COS 126

General Computer Science

Fall 2010

Robert Sedgewick

Overview

What is COS 126? Broad, but technical, introduction to computer science.

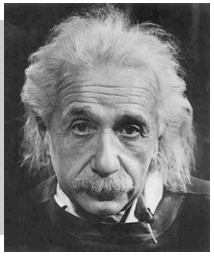
Goals.

- Demystify computer systems.
- Empower you to exploit available technology.
- Build awareness of substantial intellectual underpinnings.

Topics.

- Programming in Java.
- Machine architecture.
- Theory of computation.
- Applications to science, engineering, and commercial computing.

> *" Computers are incredibly fast, accurate, and stupid; humans are incredibly slow, inaccurate, and brilliant; together they are powerful beyond imagination. "* – *Albert Einstein*



The Basics

Lectures. [Sedgewick]

RS office hours. ← everyone needs to meet me!

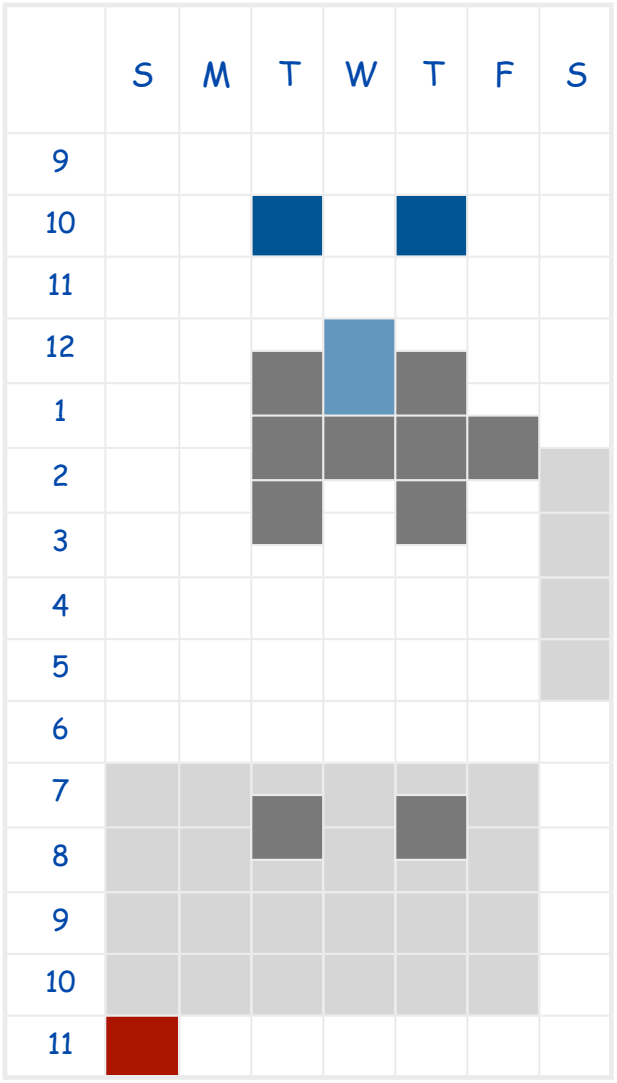
Precepts. [Gabai · Ginsburg · Vertanen · Lee · Lee · Steiglitz · Jones · Stewart · Zhu]

- Tips on assignments / worked examples
- Questions on lecture material.
- Informal and interactive.

Friend 016/017 lab. [Ugrad assistants]

- Help with systems/debugging.
- No help with course material.

Reading period. No lectures; precept 1/4



See **www.princeton.edu/~cos126**
for full details and preceptor office hours.

Grades

Course grades. No preset curve or quota.

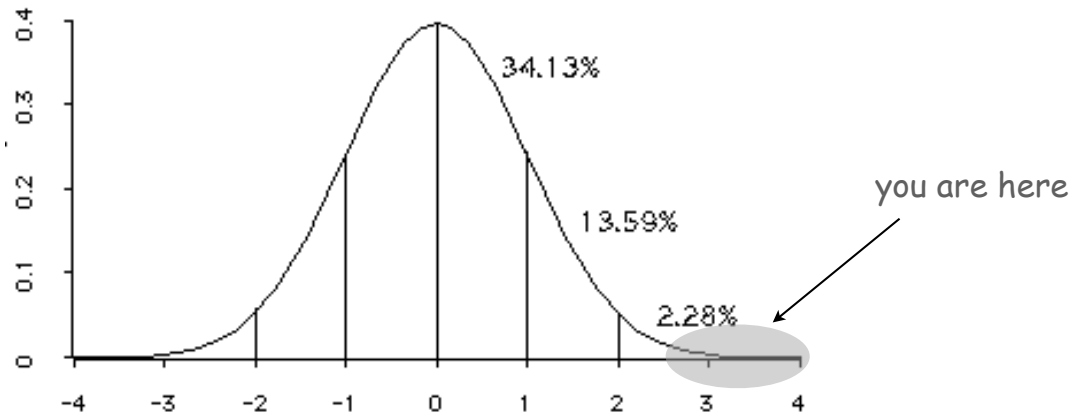
9 programming assignments. 40%.

