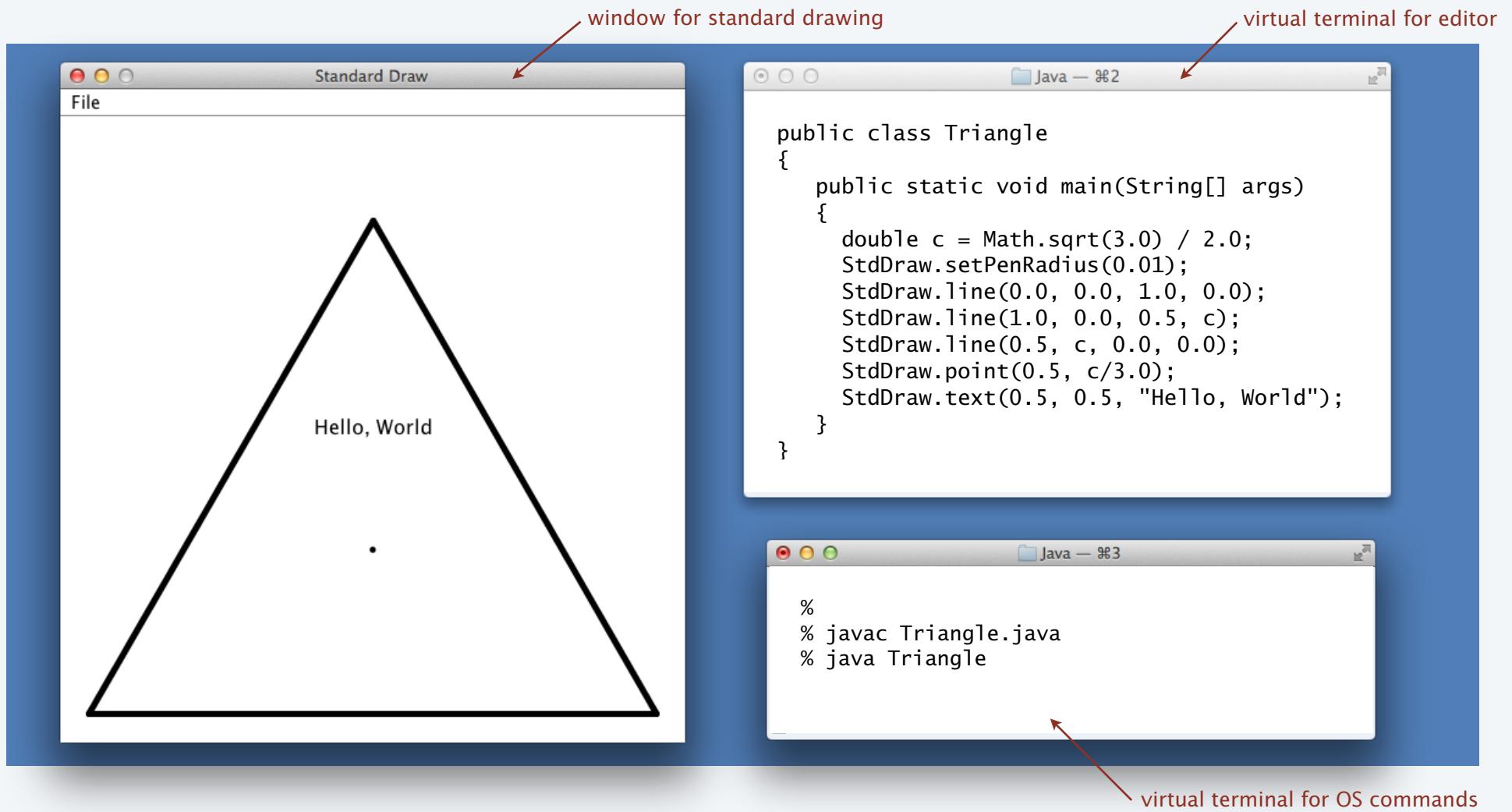


“Hello, World” for StdDraw

```
public class Triangle
{
    public static void main(String[] args)
    {
        double c = Math.sqrt(3.0) / 2.0;
        StdDraw.setPenRadius(0.01);
        StdDraw.line(0.0, 0.0, 1.0, 0.0);
        StdDraw.line(1.0, 0.0, 0.5, c);
        StdDraw.line(0.5, c, 0.0, 0.0);
        StdDraw.point(0.5, c/3.0);
        StdDraw.text(0.5, 0.5, "Hello, World");
    }
}
```



"Hello, World" for StdDraw



StdDraw application: data visualization

```
public class PlotFilter
{
    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);
        }
    }
}
```

