INTRODUCTION TO
COMPUTER SCIENCE

Rutgers University

Input / Output



http://introcs.cs.rutgers.edu

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

I/O Library

```
StdIn.java
StdOut.java
StdDraw.java
```

- The source code files listed above will become part of our standard library moving forward.
- Download them from our Canvas folder, the book site, or the course site.
- Save them in your project folders adjacent to your source code. We will use them extensively in the remainder of the course.

Book site URLs:

- https://introcs.cs.princeton.edu/java/stdlib/StdIn.java
- https://introcs.cs.princeton.edu/java/stdlib/StdOut.java
- https://introcs.cs.princeton.edu/java/stdlib/StdDraw.java

INTRODUCTION TO
COMPUTER SCIENCE
Rutgers University

Input and Output

- Standard input and output
- Standard drawing

Adopted and modified from slides by Sedgwick and Wayne

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

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

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

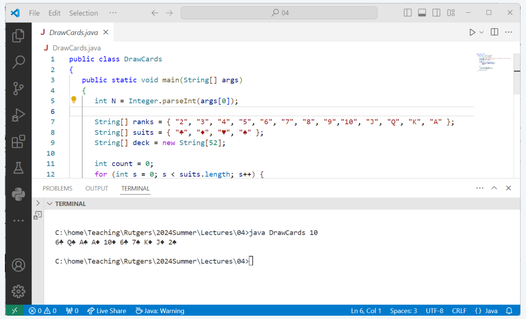
Example: "Printing" is the idea of a program producing text as output.

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.

Abstractions plays an *essential* role in understanding computation.

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

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

Input from command line

DrawCards.java

```
1   public class DrawCards
2   {
3      public static void main(String[] args)
4      {
5         int N = Integer.parseInt(args[0]);
6
7         String[] ranks = { "2", "3", "4", "5", "6", "7", "8", "9","10", "J", "Q", "K", "A" };
8         String[] suits = { "♣", "♦", "♥", "♠" };
9         String[] deck = new String[52];
10
11        int count = 0;
12        for (int s = 0; s < suits.length; s++) {
```

PROBLEMS OUTPUT TERMINAL

TERMINAL

```
C:\home\Teaching\Rutgers\2024Summer\Lectures\04>java DrawCards 10
6♣ Q♠ A♠ A♦ 10♦ 6♣ 7♠ K♦ J♦ 2♠

C:\home\Teaching\Rutgers\2024Summer\Lectures\04>
```