2 exams (in class, 10/21-22 and 12/16-17). 50%.

Final programming project (due Dean's date - 1). 10%.

Extra credit / staff discretion. Adjust borderline cases.

participation helps, frequent absence hurts



Due dates

	Su	Mo	Tu	We	Th	Fr	Sa
SEP				1	2	3	4
	5	6	7	8	9	10	11
	12	13	14	15	16	17	18
	19	20	21	22	23	24	25
	26	27	28	29	30		
OCT						1	2
	3	4	5	6	7	8	9
	10	11	12	13	14	15	16
	17	18	19	20	21	22	23
	24	25	26	27	28	29	30
	31						
NOV		1	2	3	4	5	6
	7	8	9	10	11	12	13
	14	15	16	17	18	19	20
	21	22	23	24	25	26	27
	28	29	30				
DEC				1	2	3	4
	5	6	7	8	9	10	11
	12	13	14	15	16	17	18
	19	20	21	22	23	24	25
	26	27	28	29	30	31	
JAN							1
	2	3	4	5	6	7	8
	9	10	11	12	13	14	15
	16	17	18	19	20	21	22
	23	24	25	26	27	28	29
	30	31					

Course Materials

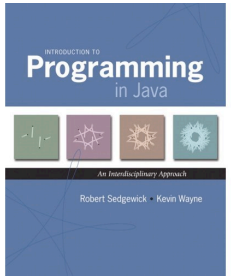
Course website. [www.princeton.edu/~cos126]

- Submit assignments, check grades.
- Programming assignments.
- Lecture slides.

print before lecture;
annotate during lecture

skim before lecture;
read thoroughly afterwards

Required readings. Sedgewick and Wayne. Intro to Programming in Java: An Interdisciplinary Approach.

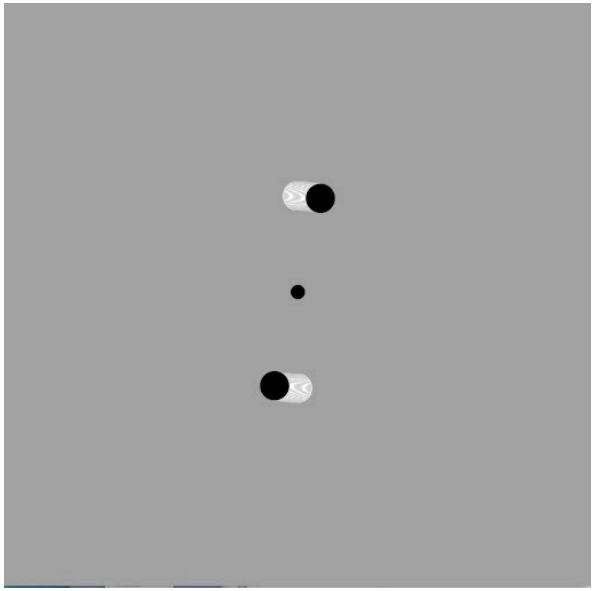


Recommended readings. Harel. What computers can't do.

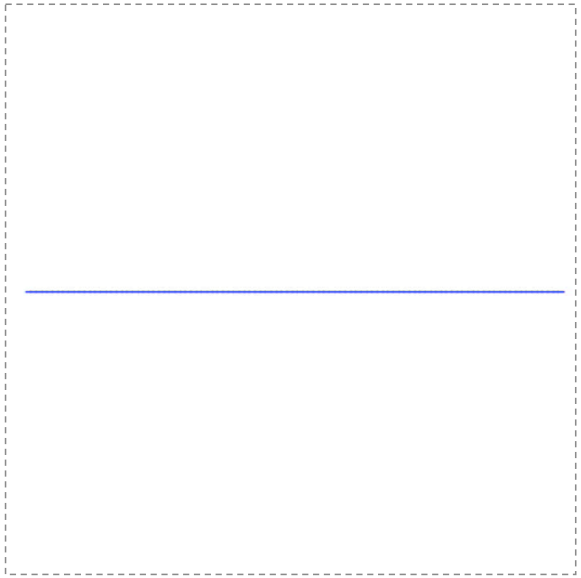
Programming Assignments

Desiderata.

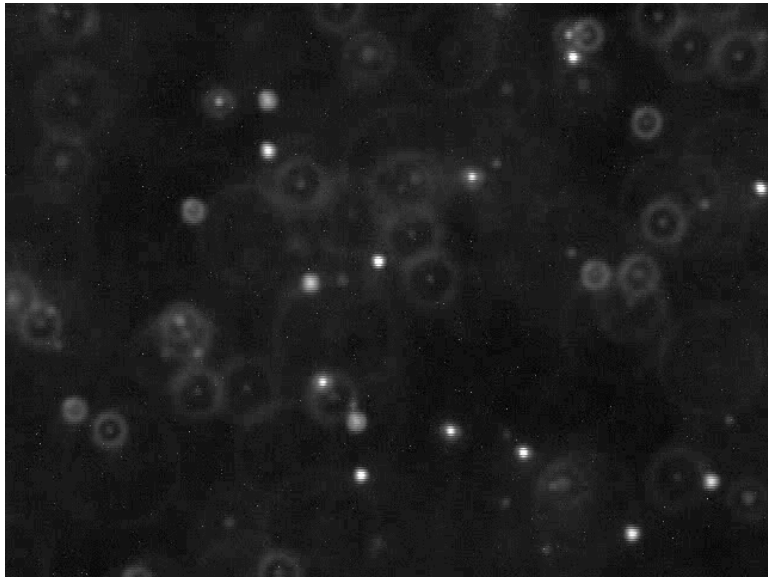
- Address an important scientific or commercial problem.
- Illustrate the importance of a fundamental CS concept.
- You solve problem from scratch!