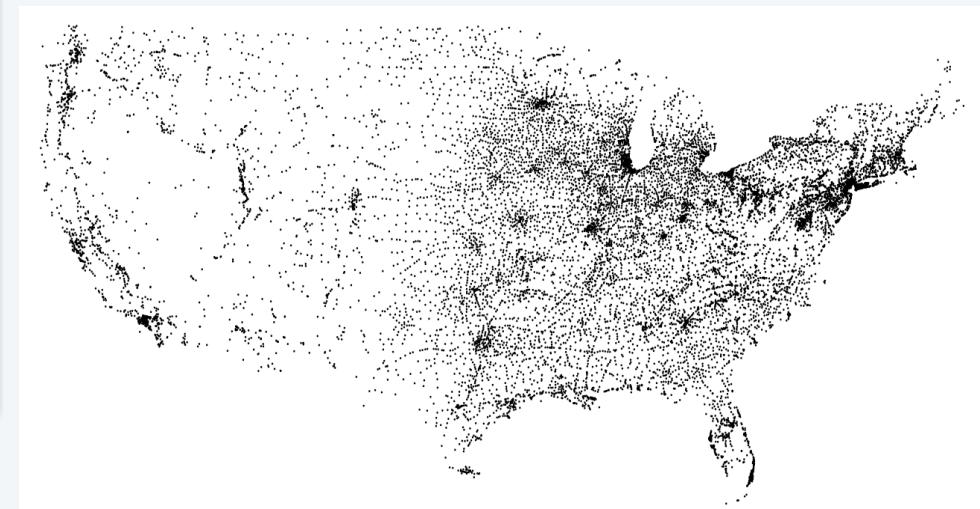
Annotations:

- read coords of bounding box → points to the first four lines of code.
- rescale → points to the two lines of code setting scales.
- read and plot a point → points to the inner loop where a point is plotted.

bounding box coords

```
% more < USA.txt
669905.0 247205.0 1244962.0 490000.0
1097038.8890 245552.7780
1103961.1110 247133.3330
1104677.7780 247205.5560
...
% java PlotFilter < USA.txt
```

sequence of point coordinates (13,509 cities)



StdDraw application: plotting a function

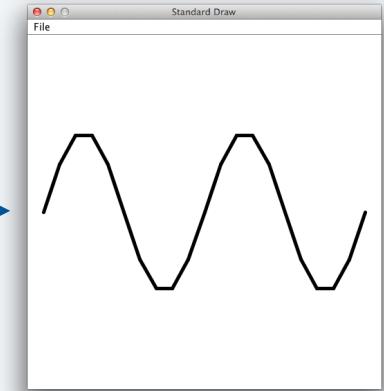
Goal. Plot $y = \sin(4x) + \sin(20x)$ in the interval $(0, \pi)$.

```
% java PlotFunctionEx 20
```

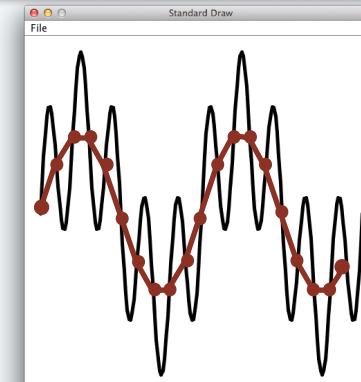
Method. Take N samples, regularly spaced.

```
public class PlotFunctionEx
{
    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]);
    }
}
```

Lesson 1: Plotting is easy. →



```
% java PlotFunctionEx 200
```



Lesson 2: Take a sufficiently large sample—otherwise you might miss something!



COMPUTER SCIENCE
SEGEWICK / WAYNE
PART I: PROGRAMMING IN JAVA

CS.4.B.IO.Drawing

4. Input and Output

- Standard input and output
- Standard drawing
- **Fractal drawings**
- Animation