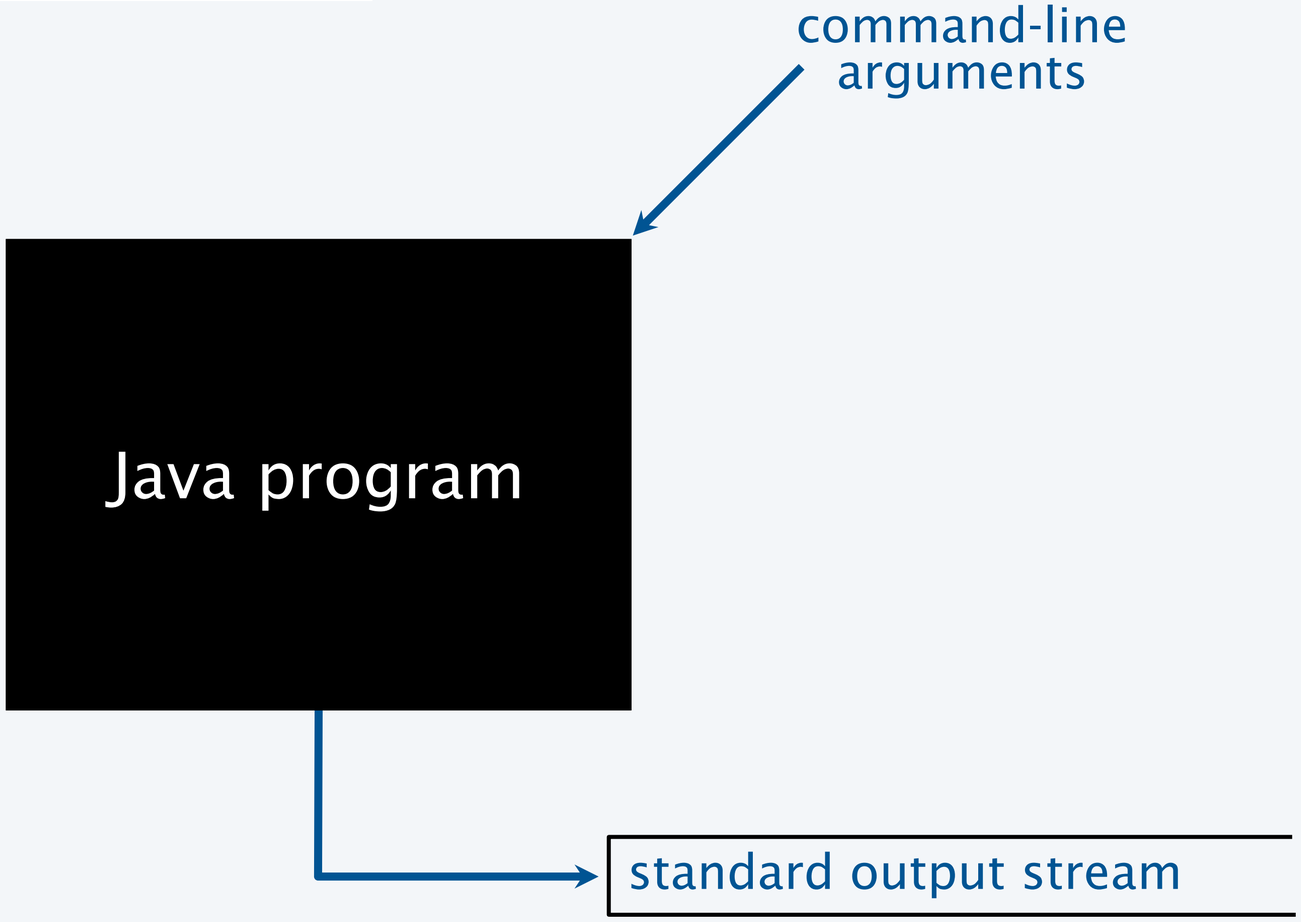
Output to standard output stream



Virtual VT-100 terminal

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.
- Must call **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());
   }
}
```

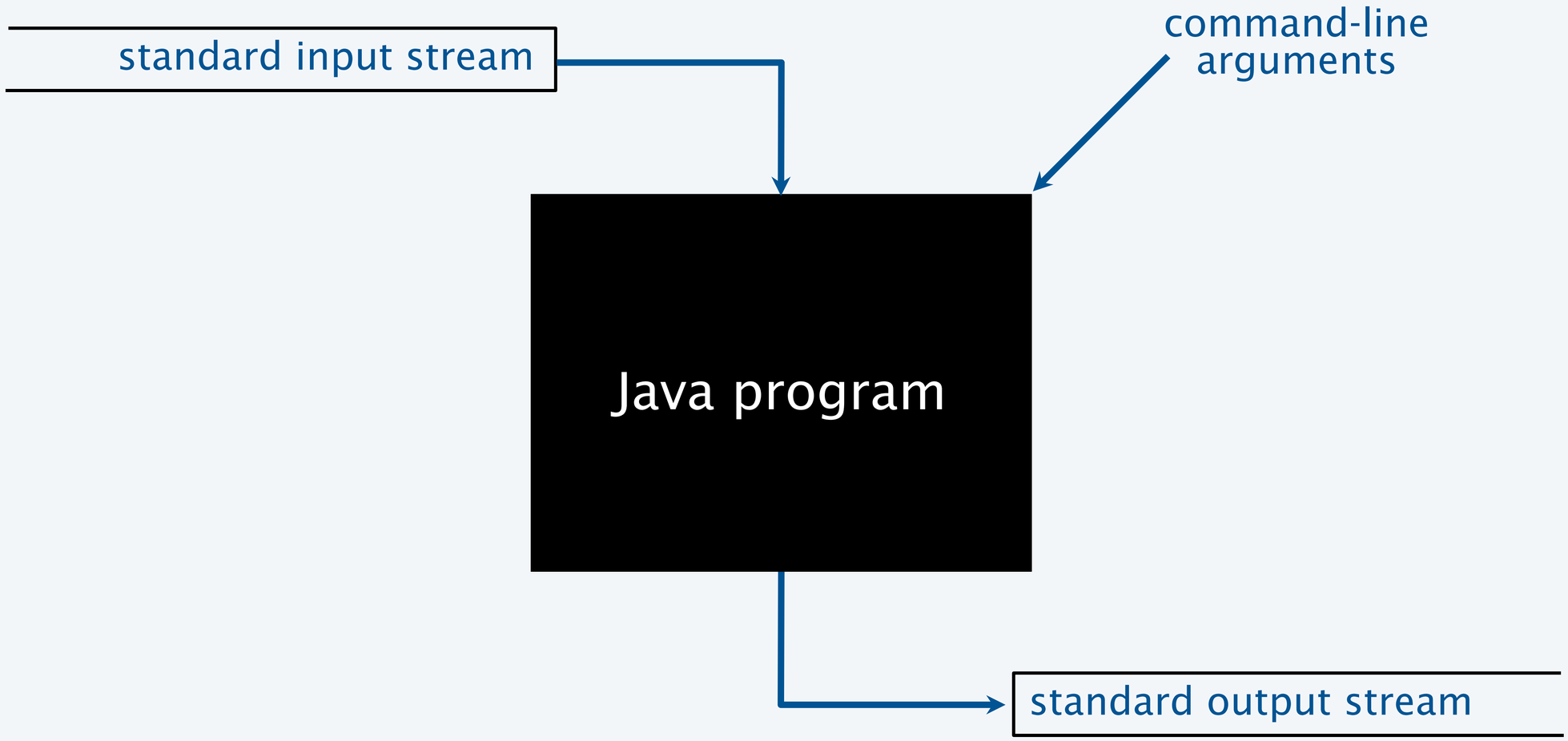
```
% java RandomSeq 4
0.9320744627218469
0.4279508713950715
0.08994615071160994
0.6579792663546435
```

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

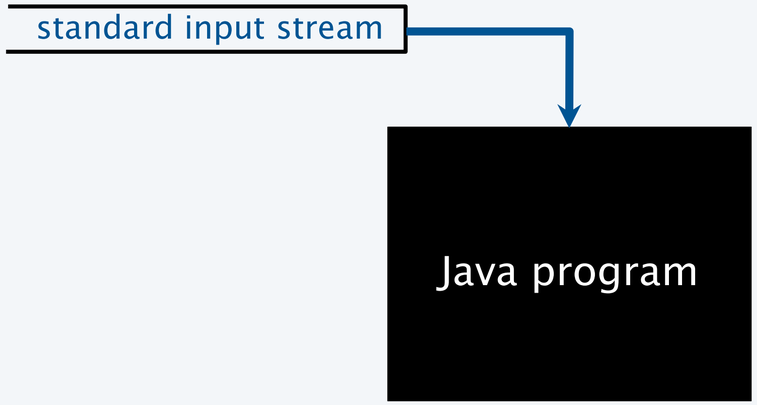
Add an infinite *input* stream.



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

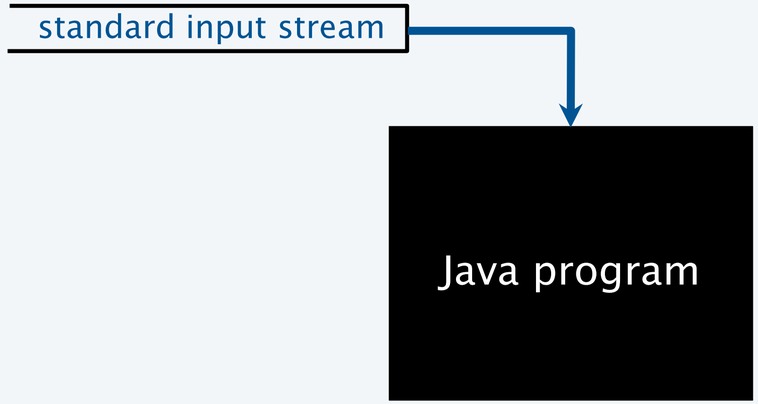
- Can provide new arguments ***while*** **the program is executing**.
- **No limit** on the amount of data we can input to a program.
- Conversion to primitive types is **handled explicitly** (stay tuned).

StdIn library (book version)

Developed for this book, but broadly useful

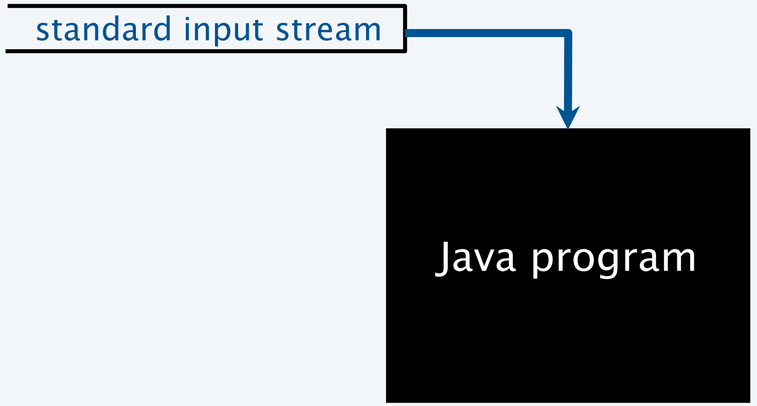
- Implement abstractions invented for UNIX in the 1970s.
- Available for download at book site.

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



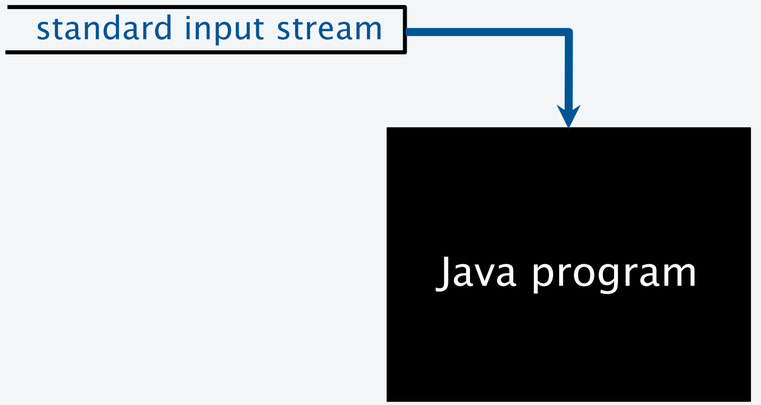
StdIn library (**processing chars and Strings**)

public class StdIn	