N-body simulation



pluck a guitar string



estimate Avogadro's number

Programming Assignments

Desiderata.

- Address an important scientific or commercial problem.
- Illustrate the importance of a fundamental CS concept.
- You solve problem from scratch!

Due. Mondays 11pm via Web submission.

Computing equipment.

- Your laptop. [OS X, Windows, Linux, iPhone, …]
- OIT desktop. [Friend 016 and 017 labs]

What's Ahead?

Lecture 2. Intro to Java.

Precept 1. Meets today/tomorrow.

Not registered? Go to any precept now; officially register ASAP.

Change precepts? Use SCORE.

see Colleen Kenny-McGinley in CS 210
if the only precept you can attend is closed

Assignment 0. [www.princeton.edu/~cos126/assignments.php]

- Due Monday 11PM.
- Read Sections 1.1 and 1.2 in textbook.
- Install Java programming environment + a few exercises.
- Lots of help available, don't be bashful.

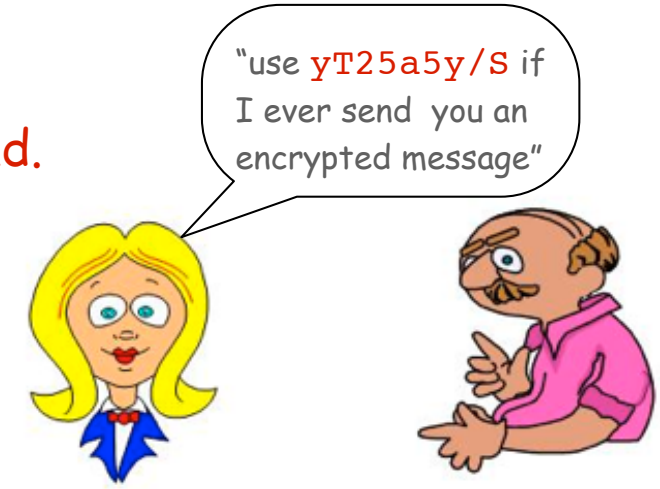
END OF ADMINISTRATIVE STUFF

0. Prologue: A Simple Machine

Secure Chat with a One-Time Pad

Alice wants to send a secret message to Bob

- Sometime in the past, they exchange a one-time pad.
- Alice uses the pad to encrypt the message.



Secure Chat 1.0 [alice]

```
[alice]: Hey, Bob
  [bob]: Hi, Alice!
[alice]: SENDMONEY
```

Encrypt SENDMONEY with yT25a5y/S

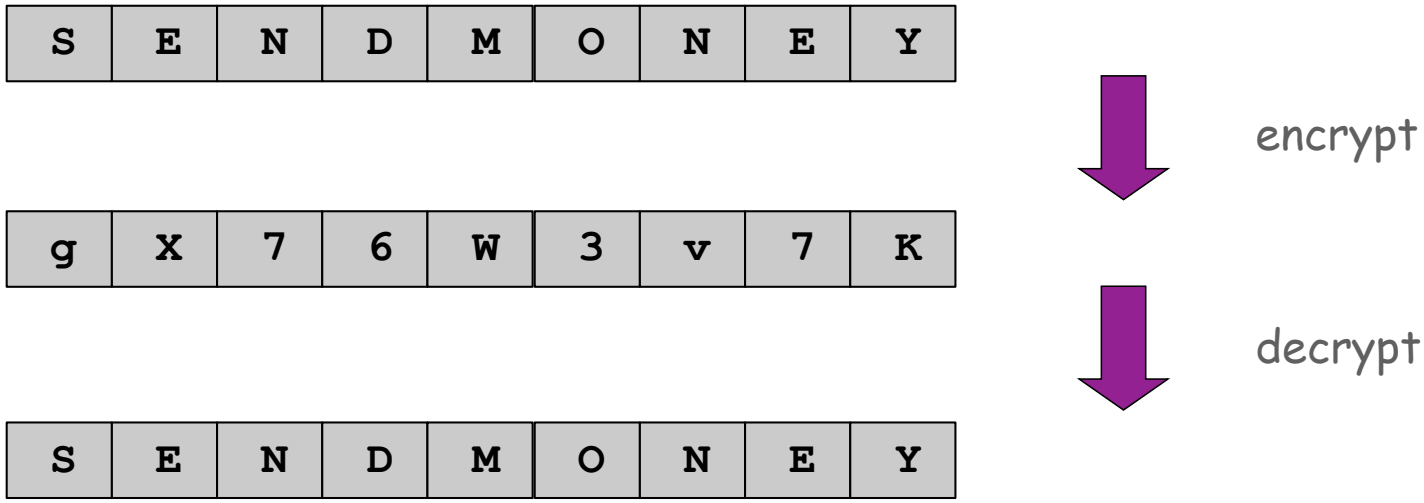
Secure Chat 1.0 [bob]

```
[alice]: Hey, Bob
  [bob]: Hi, Alice!
[alice]: gX76W3v7K
```

Key point: Without the pad, Eve cannot understand the message.