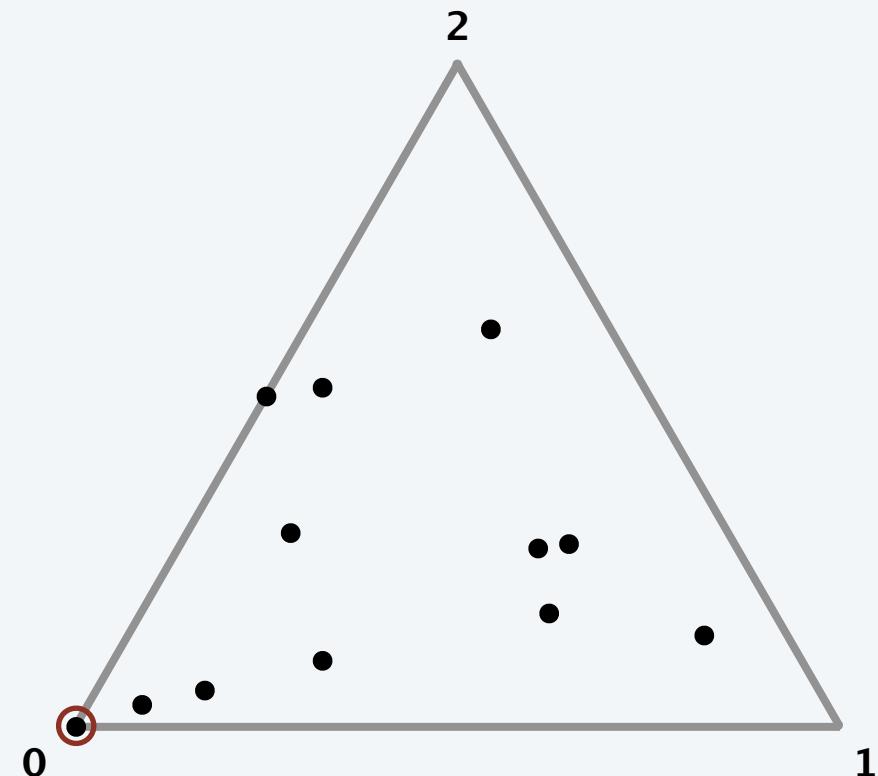
StdDraw application: a random game

Draw an equilateral triangle, number the vertices 0, 1, 2 and make 0 the *current point*.

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

vertex	ID	probability	new x	new y
(0, 0)	0	1/3	.5x	.5y
(1, 0)	1	1/3	.5x + .5	.5y
(.5, $\sqrt{3}/2$)	2	1/3	.5x + .25	.5y + .433

0	1	2	3	4	5	6	7	8	9	10
2	1	2	0	1	0	0	0	2	1	1



StdDraw application: a random game

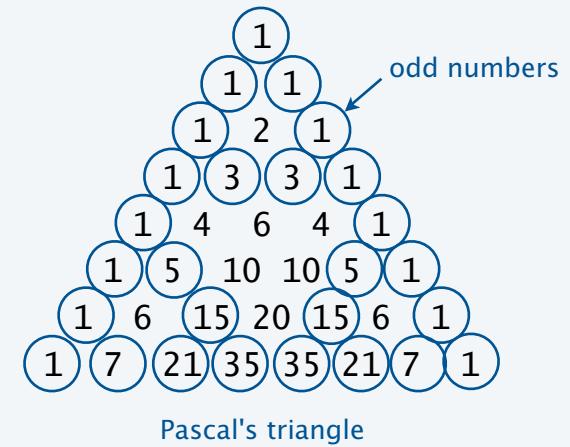
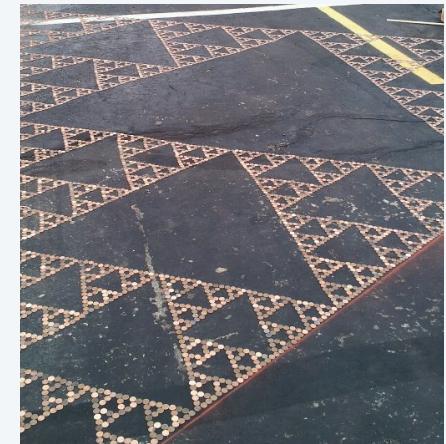
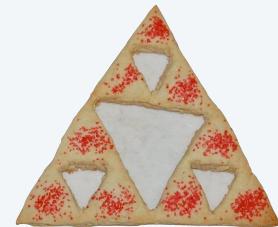
```
public class Chaos
{
    public static void main(String[] args)
    {
        int trials = Integer.parseInt(args[0]);

        double c = Math.sqrt(3.0) / 2.0;
        double[] cx = { 0.000, 1.000, 0.500 };
        double[] cy = { 0.000, 0.000, c };

        StdDraw.setPenRadius(0.01);
        double x = 0.0, y = 0.0;
        for (int t = 0; 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);
        }
    }
}
```

```
% java Chaos 10000
```

