Introduction to Programming

Week 5

Magnus Madsen

The Study Cafe

PSA: The study cafe is now staffed on *Mondays* (12:15) and *Fridays* (12:15).

Time	**Teaching Assistant**
Monday 12:15	Tayo / William
Friday 12:15	Valdemar / Simon

Week 5: Input and Output

Monday

- Standard in and out
- Pipes and redirection
- Drawing

Thursday

- Audio
- Live programming

Prologue

Quote of the Week

"Computer programming is tremendous fun. Like music, it is a skill that derives from an unknown blend of innate talent and constant practice. Like drawing, it can be shaped to a variety of ends – commercial, artistic, and pure entertainment. Programmers have a well-deserved reputation for working long hours, but are rarely credited with being driven by creative fevers. Programmers talk about software development on weekends, vacations, and over meals not because they lack imagination, but because their imagination reveals worlds that others cannot see."

— Larry O'Brien and Bruce Eckel

Epigram of the Week

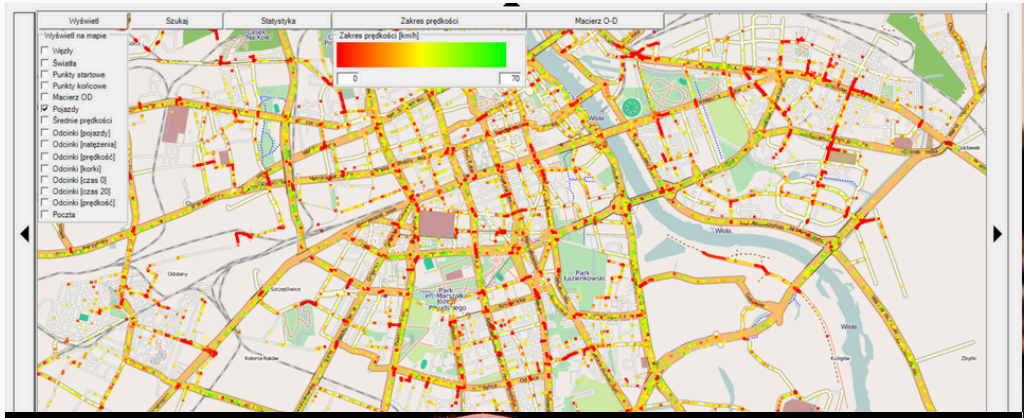
"Once you understand how to write a program get someone else to write it."

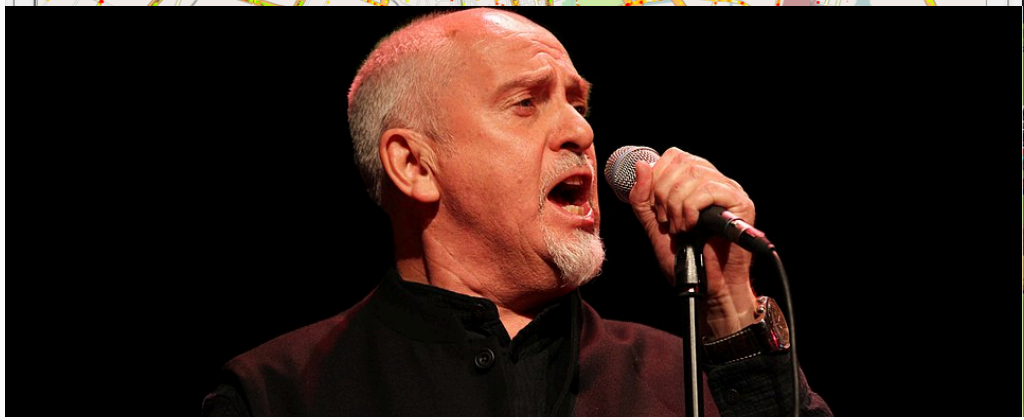
— Alan Perlis

Input and Output

What is the input and output of program?

But input and output can be so much more:





Reminder: Standard Out

We have already seen how to use standard out:

```
public class SayHello {
    public static void main(String[] args) {
        System.out.print("Hello, ");
        System.out.println("user!");
    }
}
```

Reminder: Command Line Arguments

And we have seen how to use command line arguments:

```
public class Average {
    public static void main(String[] args) {
        double x = Double.parseDouble(args[0]);
        double y = Double.parseDouble(args[1]);
        System.out.println((x + y) / 2);
    }
}
```

Book APIs

The book has a large selection of APIs.

StdIn	*read numbers and text from standard input*
StdOut	*write numbers and text to standard output*
StdDraw	*draw geometric shapes in a window*
StdAudio	*create, play, and manipulate sound*
StdAudioStereo	*create, play, and manipulate stereo sound*
StdRandom	*generate random numbers*
StdStats	*compute statistics*
...	
StdPicture	*process one color image*

You can find these here: https://magnus-madsen.github.io/course-intprog/book.html

How to use the book APIs

1. Download the relevant Java file(s) from the course website.
   - https://magnus-madsen.github.io/course-intprog/book.html
2. Put the downloaded file in the same directory as your Java file(s).
3. Write and compile your Java program as usual with IntelliJ IDEA.