Encryption Machine

Goal. Design a machine to encrypt and decrypt data.



Encryption Machine

Goal. Design a machine to encrypt and decrypt data.

S	E	N	D	M	O	N	E	Y

encrypt

g	X	7	6	W	3	v	7	K

decrypt

S	E	N	D	M	O	N	E	Y

Enigma encryption machine.

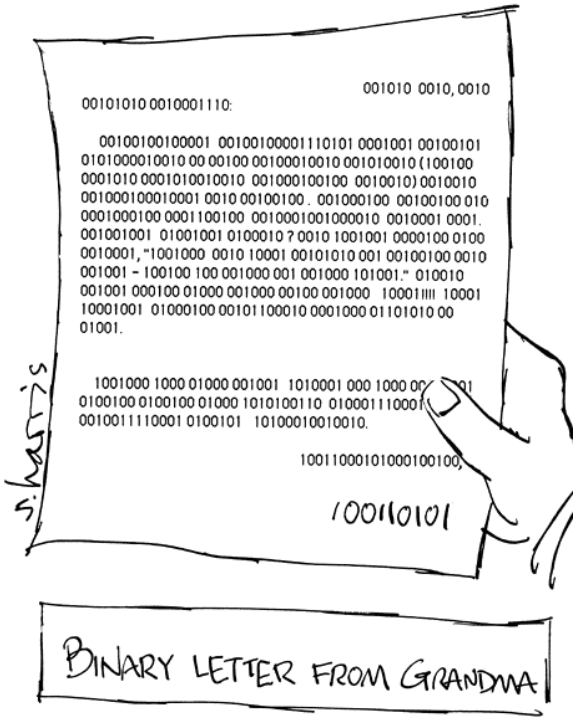
- "Unbreakable" German code during WWII.
- Broken by Turing bombe.
- One of first uses of computers.
- Helped win Battle of Atlantic by locating U-boats.

A Digital World

Data is a sequence of bits. [bit = 0 or 1]

- Text.
- Programs, executables.
- Documents, pictures, sounds, movies, …

thousands of bits



Copyright 2004, Sidney Harris
http://www.sciencecartoonsplus.com

billions of bits

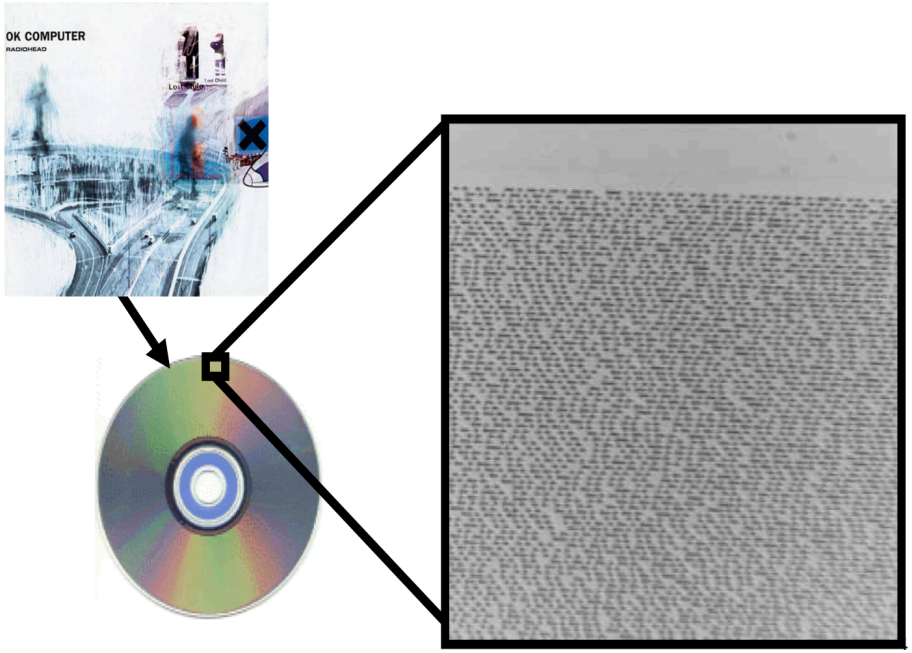


image courtesy of David August

A Digital World

Data is a sequence of bits. [bit = 0 or 1]

- Text.
- Programs, executables.
- Documents, pictures, sounds, movies, ...

Ex. Base64 encoding of text.