Sierpinski triangles in the wild



Iterated function systems

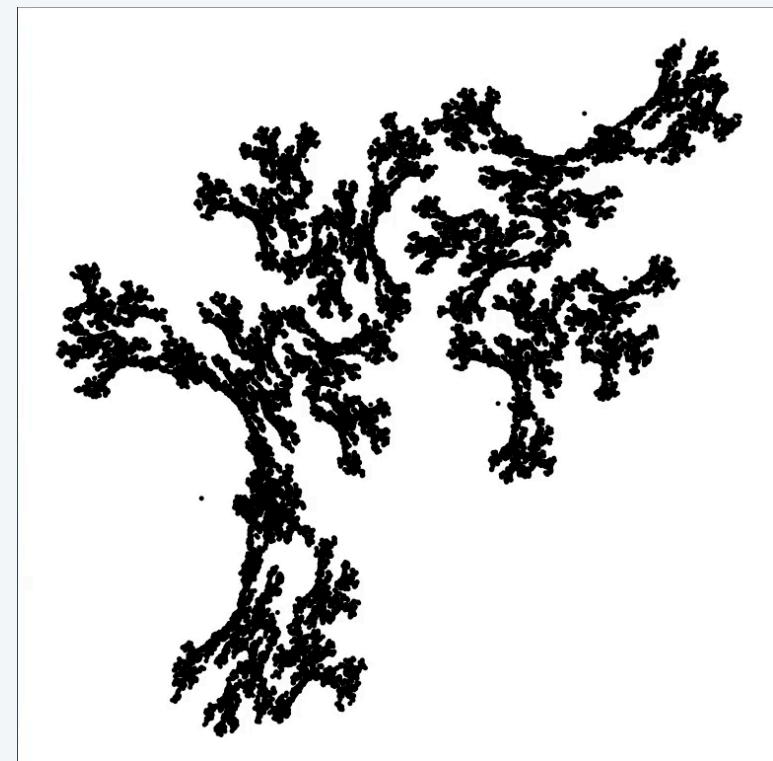
What happens when we change the rules?

probability	new x	new y
40%	.31x - .53y + .89	-.46x - .29y + 1.10
15%	.31x - .08y + .22	.15x - .45y + .34
45%	.55y + .01	.69x - .20y + .38

[IFS.java](#) (Program 2.2.3) is a *data-driven* program that takes the coefficients from *standard input*.

```
% more coral.txt
3
 0.40  0.15  0.45
3 3
 0.307692 -0.531469  0.8863493
 0.307692 -0.076923  0.2166292
 0.000000  0.545455  0.0106363
3 3
-0.461538 -0.293706  1.0962865
 0.153846 -0.447552  0.3383760
 0.692308 -0.195804  0.3808254
```

```
% java IFS 10000 < coral.txt
```



Iterated function systems

Another example of changing the rules

```
% java IFS 10000 < barnsley.txt
```

probability	new x	new y
2%	0.5	.27y
15%	$-.14x + .26y + .57$	$.25x + .22y - .04$
13%	$.17x - .21y + .41$	$.22x + .18y + .09$
70%	$.78x + .03y + .11$	$-.03x + .74y + .27$

```
% more barnsley.txt
4
.02 .15 .13 .70
4 3
.000 .000 .500
-.139 .263 .570
.170 -.215 .408
.781 .034 .1075
4 3
.000 .270 .000
.246 .224 -.036
.222 .176 .0893
-.032 .739 .270
```



Iterated function systems

Simple iterative computations yield patterns that are remarkably similar to those found in the natural world.

Q. What does computation tell us about nature?



an IFS fern

Q. What does nature tell us about computation?



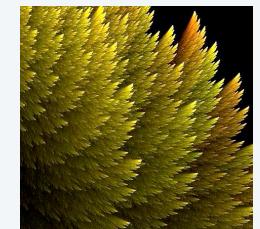
a real fern

20th century sciences. Formulas.



a real plant

21st century sciences. Algorithms?



an IFS plant

Note. You have seen many practical applications of integrated function systems, in movies and games.



A faint, light gray watermark of a fern frond pattern is visible across the entire slide.

COMPUTER SCIENCE

SEGEWICK / WAYNE

PART I: PROGRAMMING IN JAVA

Image sources

<http://paulbourke.net/fractals/gasket/cokegasket.gif>

<http://www.buzzfeed.com/atmccann/11-awesome-math-foods#39wokfk>

<http://sheilakh.deviantart.com/art/The-Legend-of-Sierpinski-308953447>

http://commons.wikimedia.org/wiki/File:Lady_Fern_frond_-_normal_appearance.jpg

http://img3.wikia.nocookie.net/_cb20100707172110/jamescameronsavatar/images/e/e1/Avatar_concept_art-3.jpg

4. Input and Output

- Standard input and output
- Standard drawing
- Fractal drawings
- Animation

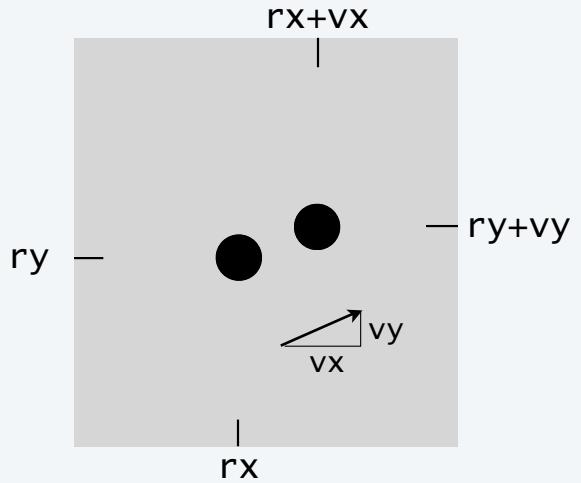
Animation

To create **animation** with StdDraw.