How to use the book APIs

1. Download the relevant Java file(s) from the course website.
   - https://magnus-madsen.github.io/course-intprog/book.html
2. Put the downloaded file in the same directory as your Java file(s).
3. Write and compile your Java program as usual with IntelliJ IDEA.

Note: This is *not* how Java libraries are typically distributed.

Why not use Java APIs?

Why not use Java APIs?:

- The Java APIs are too heavy weight for an introductory programming course.
- The Java APIs are old and sometimes badly designed.

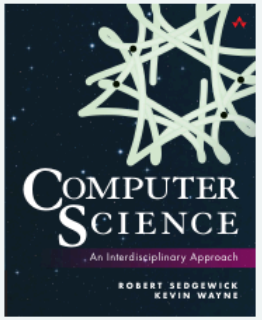
Upshot: **Using the Book APIs is simple and pleasant.**

Upshot: **You have the source code, so you can "peek under the hood."**

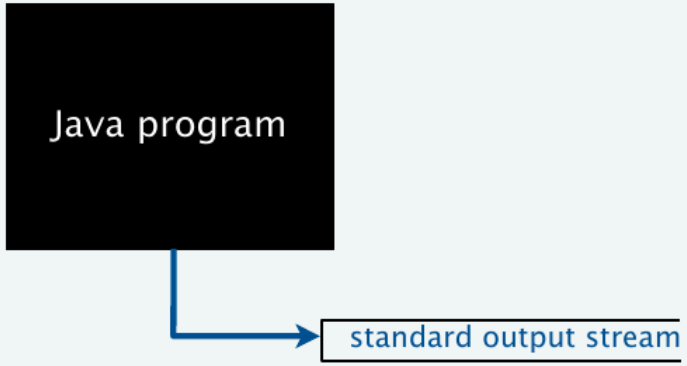
StdOut library

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.



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

The `StdOut.printf` method

`StdOut.printf` is a function for printing *formatted text*. Rather than providing the exact string to print, you provide a **template** and **arguments** to fill the template.

This makes it easy to create text in a regular format.

```
"%w.pc"
```

`w`	field width	integer
`p`	precision	integer
`c`	conversion code	character

There are many *conversion codes*, but the most common are:

`d`	integer
`f`	floating-point
`s`	string

Example: Using `StdOut.printf` (1/2)

We can write `%.2f` which means:

- `f`: floating point number
- `%.2`: two places after the decimal point

```
StdOut.printf("PI is approximately %.2f.\n", Math.PI);
```

```
PI is approximately 3.14.
```

Example: Using `StdOut.printf` (2/2)

We can use multiple patterns in the same call to `printf`.

```
StdOut.printf("The square root of %.1f is %.6f", 2.0, Math.sqrt(2.0));
```

```
The square root of 2.0 is 1.414214
```

Example: Width and Precision in `printf`

We can use field width to print a nicely looking table:

```
StdOut.printf("Name: %8s, Age: %3d, Score: %6.2f\n", "Alice", 25, 87.456);
StdOut.printf("Name: %8s, Age: %3d, Score: %6.2f\n", "Bob", 7, 92.1);
```

```
Name:    Alice, Age:  25, Score:  87.46
Name:      Bob, Age:   7, Score:  92.10
```

Q:

What does this print?

```
System.out.println("PI: %2.1f", Math.PI)
```

Q:

What does this print?

```
StdOut.printf("PI: %4.8f", Math.PI)
```

Example: Zero Padding

We can also use width to zero-pad numbers.

```
StdOut.printf("Date: %04d-%02d-%02d", 1849, 6, 5);
```

```
Date: 1849-06-05
```

Example: Extras

We can print a literal "`%`" by using two "`%`".

We can print a new-line by using "`%n`".

```
StdOut.printf("My favorite symbols are: %%, &, and X!%n%nAnd also $!")
```

```
My favorite symbols are: %, &, and X!

And also $!
```

Q:

What does this print?

```
printf("Result: %5.2f%% complete%n", 67.891);
```

Q:

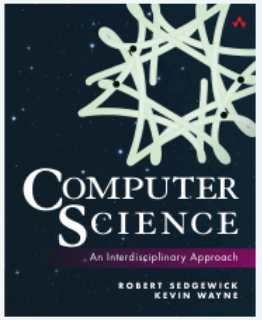
What does this print?

```
printf("%10s|%04d|%8.3f%n", "Java", 42, 3.14159);
```

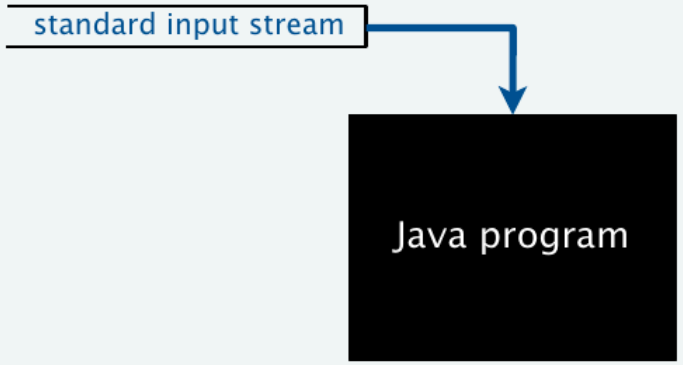
StdIn library

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.