- Simple method for representing `A-Z, a-z, 0-9, +, /`
- 6 bits to represent each symbol (64 symbols)

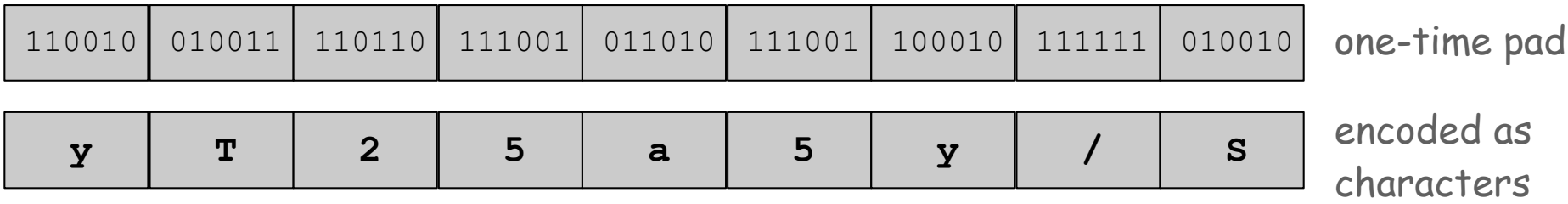
000000 A	001000 I	010000 Q	011000 Y	100000 g	101000 o	110000 w	111000 4
000001 B	001001 J	010001 R	011001 Z	100001 h	101001 p	110001 x	111001 5
000010 C	001010 K	010010 S	011010 a	100010 i	101010 q	110010 y	111010 6
000011 D	001011 L	010011 T	011011 b	100011 j	101011 r	110011 z	111011 7
000100 E	001100 M	010100 U	011100 c	100100 k	101100 s	110100 0	111100 8
000101 F	001101 N	010101 V	011101 d	100101 l	101101 t	110101 1	111101 9
000110 G	001110 O	010110 W	011110 e	100110 m	101110 u	110110 2	111110 +
000111 H	001111 P	010111 X	011111 f	100111 n	101111 v	110111 3	111111 /

Secure Chat with a One-Time Pad

First challenge: Create a one-time pad.

Good choice: A random sequence of bits (stay tuned).

Note: any sequence of bits can be encoded as characters



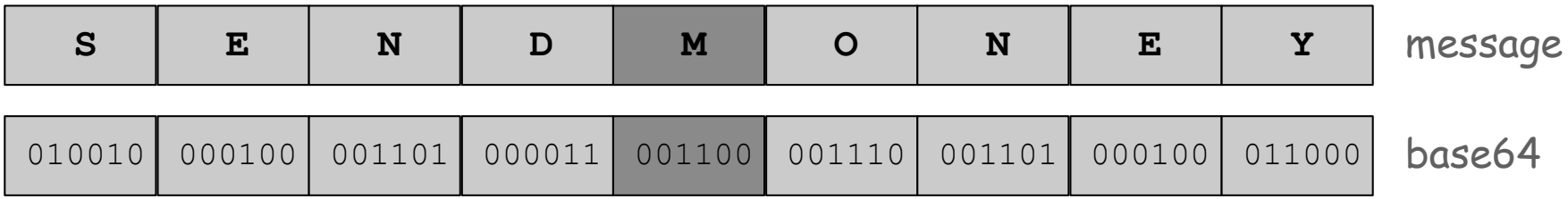
One-Time Pad Encryption

Encryption.

- Convert text message to N bits.

Base64 Encoding

char	dec	binary
A	0	000000
B	1	000001
...	...	...
M	12	001100
...	...	...



One-Time Pad Encryption

Encryption.

- Convert text message to N bits.
- Use N random bits as one-time pad.

S	E	N	D	M	O	N	E	Y	message
010010	000100	001101	000011	001100	001110	001101	000100	011000	base64
110010	010011	110110	111001	011010	111001	100010	111111	010010	one-time pad
y	T	2	5	a	5	y	/	S	

One-Time Pad Encryption

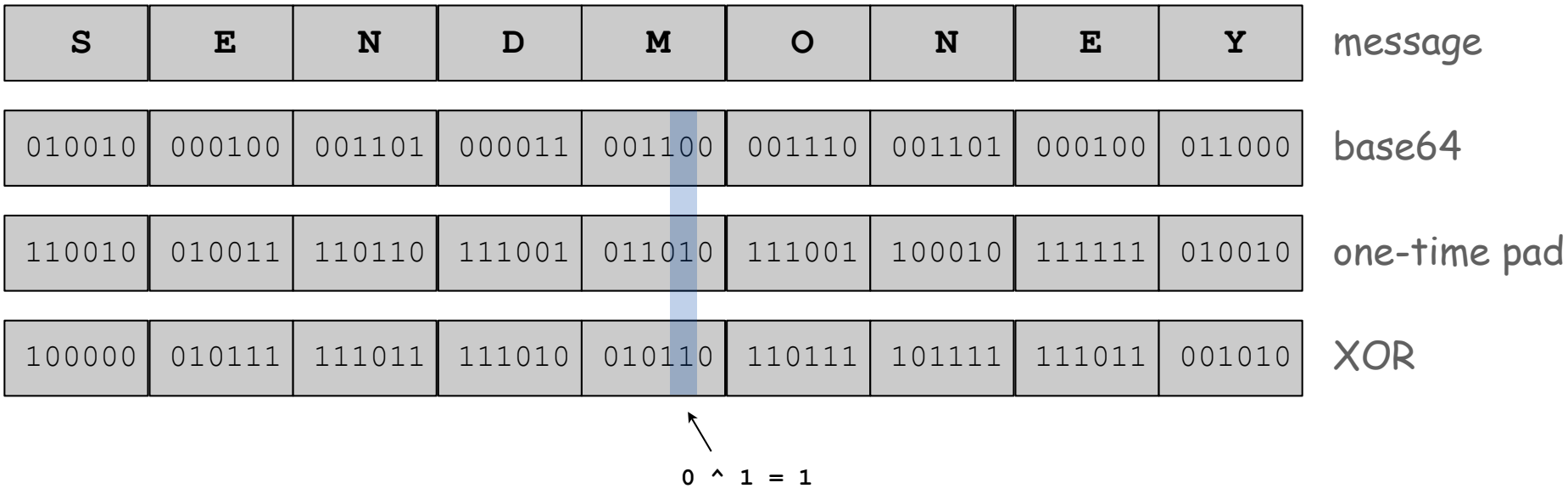
Encryption.