Repeat the following:

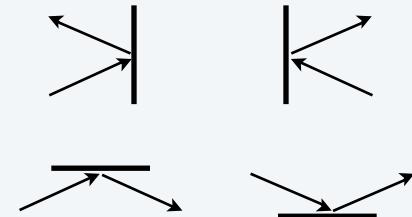
- Clear the screen.
- Move the object.
- Draw the object.
- Display and pause briefly.

When display time is much greater than the screen-clear time, we have the illusion of motion.



Bouncing ball.

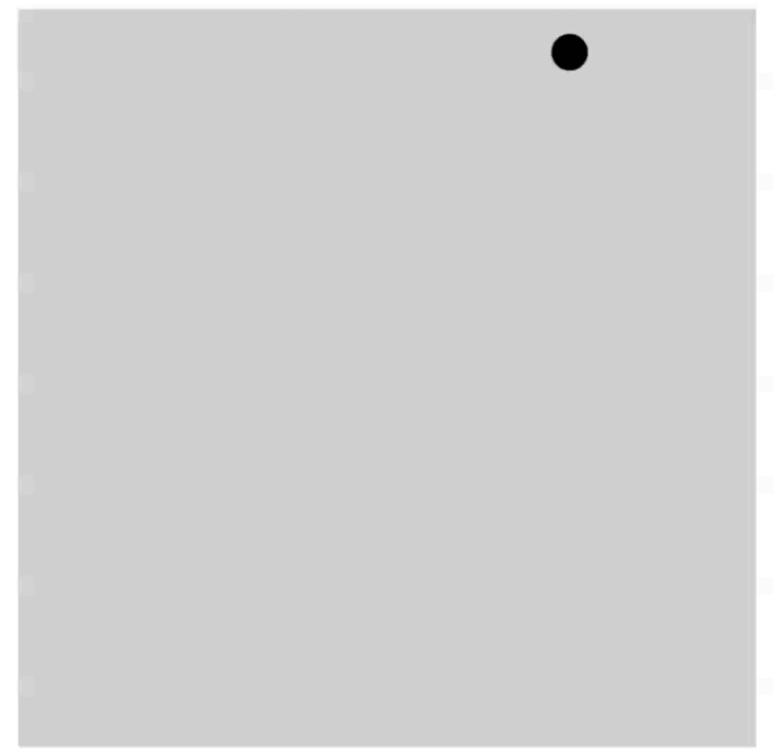
- Ball has position (rx, ry) and constant velocity (vx, vy) .
- To *move* the ball, update position to $(rx+vx, ry+vy)$.
- If the ball hits a *vertical* wall, set vx to $-vx$.
- If the ball hits a *horizontal* wall, set vy to $-vy$.



Bouncing ball

```
public class BouncingBall
{
    public static void main(String[] args)
    {
        double rx = .480, ry = .860;
        double vx = .015, vy = .023;
        double radius = .05;
        StdDraw.setXscale(-1.0, +1.0);
        StdDraw.setYscale(-1.0, +1.0);
        while(true)
        {
            StdDraw.setPenColor(StdDraw.LIGHT_GRAY);
            StdDraw.filledSquare(0.0, 0.0, 1.0);
            if (Math.abs(rx + vx) + radius > 1.0) vx = -vx;
            if (Math.abs(ry + vy) + radius > 1.0) vy = -vy;
            rx = rx + vx;
            ry = ry + vy;
            StdDraw.setPenColor(StdDraw.BLACK);
            StdDraw.filledCircle(rx, ry, radius);
            StdDraw.show(20);
        }
    }
}
```

```
% java BouncingBall
```