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*



Example: using `StdIn` (1/3)

We can use `StdIn.readString` to read one word at a time.

```
public class Screaming {
    public static void main(String[] args) {
        while (true) {
            String s = StdIn.readString();
            System.out.println(s.toUpperCase());
        }
    }
}
```

```
$ java Screaming
```

```
hello (enter)
HELLO
world (enter)
WORLD
```

Example: using `StdIn` (2/3)

We can use `StdIn.readDouble` to read one double at a time.

```
public class Average {
    public static void main(String[] args) {
        StdOut.print("Enter two nums: ");
        double x = StdIn.readDouble();
        double y = StdIn.readDouble();
        double average = (x + y) / 2.0;
        StdOut.printf("Avg: %.2f%n", average);
    }
}
```

```
$ java Average
```

```
Enter two nums: 3.5 7.8
Avg: 5.65
```

Example: using `StdIn` (3/3)

We have to be careful with our inputs...

```
public class ReadNumbers {
    public static void main(String[] args) {
        StdOut.print("Enter an int: ");
        int number = StdIn.readInt();
        StdOut.println("Int: " + number);
    }
}
```

```
$ java ReadNumbers
```

```
Enter an integer: hello
Exception in thread "main"
java.util.InputMismatch...
```


If you provide the wrong type of input, your program will crash with an exception.

Typing into Standard In

This program reads *all* the integers from standard input and computes their sum.

```
public class SumAll {
    public static void main(String[] args) {
        int[] input = StdIn.readAllInts();
        int sum = 0;
        for (int i = 0; i < input.length; i++) {
            sum += input[i];
        }
        StdOut.println("The total is: " + sum);
    }
}
```

i When reading from standard input, you can signal the **end of input** by pressing:

- **Linux/Mac**: `Ctrl+D`
- **Windows**: `Ctrl+Z`, then `Enter`

This sends an EOF (End of File) signal, telling the program that no more input is coming.

Pipes

Not Your Grandfather's Pipe

Not a Sewage Pipe

Although, the old adage: “Garbage In, Garbage Out” applies.

Redirection and Pipes

Pipes are a powerful **abstraction** that enable **modular programming**.

They let programs read from and write to *standard streams*, without knowing whether they're connected to the terminal, files, or even other programs.

> **i** Pipes are great for composing small, focused programs. However, they can become complex when managing multiple data sources in larger applications.

Like a sink that doesn't care about its water source or drain, programs using pipes focus on their core task.

Everything is a File

Pipes are integral to the *Unix philosophy*, part of which encourages that **everything is a file**.

All of the following are files in a Unix OS that can be used with pipes.

`/home/alice/foo.txt`	regular text file
`/home/alice/Fun.java`	source code
`/home/alice/Fun.class`	Java bytecode
`/usr/bin/java`	executable file
`/home/alice/`	directory
`/dev/lp0`	printer
`/dev/random`	random data
`/dev/sda`	hard disk
`/proc/cpuinfo`	CPU information
`/dev/input/mouse0`	mouse input

Example: Redirect Standard Out

We can use pipes to redirect the output of a Java program to a file:

```
class GenerateNumbers {
    public static void main(String[] args) {
        int n = Integer.parseInt(args[0]);
        for (int i = 1; i <= n; i++) {
            StdOut.println(i);
        }
    }
}
```

```
$ java GenerateNumbers 3 > numbers.txt # redirect output
$ cat numbers.txt                      # print file
```

```
1
2
3
```

Example: Redirect Standard In

We can redirect input from a file to a Java program.

```
class SquareNumbers {
    public static void main(String[] args) {
        while (!StdIn.isEmpty()) {
            int number = StdIn.readInt();
            int square = number * number;
            StdOut.println(square);
        }
    }
}
```

```
$ echo "5 2 5" > numbers.txt       # write numbers into a file
$ java SquareNumbers < numbers.txt # redirect input
```

```
25
4
25
```

Example: Redirect Both Standard In and Standard Out

We can redirect both input and output to work with files:

```
$ echo "5 2 5" > input.txt
$ java SquareNumbers < input.txt > output.txt
$ cat output.txt
```

```
25
4
25
```

Example: Connecting Two Programs via Pipes

We can connect the output of one program to the input of another using pipes:

```
$ java GenerateNumbers 3 | java SquareNumbers
```

```
1
4
9
```

We can add more pipes to make a longer chain:

```
$ java GenerateNumbers 3 | java SquareNumbers | java SquareNumbers
```

```
1
16
81
```

Beyond Simple Pipes

Shell programming with pipes is very powerful.