- Convert text message to N bits.
- Use N random bits as one-time pad.
- Take bitwise XOR of two bitstrings.

sum corresponding pair of bits: 1 if sum is odd, 0 if even

XOR Truth Table

x	y	x ^ y
0	0	0
0	1	1
1	0	1
1	1	0



One-Time Pad Encryption

Base64 Encoding

char	dec	binary
A	0	000000
B	1	000001
…	…	…
w	22	010110
…	…	…

Encryption.

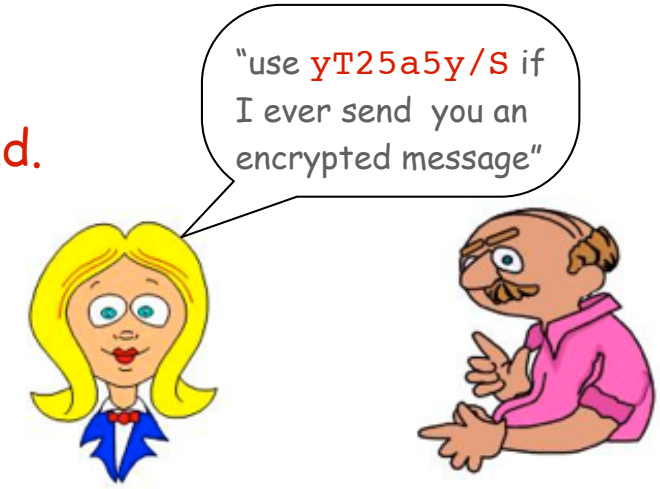
- Convert text message to N bits.
- Use N random bits as one-time pad.
- Take bitwise XOR of two bitstrings.
- Convert binary back into text.

S	**E**	**N**	**D**	**M**	**O**	**N**	**E**	**Y**	message
010010	000100	001101	000011	001100	001110	001101	000100	011000	base64
110010	010011	110110	111001	011010	111001	100010	111111	010010	one-time pad
100000	010111	111011	111010	010110	110111	101111	111011	001010	XOR
g	**X**	**7**	**6**	**W**	**3**	**v**	**7**	**K**	encrypted

Secure Chat with a One-Time Pad

Alice wants to send a secret message to Bob

- Sometime in the past, they exchange a one-time pad.
- Alice uses the pad to encrypt the message.



Secure Chat 1.0 [alice]

```
[alice]: Hey, Bob
  [bob]: Hi, Alice!
[alice]: SENDMONEY
```

Encrypt SENDMONEY with yT25a5y/S

Secure Chat 1.0 [bob]

```
[alice]: Hey, Bob
  [bob]: Hi, Alice!
[alice]: gX76W3v7K
```

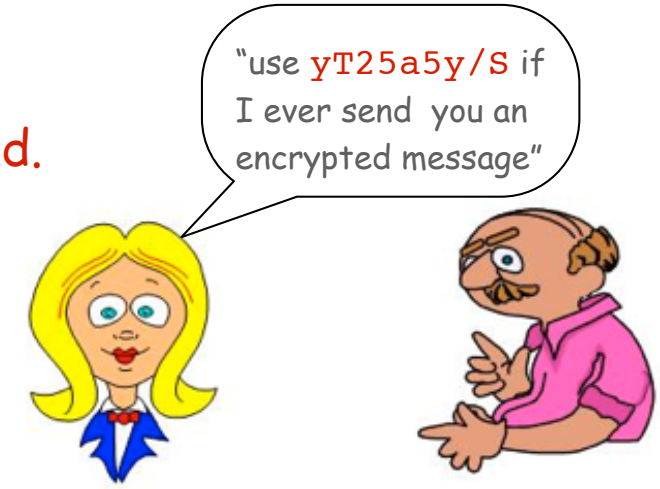
Key point: Without the pad, Eve cannot understand the message.

But how can Bob understand the message?