Pop quiz on animation

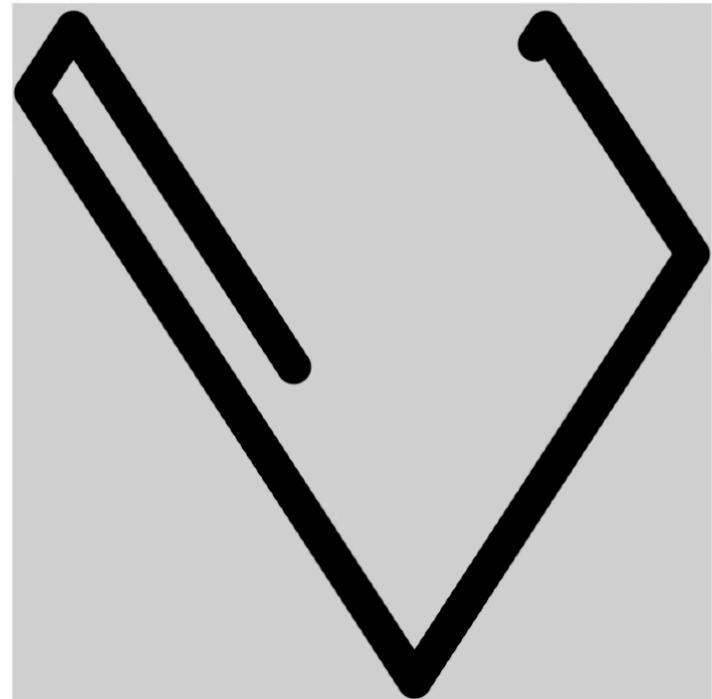
Q. What happens if we move *clear the screen* out of the loop?

```
public class BouncingBall
{
    public static void main(String[] args)
    {
        double rx = .480, ry = .860;
        double vx = .015, vy = .023;
        double radius = .05;
        StdDraw.setXscale(-1.0, +1.0);
        StdDraw.setYscale(-1.0, +1.0);
        while(true)
        {
            StdDraw.setPenColor(StdDraw. LIGHT_GRAY);
            StdDraw.filledSquare(0.0, 0.0, 1.0);
            if (Math.abs(rx + vx) + radius > 1.0) vx = -vx;
            if (Math.abs(ry + vy) + radius > 1.0) vy = -vy;
            rx = rx + vx;
            ry = ry + vy;
            StdDraw.setPenColor(StdDraw.BLACK);
            StdDraw.filledCircle(rx, ry, sz);
            StdDraw.show(20);
        }
    }
}
```

Pop quiz on animation

Q. What happens if we move *clear the screen* out of the loop?

```
public class BouncingBall
{
    public static void main(String[] args)
    {
        double rx = .480, ry = .860;
        double vx = .015, vy = .023;
        double radius = .05;
        StdDraw.setXscale(-1.0, +1.0);
        StdDraw.setYscale(-1.0, +1.0);
        StdDraw.setPenColor(StdDraw. LIGHT_GRAY);
        StdDraw.filledSquare(0.0, 0.0, 1.0);
        while(true)
        {
            if (Math.abs(rx + vx) + radius > 1.0) vx = -vx;
            if (Math.abs(ry + vy) + radius > 1.0) vy = -vy;
            rx = rx + vx;
            ry = ry + vy;
            StdDraw.setPenColor(StdDraw. BLACK);
            StdDraw.filledCircle(rx, ry, radius);
            StdDraw.show(20);
        }
    }
}
```

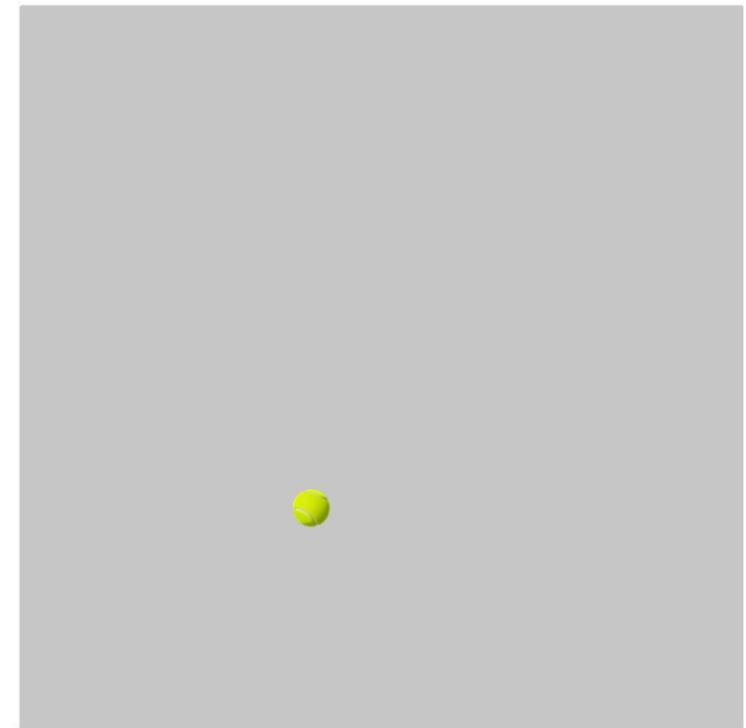


A. We see the ball's entire path.

Deluxe bouncing ball

```
public class BouncingBallDeluxe
{
    public static void main(String[] args)
    {
        double rx = .480, ry = .860;
        double vx = .015, vy = .023;
        double radius = .05;
        StdDraw.setXscale(-1.0, +1.0);
        StdDraw.setYscale(-1.0, +1.0);
        while(true)
        {
            StdDraw.setPenColor(StdDraw. LIGHT_GRAY);
            StdDraw.filledSquare(0.0, 0.0, 1.0);
            if (Math.abs(rx + vx) + radius > 1.0)
            { StdAudio.play("pipebang.wav"); vx = -vx; }
            if (Math.abs(ry + vy) + radius > 1.0)
            { StdAudio.play("pipebang.wav"); vy = -vy; }
            rx = rx + vx;
            ry = ry + vy;
            StdDraw.picture(rx, ry, "TennisBall.png");
            StdDraw.show(20);
        }
    }
}
```

```
% java BouncingBallDeluxe
```