boolean hasNextChar()	true *iff standard input has more characters*
char readChar()	*read a value of type* char and returns
boolean hasNextLine()	true *iff standard input has more input lines*
String readLine()	*read remaining portion of line as type* String
String readAll()	*read remaining portion of input as type String*



StdIn library

public class StdIn	
double[] readAllDoubles()	*read all remaining doubles and return in an array*
int[] readAllInts()	*read all remaining ints and return in an array*
long[] readAllLongs()	*read all remaining longs and return in an array*
String[] readAllStrings()	*read all remaining Strings and return in an array*
String[] readAllLines()	*read all remaining lines and return in an array of Strings*

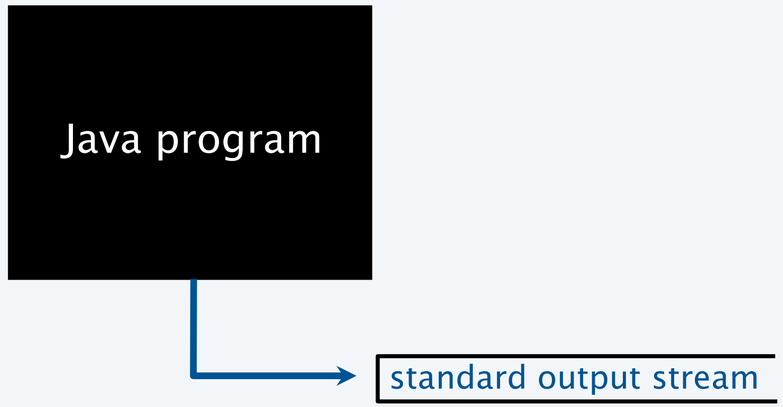


StdOut library

Developed for this course, but broadly useful

- Implement abstractions invented for UNIX in the 1970s.

public class StdOut	
void print(String s)	*put* s *on the output stream*
void println()	*put a newline on the output stream*
void println(String s)	*put* s*, then a newline on the stream*
void printf(String f, ...)	*formatted output*



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.