Secure Chat with a One-Time Pad

Alice wants to send a secret message to Bob

- Sometime in the past, they exchange a one-time pad.
- Alice uses the pad to encrypt the message.
- Bob uses the same pad to decrypt the message.



Secure Chat 1.0 [alice]

```
[alice]: Hey, Bob
  [bob]: Hi, Alice!
[alice]: SENDMONEY
```

Encrypt SENDMONEY with yT25a5y/S

Secure Chat 1.0 [bob]

```
[alice]: Hey, Bob
  [bob]: Hi, Alice!
[alice]: gX76W3v7K
         SENDMONEY
```

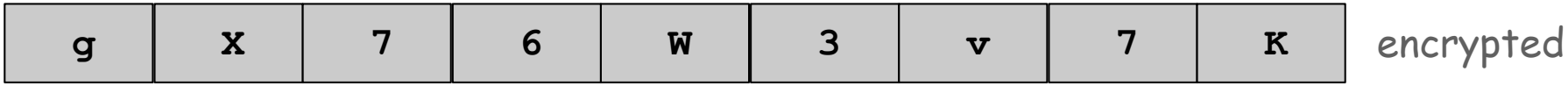
Decrypt with yT25a5y/S

Key point: Without the pad, Eve cannot understand the message.

One-Time Pad Decryption

Decryption.

- Convert encrypted message to binary.



One-Time Pad Decryption

Base64 Encoding

char	dec	binary
A	0	000000
B	1	000001
…	…	…
W	22	010110
…	…	…

Decryption.

- Convert encrypted message to binary.

g	X	7	6	W	3	v	7	K	encrypted
100000	010111	111011	111010	010110	110111	101111	111011	001010	base64

One-Time Pad Decryption

Decryption.

- Convert encrypted message to binary.
- Use same N "random" bits (one-time pad).

g	**X**	**7**	**6**	**W**	**3**	**v**	**7**	**K**	encrypted
100000	010111	111011	111010	010110	110111	101111	111011	001010	base64
110010	010011	110110	111001	011010	111001	100010	111111	010010	one-time pad
y	**T**	**2**	**5**	**a**	**5**	**y**	**/**	**S**	

One-Time Pad Decryption

Decryption.

- Convert encrypted message to binary.
- Use same N random bits (one-time pad).
- Take bitwise XOR of two bitstrings.

XOR Truth Table

x	y	x ^ y
0	0	0
0	1	1
1	0	1
1	1	0

g	X	7	6	W	3	v	7	K	
100000	010111	111011	111010	010110	110111	101111	111011	001010	encrypted
110010	010011	110110	111001	011010	111001	100010	111111	010010	base64
010010	000100	001101	000011	001100	001110	001101	000100	011000	one-time pad
									XOR

1 ^ 1 = 0

One-Time Pad Decryption

Decryption.

- Convert encrypted message to binary.
- Use same N random bits (one-time pad).
- Take bitwise XOR of two bitstrings.
- Convert back into text.

Base64 Encoding

char	dec	binary
A	0	000000
B	1	000001
…	…	…
M	12	001100
…	…	…

g	**X**	**7**	**6**	**W**	**3**	**v**	**7**	**K**	encrypted
100000	010111	111011	111010	010110	110111	101111	111011	001010	base64
110010	010011	110110	111001	011010	111001	100010	111111	010010	one-time pad
010010	000100	001101	000011	001100	001110	001101	000100	011000	XOR
S	**E**	**N**	**D**	**M**	**O**	**N**	**E**	**Y**	message

Why Does It Work?

Crucial property. Decrypted message = original message.

Notation	Meaning
a	original message bit
b	one-time pad bit
^	XOR operator
a ^ b	encrypted message bit
(a ^ b) ^ b	decrypted message bit

XOR Truth Table

x	y	x ^ y
0	0	0
0	1	1
1	0	1
1	1	0

Why is crucial property true?

- Use properties of XOR.
- (a ^ b) ^ b = a ^ (b ^ b) = a ^ 0 = a

associativity of ^

always 0

identity

One-Time Pad Decryption (with the wrong pad)

Decryption.

- Convert encrypted message to binary.

g	X	7	6	W	3	v	7	K

encrypted

One-Time Pad Decryption (with the wrong pad)

Decryption.

- Convert encrypted message to binary.

g	X	7	6	W	3	v	7	K	encrypted
100000	010111	111011	111010	010110	110111	101111	111011	001010	base64

One-Time Pad Decryption (with the wrong pad)

Decryption.

- Convert encrypted message to binary.
- Use wrong N bits (bogus one-time pad).

g	X	7	6	W	3	v	7	K	encrypted
100000	010111	111011	111010	010110	110111	101111	111011	001010	base64
101000	011100	110101	101111	010010	111001	100101	101010	001010	wrong bits

One-Time Pad Decryption (with the wrong pad)

Decryption.

- Convert encrypted message to binary.
- Use wrong N bits (bogus one-time pad).
- Take bitwise XOR of two bitstrings.

g	**X**	**7**	**6**	**W**	**3**	**v**	**7**	**K**	encrypted
100000	010111	111011	111010	010110	110111	101111	111011	001010	base64
101000	011100	110101	101111	010010	111001	100101	101010	001010	wrong bits
001000	001011	001110	010101	000100	001110	001010	010001	000000	XOR

One-Time Pad Decryption (with the wrong pad)

Decryption.