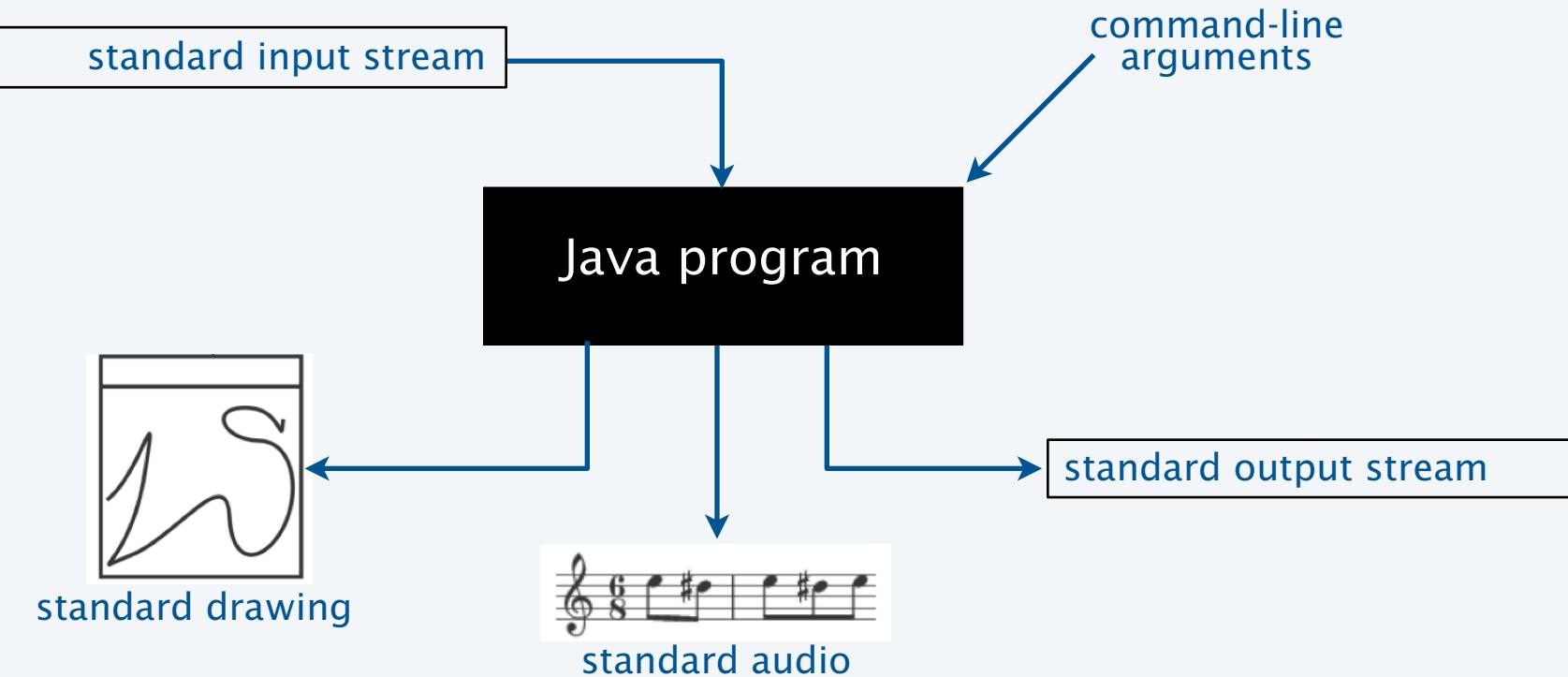
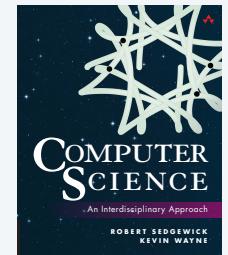


Stay tuned to next lecture for full description of StdAudio.