```
// AddTwo.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.

```
// Average.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  32.0  3.0  7.0
  <Ctrl-d>
  10.5
```

Summary: prototypical applications of standard output and standard input

StdOut: Generate a stream of random numbers

```
// RandomSeq.java
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

```
// Average.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);
   }
}
```

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

LO 6.1j LO 6.1k

Redirect standard output to a file

```
% java RandomSeq 1000000 > data.txt

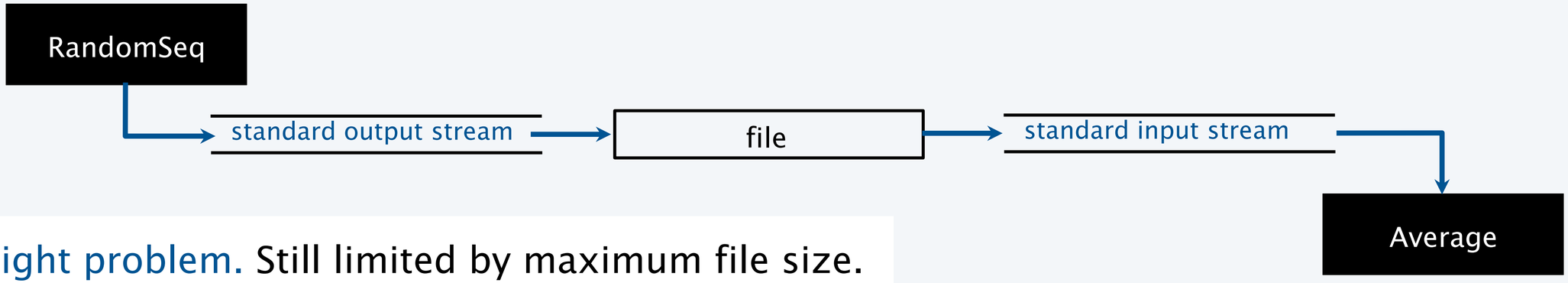
% more data.txt
0.09474882292442943
0.2832974030384712
0.1833964252856476
0.2952177517730442
0.8035985765979008
0.7469424300071382
0.5835267075283997
0.3455279612587455
...
```

"redirect standard output to"

Redirect from a file to standard input

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

"take standard input from"



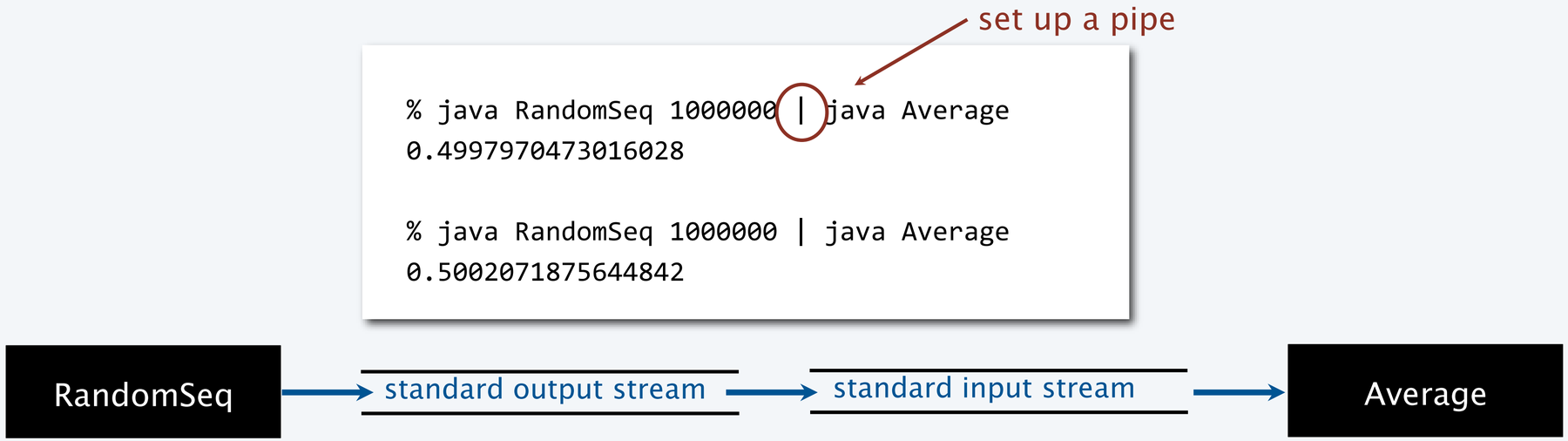
Slight problem. Still limited by maximum file size.

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.



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.

Revisit 2D Problem—CHALLENGE #3

Consider a line of students waiting to be seated in a classroom.

- This is represented by an array of Strings (names)
- Consider the classroom is a rectangular array of seats. (5 x 4 array of Strings).
- Fill this classroom with the student waiting in line. Fill the seats in ROW-MAJOR order.

X X X X
X X X X
X X X X
X X X X
X X X X

Students will then leave the classroom in COL-MAJOR order.

Too tedious to enter, and too prone to mistakes

```
$ javac Classroom.java
$ java Classroom 4 3 ana andy barb danny enzo maria fran lars juan perry nan sai
Students:   ana andy barb danny enzo maria fran lars juan perry nan sai

classroom
ana     andy    barb
danny   enzo    maria
fran    lars    juan
perry   nan     sai
students leaving room by column: ana danny fran perry andy enzo lars nan barb maria juan sai
$
```

Let's look at our Classroom again! *READ NAMES FROM A TEXT FILE!*

These are the ONLY changes to make to Classroom.java!!!

Save as Classroom2.java

*Copy **names.txt** to your program folder*