Here is a sophisticated example:

```
cat data.csv | grep "Plastic" | cut -d, -f2 | sort -u
```

This program is made of four programs:

- `cat`: prints the content of `data.csv`
- `grep`: filters to lines containing `Plastic`
- `cut`: selects one field from each line
- `sort`: sorts the results and eliminates duplicates

Q:

You have a program `FilterEven` that only outputs even numbers from its input. What does this pipeline produce?

```
$ java GenerateNumbers 6
  | java FilterEven
  | java SquareNumbers
```

Q:

What is the difference between these two commands?

```
$ java GenerateNumbers 5 > output.txt | java SquareNumbers
```

vs

```
$ java GenerateNumbers 5 | java SquareNumbers > output.txt
```

Drawing

The `StdDraw` API

We can use the `StdDraw` API to draw figures on the screen!

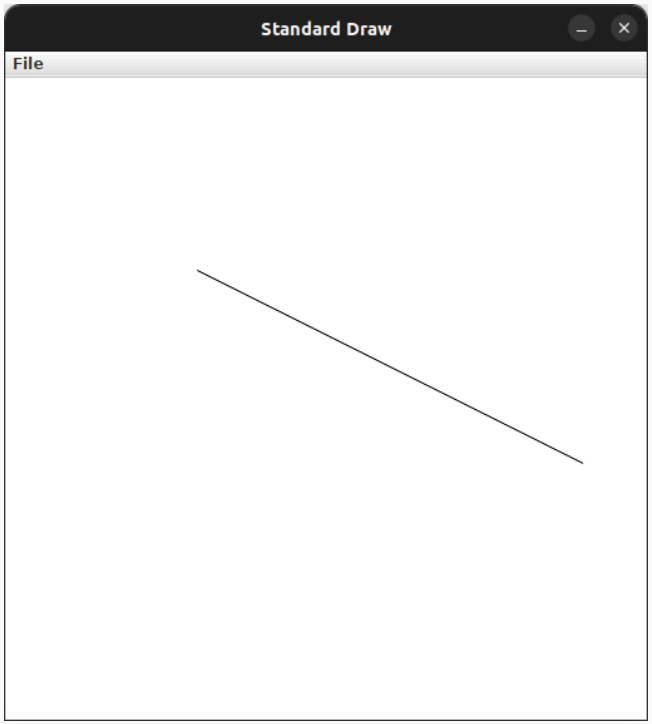
`void line(double x0, double y0, double x1, double y1)`
`void point(double x, double y)`
`void text(double x, double y, String text)`
`void circle(double x, double y, double radius)`
`void square(double x, double y, double halfLength)`
`void polygon(double[] x, double[] y)`
`void picture(double x, double y, String filename)`
`void setPenRadius(double radius)`
`void setPenColor(Color color)`
`...`

Example: Drawing a Line

By default, coordinates range from (0.0, 0.0) to (1.0, 1.0). The origin is bottom, left.

We can use `StdDraw.line` to draw a line:

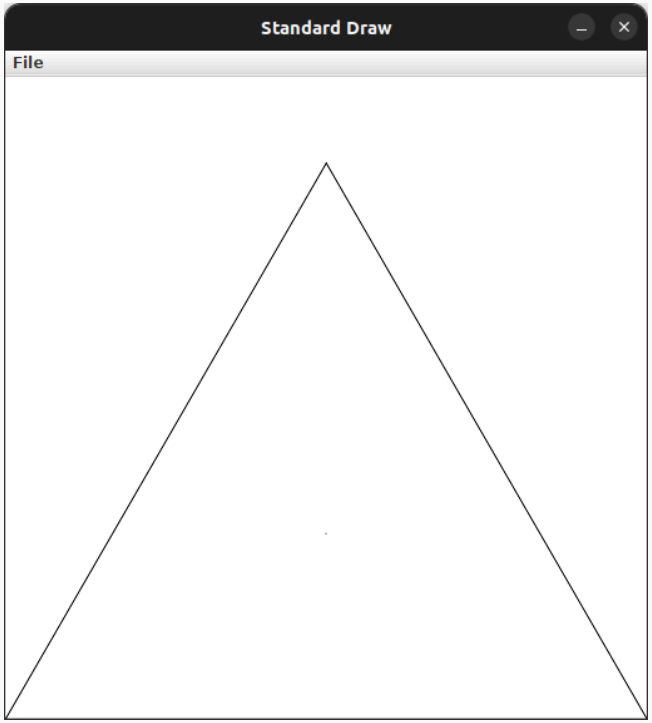
```
public class Line {
    public static void main(String[] args) {
        double x0 = 0.3;
        double y0 = 0.7;
        double x1 = 0.9;
        double y1 = 0.4;
        StdDraw.line(x0, y0, x1, y1);
    }
}
```



Example: Drawing a Triangle

We can connect lines to make shapes.

```
public class Triangle {
    public static void main(String[] args) {
        double t = Math.sqrt(3.0) / 2.0;
        StdDraw.line(0.0, 0.0, 1.0, 0.0);
        StdDraw.line(1.0, 0.0, 0.5, t);
        StdDraw.line(0.5, t, 0.0, 0.0);
    }
}
```



Scaling

The default `StdDraw` canvas:

- is 512 x 512 pixels
- uses the range 0.0–1.0 on the x-axis
- uses the range 0.0–1.0 on the y-axis
- uses a 0.002-radius pen

We can adjust values these with the scaling functions.