A set of I/O abstractions for Java

Developed for this course, but broadly useful

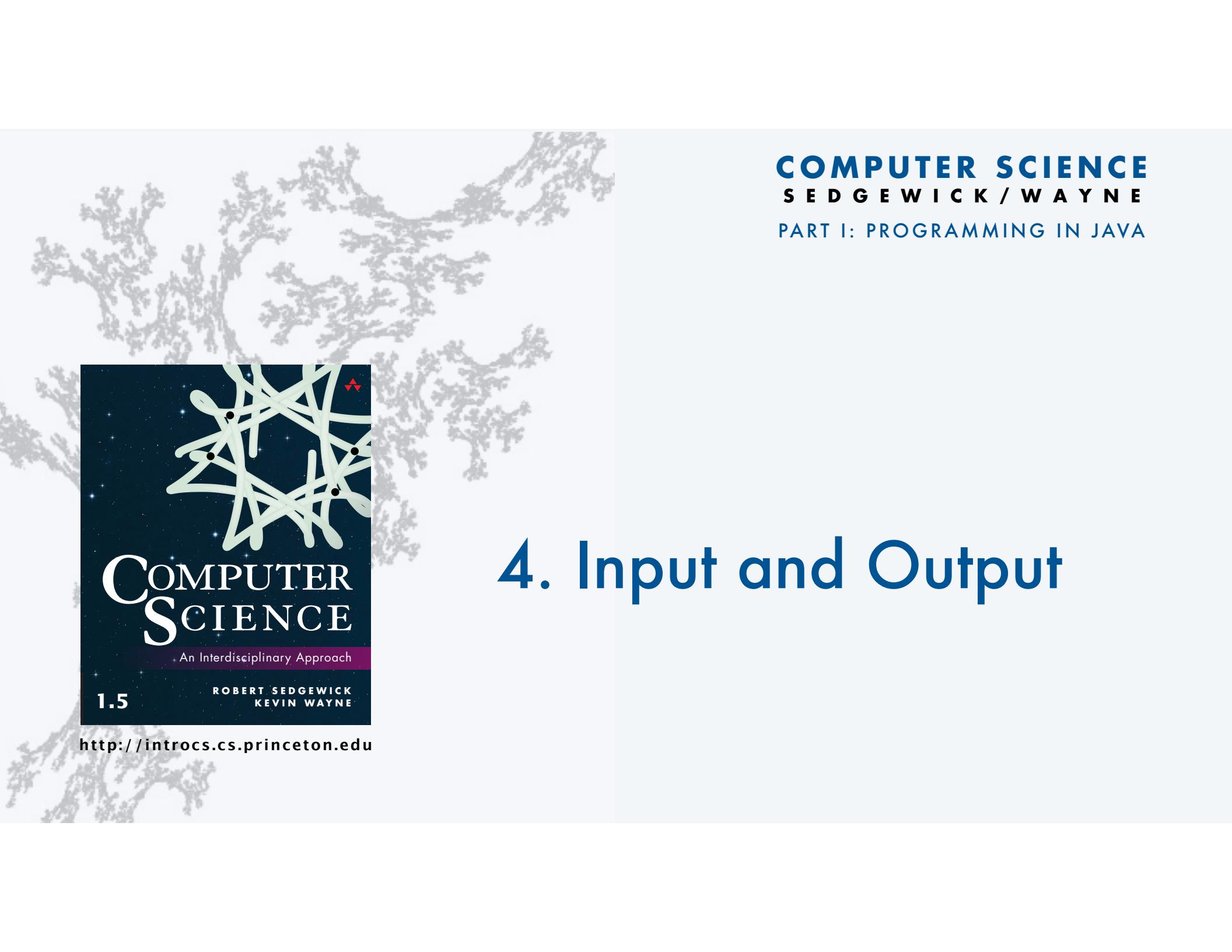
- StdIn, StdOut, StdDraw, and StdAudio.
- Available for download at booksite (and included in introcs software).



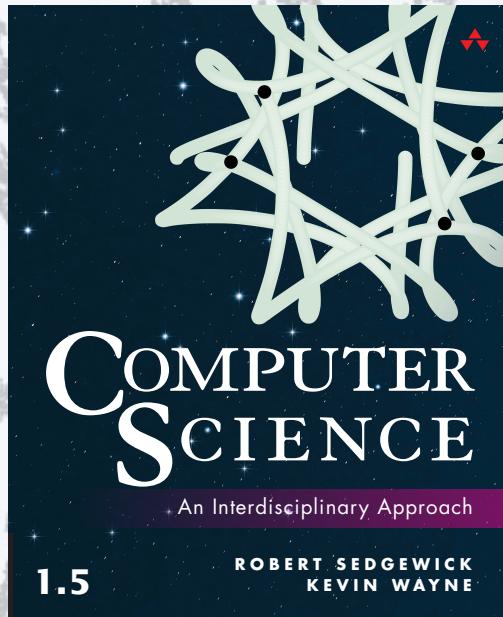


COMPUTER SCIENCE
SEGEWICK / WAYNE
PART I: PROGRAMMING IN JAVA

CS.4.D.IO.Animation



COMPUTER SCIENCE
SEGEWICK / WAYNE
PART I: PROGRAMMING IN JAVA



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

4. Input and Output