```
// for (int count = 0; count < numStudents; count++ ) {
//       studentLine[count] = args[count+ 2]; //get the students into the line (1D Array)
// }

int index = 0;
while (!StdIn.isEmpty() && index < numStudents){
     String s = StdIn.readString();
     studentLine[index] = s;
     index++;
}
```

javac Classroom2.java

java Classroom2 5 4 < names.txt

Revisit 2D Problem—CHALLENGE #3

Consider a line of students waiting to be seated in a classroom.

- This is represented by an array of Strings (names)
- Consider the classroom is a rectangular array of seats. (5 x 4 array of Strings).
- Fill this classroom with the student waiting in line. Fill the seats in ROW-MAJOR order.

X X X X
X X X X
X X X X
X X X X
X X X X

Students will then leave the classroom in COL-MAJOR order.

javac Classroom2.java
java Classroom2 5 4 < names.txt

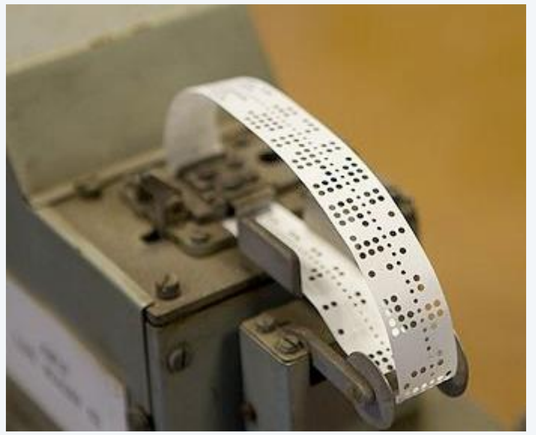
```
$ javac Classroom2.java
$ java Classroom2 5 4 < names.txt
Students:   joe mary tomm alisha edwardo dana rohan shahir raul nicole hasham dylan zikun keisha lisa marly
owen elsa julian daniel