For example, we can change the scale from fractions to percentages:

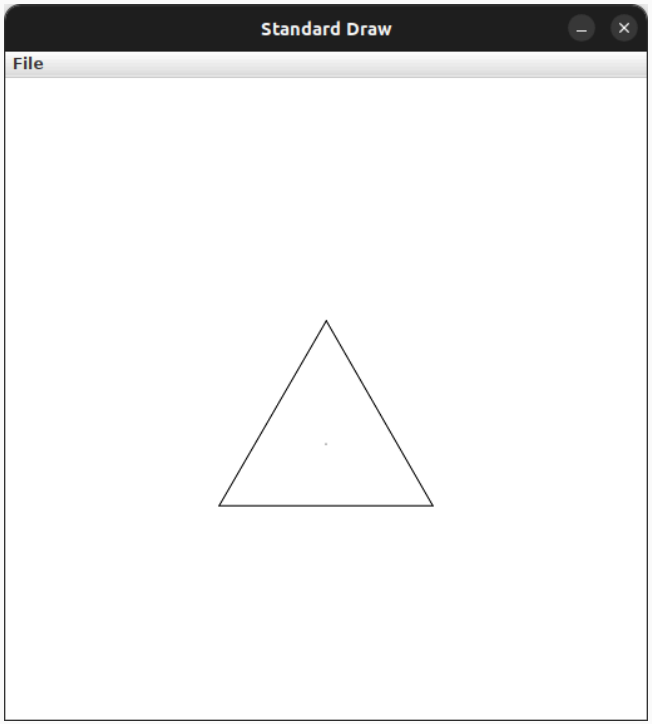
```
StdDraw.setXscale(0.0, 100.0);
StdDraw.setYscale(0.0, 100.0);
```

Scaling makes it easy to resize an image without having to change coordinates individually.

Example: Scaling

We can make the triangle smaller by changing the range from 0.0–1.0 to −1.0–2.0.

```
public class ScaledTriangle {
    public static void main(String[] args) {
        StdDraw.setXscale(-1.0, 2.0);
        StdDraw.setYscale(-1.0, 2.0);
        double t = Math.sqrt(3.0) / 2.0;
        StdDraw.line(0.0, 0.0, 1.0, 0.0);
        StdDraw.line(1.0, 0.0, 0.5, t);
        StdDraw.line(0.5, t, 0.0, 0.0);
        StdDraw.point(0.5, t / 3.0);
    }
}
```



Drawing Shapes

`StdDraw` has functions for drawing filled and unfilled shapes.

Each takes the shape's center coordinates `(x, y)` as arguments.

`void circle(double x, double y, double radius)`
`void rectangle(double x, double y, double halfWidth, double halfHeight)`
`void filledCircle(double x, double y, double radius)`
`void filledRectangle(double x, double y, double halfWidth, double halfHeight)`

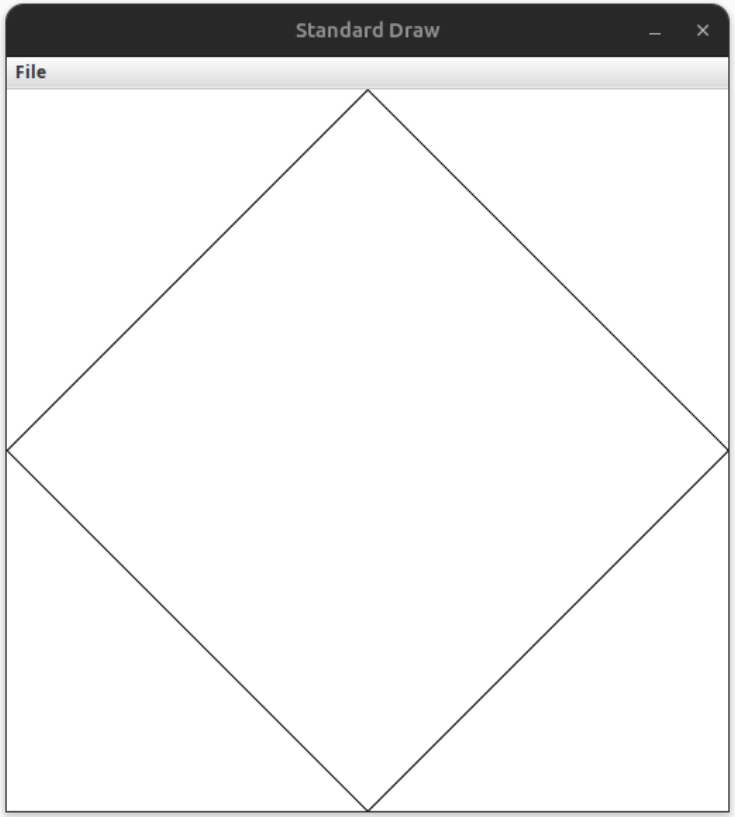
Drawing Polygons

We can also draw polygons by giving a list of x-and y-coordinates for the vertices.

