4. Input and Output
4. Input and Output

- Standard input and output
- Standard drawing
- Fractal drawings
- Animation
Basic building blocks for programming

any program you might want to write

- objects
- functions and modules
- graphics, sound, and image I/O
- arrays
- conditionals and loops
- Math
- text I/O
- primitive data types
- assignment statements

Ability to interact with the outside world
Input and output

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

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.

An abstraction is something that exists only as an idea.

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.

Interested in thinking more deeply about this concept? Consider taking a philosophy course.

This lecture. Abstractions for delivering input to or receiving output from our programs.
Terminal. An abstraction for providing input and output to a program.

Quick review
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.

```java
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.

```java
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

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

No limit on amount of output
...
Improved input-output abstraction

Add an infinite *input* stream.

---

**standard input stream**

**Java program**

**command-line arguments**

**standard output 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).
Developed for this course, but broadly useful

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

<table>
<thead>
<tr>
<th>public class StdIn</th>
<th>method</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>boolean isEmpty()</td>
<td>true if no more values</td>
<td></td>
</tr>
<tr>
<td>int readInt()</td>
<td>read a value of type int</td>
<td></td>
</tr>
<tr>
<td>double readDouble()</td>
<td>read a value of type double</td>
<td></td>
</tr>
<tr>
<td>long readLong()</td>
<td>read a value of type long</td>
<td></td>
</tr>
<tr>
<td>boolean readBoolean()</td>
<td>read a value of type boolean</td>
<td></td>
</tr>
<tr>
<td>char readChar()</td>
<td>read a value of type char</td>
<td></td>
</tr>
<tr>
<td>String readString()</td>
<td>read a value of type String</td>
<td></td>
</tr>
<tr>
<td>String readAll()</td>
<td>read the rest of the text</td>
<td></td>
</tr>
</tbody>
</table>
StdOut library

Developed for this course, but broadly useful
- Implement abstractions invented for UNIX in the 1970s.
- Available for download at booksite.
- Included in introcs software you downloaded at the beginning of the course.

<table>
<thead>
<tr>
<th>public class StdOut</th>
</tr>
</thead>
<tbody>
<tr>
<td>void print(String s)</td>
</tr>
<tr>
<td>void println()</td>
</tr>
<tr>
<td>void println(String s)</td>
</tr>
<tr>
<td>void printf(String f, ...)</td>
</tr>
</tbody>
</table>

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.
StdIn/StdOut warmup

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

```java
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).

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

```java
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);
    }
}
```

% 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**

```java
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**

```java
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

Slight problem. Still limited by maximum file size.
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
```

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

Lesson. Avoid limits \textit{within your program} whenever possible.
Image sources

http://www.digitalreins.com/wp-content/uploads/2013/05/Binary-code.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**
- Developed for this course, but broadly useful.
- Available for download at booksite.
- Included in *introcs* software.
## StdDraw library

```java
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()
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

```
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");
    }
}
```

window for standard drawing

virtual terminal for editor

virtual terminal for OS commands
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);
        }
    }
}

% 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
StdDraw application: plotting a function

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

**Method.** Take \( N \) samples, regularly spaced.

```java
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.

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

<table>
<thead>
<tr>
<th>vertex</th>
<th>ID</th>
<th>probability</th>
<th>new x</th>
<th>new y</th>
</tr>
</thead>
<tbody>
<tr>
<td>(0, 0)</td>
<td>0</td>
<td>1/3</td>
<td>.5x</td>
<td>.5y</td>
</tr>
<tr>
<td>(1, 0)</td>
<td>1</td>
<td>1/3</td>
<td>.5x + .5</td>
<td>.5y</td>
</tr>
<tr>
<td>(.5, √3/2)</td>
<td>2</td>
<td>1/3</td>
<td>.5x + .25</td>
<td>.5y + .433</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
**StdDraw application: a random game**

```java
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?

<table>
<thead>
<tr>
<th>probability</th>
<th>new x</th>
<th>new y</th>
</tr>
</thead>
<tbody>
<tr>
<td>40%</td>
<td>$0.31x - 0.53y + 0.89$</td>
<td>$-0.46x - 0.29y + 1.10$</td>
</tr>
<tr>
<td>15%</td>
<td>$0.31x - 0.08y + 0.22$</td>
<td>$0.15x - 0.45y + 0.34$</td>
</tr>
<tr>
<td>45%</td>
<td>$0.55y + 0.01$</td>
<td>$0.69x - 0.20y + 0.38$</td>
</tr>
</tbody>
</table>

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

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

```bash
% 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
```
Iterated function systems

Another example of changing the rules

<table>
<thead>
<tr>
<th>probability</th>
<th>new x</th>
<th>new y</th>
</tr>
</thead>
<tbody>
<tr>
<td>2%</td>
<td>0.5</td>
<td>.27y</td>
</tr>
<tr>
<td>15%</td>
<td>−.14x + .26y + .57</td>
<td>.25x + .22y - .04</td>
</tr>
<tr>
<td>13%</td>
<td>.17x − .21y + .41</td>
<td>.22x + .18y + .09</td>
</tr>
<tr>
<td>70%</td>
<td>.78x + .03y + .11</td>
<td>−.03x + .74y + .27</td>
</tr>
</tbody>
</table>

% 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

% java IFS 10000 < barnsley.txt
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?
Q. What does nature tell us about computation?

20th century sciences. Formulas.

21st century sciences. Algorithms?

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

an IFS fern

a real fern

a real plant

an IFS plant
Image sources

http://paulbourke.net/fractals/gasket/cokegasket.gif
http://www.buzzfeed.com/atmccann/11-awesome-math-foods#39wokfk
http://commons.wikimedia.org/wiki/File:Lady_Fern_frond_-_normal Appearance.jpg
http://img3.wikia.nocookie.net/__cb20100707172110/jamescameronsavatar/images/e/e1/Avalar_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

```java
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);
        }
    }
}
```
Pop quiz on animation

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

```java
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?

```java
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

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

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).
4. Input and Output