joe             mary            tomm            alisha
edwardo         dana            rohan           shahir
raul            nicole          hasham          dylan
zikun           keisha          lisa            marly
owen            elsa            julian          daniel
students leaving room by column: joe edwardo raul zikun owen mary dana nicole keisha elsa tomm rohan hasham
lisa julian alisha shahir dylan marly daniel
$
```

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.

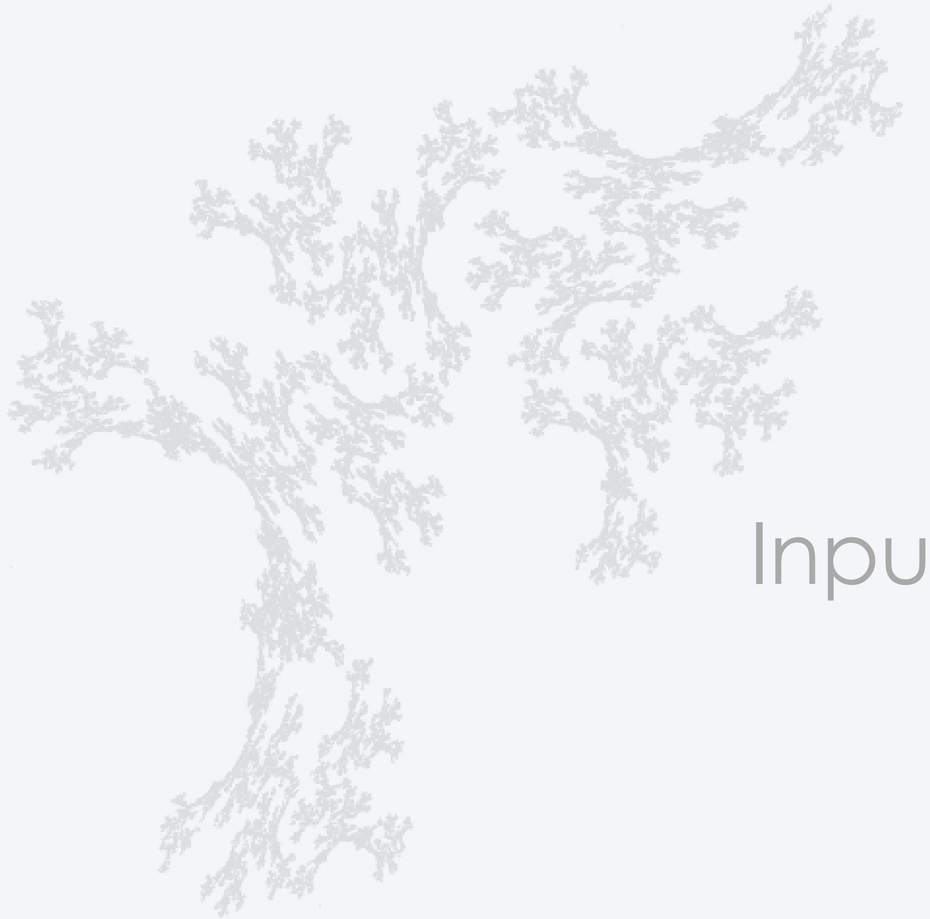


Modern computing

- 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.

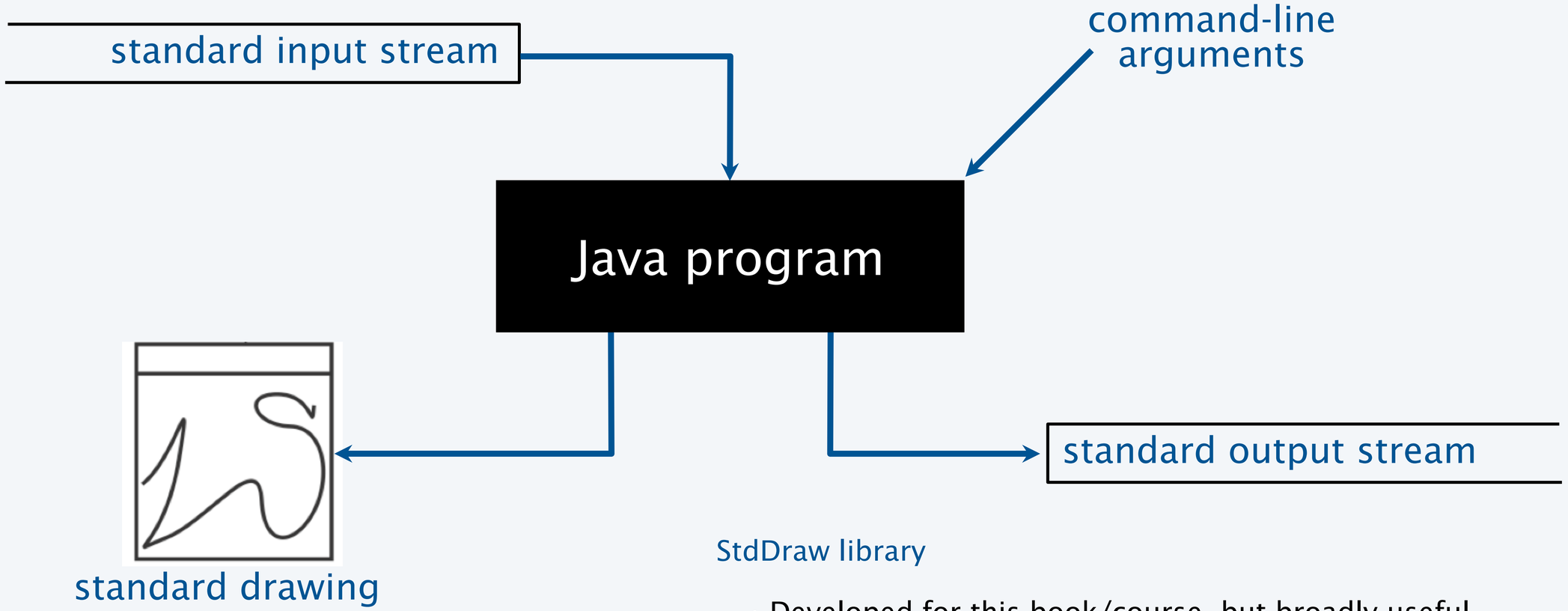


Input and Output

- Standard input and output
- Standard drawing

Further improvements to our I/O abstraction

Add the ability to create a *drawing*.



StdDraw library

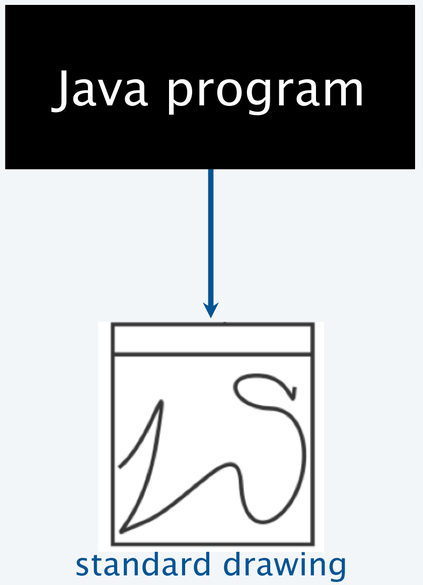
- Developed for this book/course, but broadly useful.
- Available for download at booksite.
- Included in introcs software.

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* [x_0, x_1)
void setYscale(double y0, double y1)	*reset y range to* [y_0, y_1)
void show(int dt)	*show all; pause* dt *millisecs*

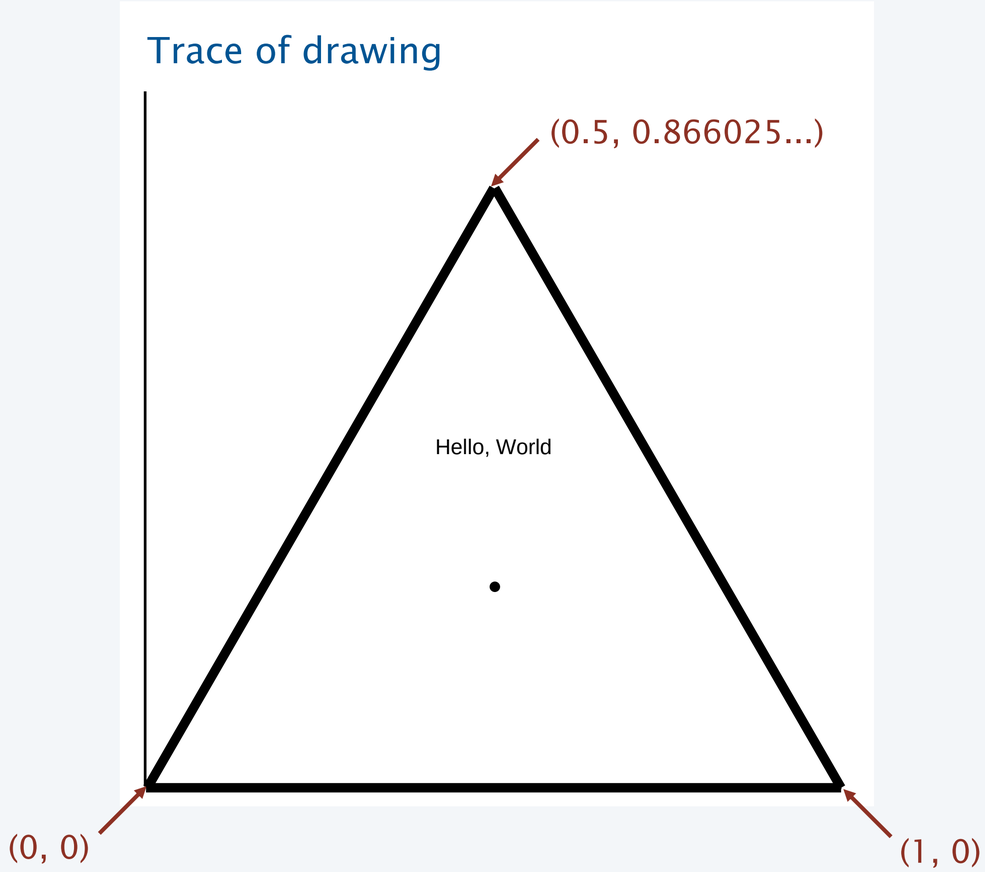


also filledCircle(), filledSquare(), and filledPolygon()



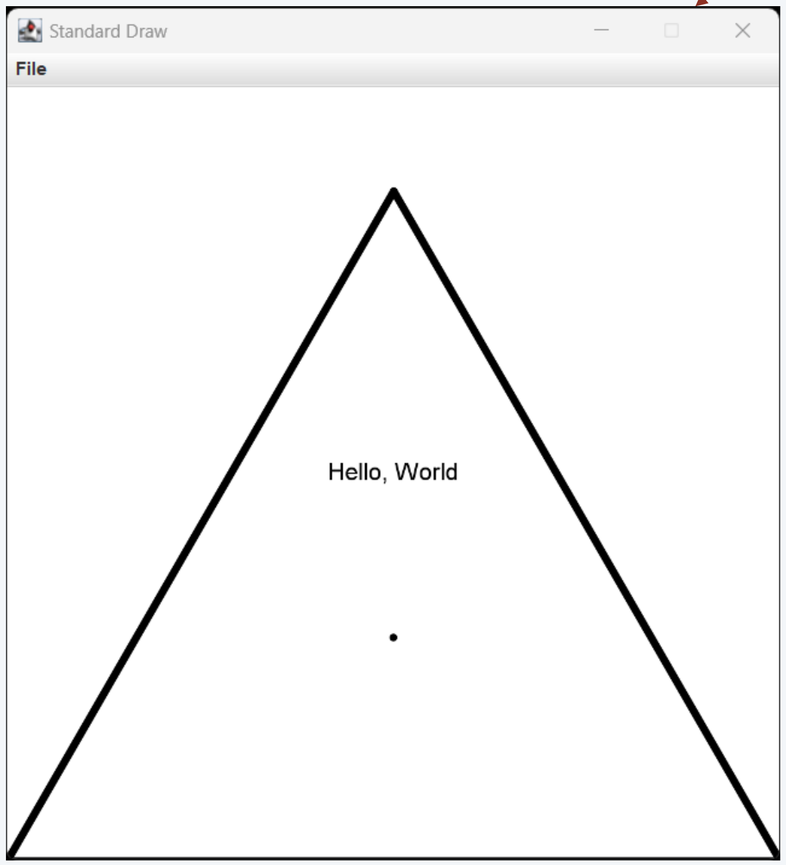
“Hello, World” for StdDraw