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

For example, to draw a diamond shape, we provide the four corners of the diamond.

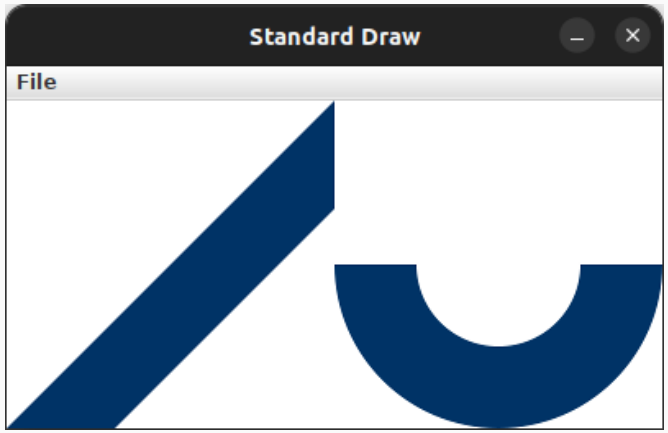
```
double[] xCoords = { 0.0, 0.5, 1.0, 0.5 };
double[] yCoords = { 0.5, 1.0, 0.5, 0.0 };
StdDraw.polygon(xCoords, yCoords);
```



Example: Drawing Shapes

We can combine shapes to make the AU logo.

Here we first draw the “A” as a polygon, then we make the “U” by drawing a circle, and partially covering it with a white circle and rectangle.




```
import java.awt.Color;
public class AarhusLogo {
    public static void main(String[] args) {
        StdDraw.setCanvasSize(400, 200);
        StdDraw.setXscale(0, 200);
        StdDraw.setYscale(0, 100);

        double[] aXCoords = { 0, 100, 100, 33 };
        double[] aYCoords = { 0, 100, 67, 0 };
        StdDraw.filledPolygon(aXCoords, aYCoords);

        StdDraw.filledCircle(150, 50, 50);
        StdDraw.setPenColor(Color.WHITE);
        StdDraw.filledCircle(150, 50, 25);
        StdDraw.filledRectangle(150, 75, 50, 25);
    }
}
```

Q:

What does the following program draw?

```
public class MysteryDraw {
    public static void main(String[] args) {
        for (double r = 0.05; r < 1.0; r = r + 0.05) {
            StdDraw.circle(0.5, 0.5, r);
        }
    }
}
```

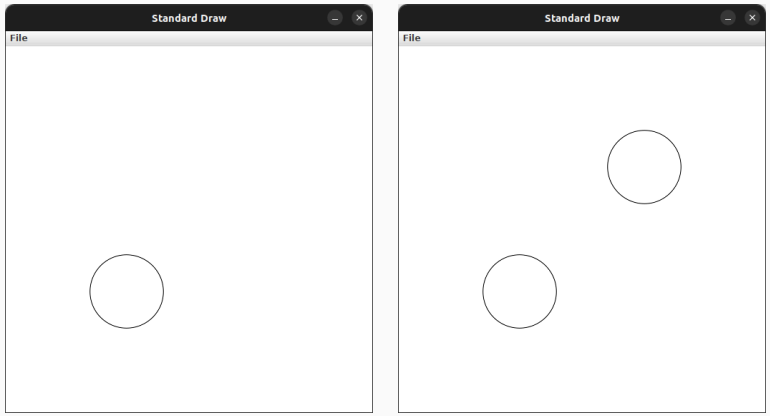
Double buffering (1/2)

Double buffering is a standard graphics rendering technique where graphics are first loaded into a *buffer*, then displayed when ready.

Double buffering ensures that graphics are only displayed when they are ready, so a user does not see an image while it is still being drawn. That is, the image does not "flicker".

```
public class SlowDraw {
    public static void main(String[] args) {
        StdDraw.circle(0.33, 0.33, 0.1);
        StdDraw.pause(3000);
        StdDraw.circle(0.67, 0.67, 0.1);
    }
}
```

Here we simulate a slow drawing by adding a pause to the middle.



Double buffering (2/2)

To enable double buffering, we use `enableDoubleBuffering()` and `show()`:

```
public class DoubleBufferedDraw {
    public static void main(String[] args) {
        StdDraw.enableDoubleBuffering();

        StdDraw.circle(0.33, 0.33, 0.1);
        StdDraw.pause(3000);
        StdDraw.circle(0.67, 0.67, 0.1);

        StdDraw.show(); // Display all at once
    }
}
```

With double buffering enabled, both circles appear simultaneously after the pause.