```
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");
   }
}
```

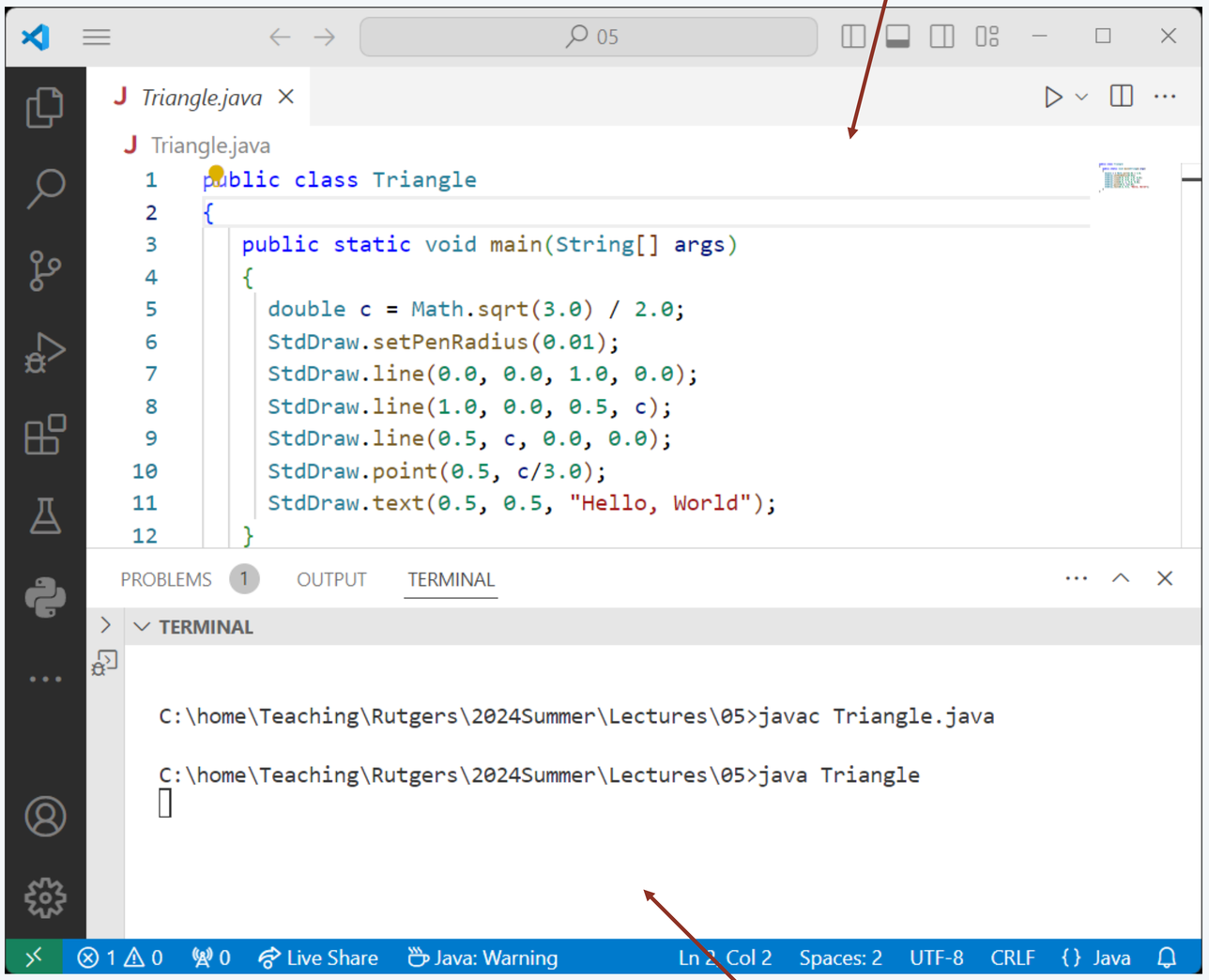


"Hello, World" for StdDraw

window for standard drawing



virtual terminal for editor



virtual terminal for OS commands

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(xmax, xmin); // Swap
        StdDraw.setYscale(ymin, ymax);

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

read coords of bounding box

rescale

read and plot a point

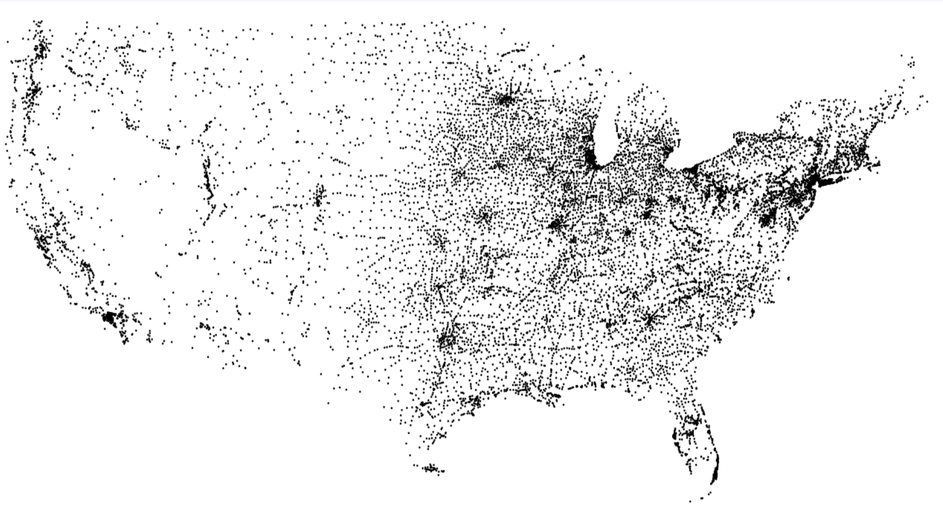
more command views text file USA.txt
On Windows, use type USA.txt

bounding box coords

```
% more < USA.txt
669905.0 164139.5 1244962.0 573065.5
817827.778 245552.778
810905.556 247133.333
810188.889 247205.556
...

% java PlotFilter < USA.txt
```

sequence of point coordinates (all cities)



StdDraw application: plotting a function

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

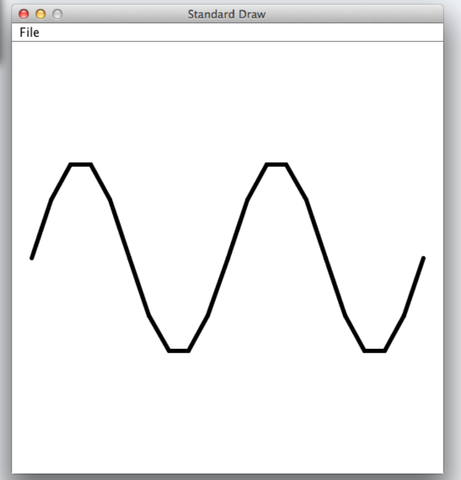
Method. Take *N* samples, regularly spaced.

```
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]);
      }

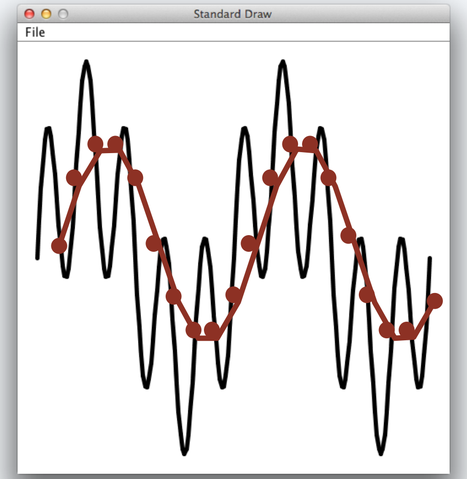
      StdDraw.setXscale(0, Math.PI);
      StdDraw.setYscale(-2.0, +2.0);
      for (int i = 0; i < N; i++)
         StdDraw.line(x[i], y[i], x[i+1], y[i+1]);
   }
}
```

```
% java PlotFunctionEx 20
```



Lesson 1: Plotting is easy. →

```
% java PlotFunctionEx 200
```



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

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

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



Input and Output

- Standard input and output
- Standard drawing
- Fractal drawings

INTRODUCTION TO COMPUTER SCIENCE

Rutgers University



Input / Output

http://introcs.cs.rutgers.edu