Animation

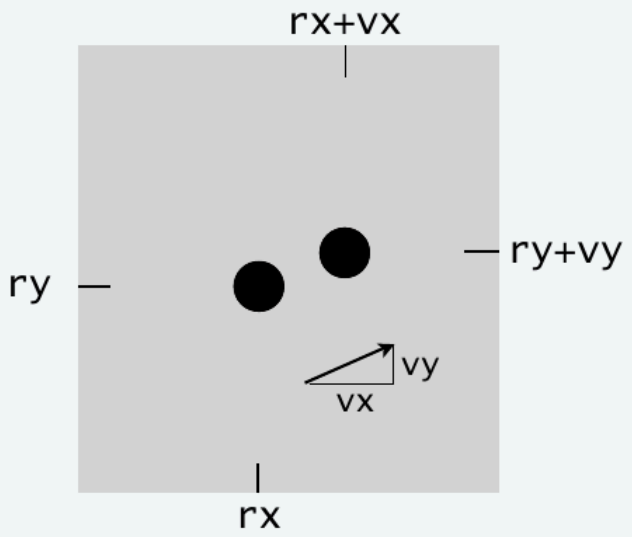
Animation

To create animation with StdDraw.

Repeat the following:

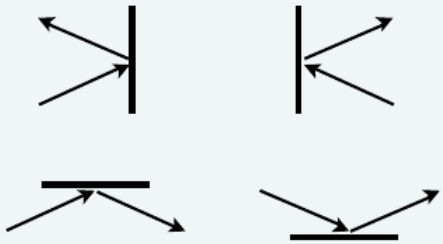
- Clear the screen.
- Move the object.
- Draw the object.
- Display and pause briefly.

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`.



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);
      }
   }
}
```

read coords of bounding box

rescale

read and plot a point

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)

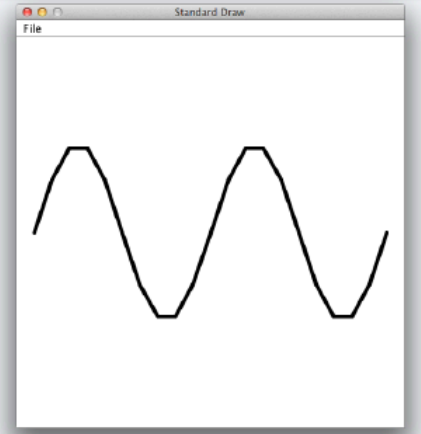
StdDraw application: plotting a function

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

Method. Take *N* samples, regularly spaced.

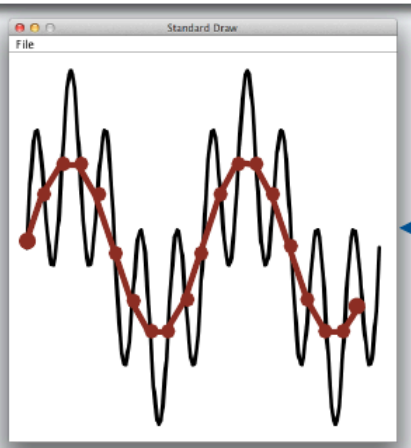
```
public class PlotFunctionEx
{
   public static void main(String[] args)
   {
      int N = Integer.parseInt(args[0]);
      double[] x = new double[N+1];
      double[] y = new double[N+1];
      for (int i = 0; i <= N; i++)
      {
         x[i] = Math.PI * i / N;
         y[i] = Math.sin(4*x[i]) + 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!

Q:

What does the following program draw?

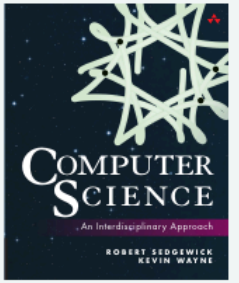
```
public class MysteryDraw {
    public static void main(String[] args) {
        StdDraw.setXscale(5.0, 10.0);
        StdDraw.setYscale(5.0, 10.0);
        StdDraw.circle(10.0, 10.0, 5.0);
    }
}
```

Audio

StdAudio library

Developed for this course, also broadly useful

- Play a sound wave (array of double values) on your computer's audio output.
- Convert to and from standard `.wav` file format.



API

`public class StdAudio`	
`void play(String file)`	*play the given* `.wav` *file*
`void play(double[] a)`	*play the given sound wave*
`void play(double x)`	*play the sample for 1/44100 second*
`void save(String file, double[] a)`	*save to a* `.wav` *file*
`double[] read(String file)`	*read from a* `.wav` *file*

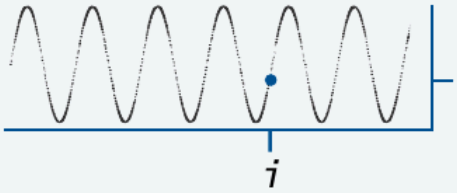
Enables you to *hear the results* of your programs that manipulate sound.

"Hello, World" for StdAudio

```
public class PlayThatNote
{
   public static double[] tone(double hz, double duration)
   {
      int N = (int) (44100 * duration);
      double[] a = new double[N+1];
      for (int i = 0; i <= N; i++)
         a[i] = Math.sin(2 * Math.PI * i * hz / 44100);
     return a;
   }

   public static void main(String[] args)
   {
      double hz = Double.parseDouble(args[0]);
      double duration = Double.parseDouble(args[1]);
      double[] a = tone(hz, duration);
      StdAudio.play(a);
   }
}
```

pitch




```
% java PlayThatNote 440.0 3.0
```

```
% java PlayThatNote 880.0 3.0
```

```
% java PlayThatNote 220.0 3.0
```

```
% java PlayThatNote 494.0 3.0
```

Compiler Error of the Week (1/2)

What is wrong with this code?

```
public class DrawCircle {
    public static void main(String[] args) {
        StdDraw.circle(0.5, 0.5, 0.3, 0.2);
    }
}
```

Compiler Error of the Week (1/2)

What is wrong with this code?

```
public class DrawCircle {
    public static void main(String[] args) {
        StdDraw.circle(0.5, 0.5, 0.3, 0.2);
    }
}
```

```
java: method circle in class StdDraw cannot be applied to given types;
  required: double,double,double
  found:    double,double,double,double
  reason: actual and formal argument lists differ in length
```

The `StdDraw.circle` method only takes **three** parameters: `(x, y, radius)`, not four.

Compiler Error of the Week (2/2)

What is wrong with this code?

```
public class ReadChar {
    public static void main(String[] args) {
        String c = StdIn.readChar();
        StdOut.printf(c);
    }
}
```

Compiler Error of the Week (2/2)

What is wrong with this code?

```
public class ReadChar {
    public static void main(String[] args) {
        String c = StdIn.readChar();
        StdOut.printf(c);
    }
}
```

```
java: incompatible types: char cannot be converted to java.lang.String
```

Live Programming

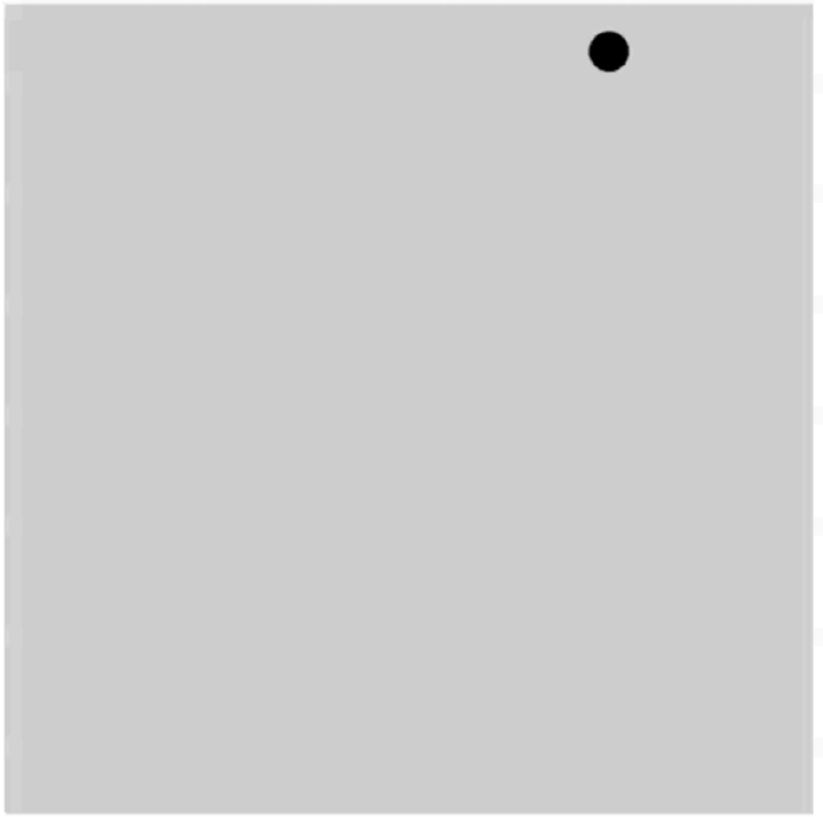
Live Programming

- Bouncing Ball
  - with directional arrow
  - with color change
  - with color trail
  - with size change
  - with acceleration on impact
  - with deformity
  - with two balls

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
```



Sources for Images and Slides

- https://introcs.cs.princeton.edu/java/lectures/
- https://commons.wikimedia.org/wiki/File:The_Women%27s_Auxiliary_Air_Force,_1939-1945._CH214.jpg
- https://commons.wikimedia.org/wiki/File:Graphical_User_Interface_of_the_Traffic_Simulation_Framework.png
- https://commons.wikimedia.org/wiki/File:Goodgame_Empire_Screenshot.jpg
- https://commons.wikimedia.org/wiki/File:Razer_OSVR_Open-Source_Virtual_Reality_for_Gaming_(16863422875).jpg
- https://commons.wikimedia.org/wiki/File:Peter-Gabriel-2011I2.jpg
- https://commons.wikimedia.org/wiki/File:%D0%92%D0%BE%D0%B4%D0%BE%D0%BF%D0%B0%D0%B4_%D0%9B%D0%B0%D1%81%D1%82%D0%BE%D0%B2%D0%BA%D0%B8.jpg
- https://commons.wikimedia.org/wiki/File:20190410_230530_Cat_Lissy_in_Bathroom_Sink_anagoria.jpg