- Convert encrypted message to binary.
- Use wrong N bits (bogus one-time pad).
- Take bitwise XOR of two bitstrings.
- Convert back into text: Oops.

g	**X**	**7**	**6**	**W**	**3**	**v**	**7**	**K**	encrypted
100000	010111	111011	111010	010110	110111	101111	111011	001010	base64
101010	110000	000011	100000	011011	000011	101110	011010	101111	wrong bits
001010	100111	111000	011010	001101	110100	000001	100001	100101	XOR
K	**n**	**4**	**a**	**N**	**0**	**B**	**h**	**l**	wrong message [usually gibberish]

Eve's problem

Key point: Without the pad, Eve cannot understand the message.

But Eve has a computer. Why not try all possible pads?

There are too many possibilities!

- 54 bits implies 2^{54} possibilities.
- If Eve could check 1 million per second. it would take 571 years to try them all!
- If we add another bit, it would take her another 571 years.

1000 bits: 2^{1000} possibilities.

Age of the universe in microseconds: 2^{70}

AAAAAAAAA	yT25a5y/S
AAAAAAAAB	yT25a5y/T
AAAAAAAAC	yT25a5y/Q
...	
qwDgbDuav	Kn4aN0Bhl
...	
yT25a5y/S	SENDMONEY
...	
////////+	NsJGlGNAs
/////////	NsJGlGNAt

Exponential growth dwarfs technological improvements.

Goods and Bads of One-Time Pads

Good.

- Easily computed by hand.
- Very simple encryption/decryption processes.
- Provably unbreakable if bits are truly random. [Shannon, 1940s]

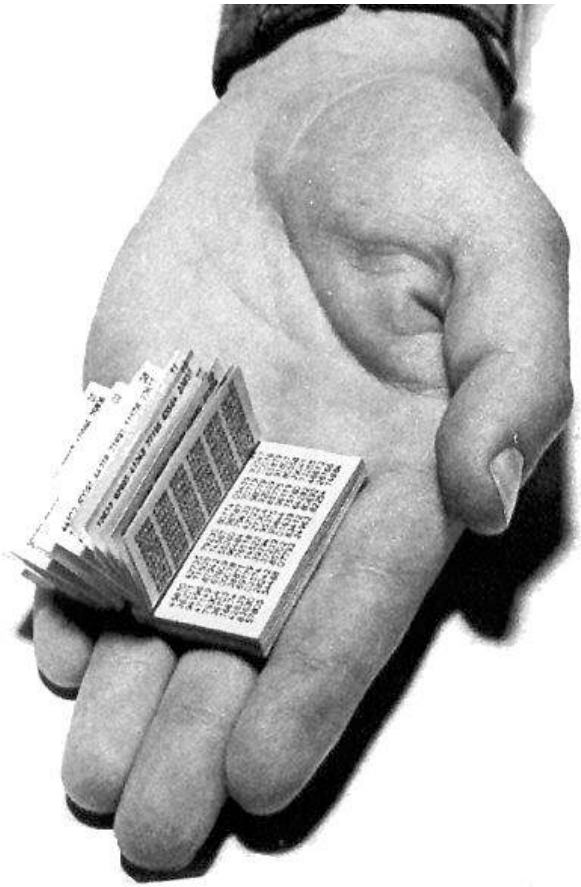
eavesdropper Eve sees only random bits

"one time" means one time only

Bad.

- Easily breakable if pad is re-used.
- Pad must be as long as the message.
- Truly random bits are very hard to come by.
- Pad must be distributed securely.

impractical for Web commerce



a Russian one-time pad

Pseudo-Random Bit Generator

Practical middle-ground.

- Make a "random" bit generator gadget.
- Alice and Bob each get identical small gadgets

  [same gadget works for both]
- Alice and Bob also each get identical books of small seeds.

instead of identical large one-time pads

Goal. Small gadget that produces a long sequence of bits.

Pseudo-Random Bit Generator

Goal: Small gadget that produces a long sequence of pseudo-random bits.

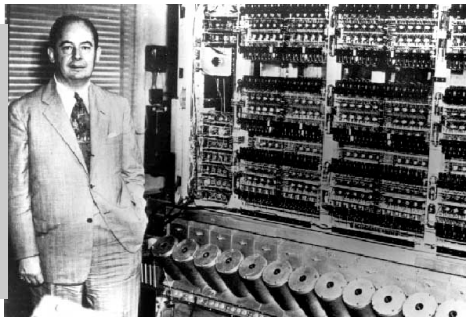
- Enigma
- Linear feedback shift register.
- Linear congruential generator.
- Blum-Blum-Shub generator.
- [many others have been invented]

Pseudo-random? Bits are not really random:

- Bob's and Alice's gadgets must produce the same bits from the same seed.
- Bits must have as many properties of random bits as possible (to foil Eve).

Ex 1. approx 1/2 0s and 1/2 1s
Ex 2. approx 1/4 each of 00, 01, 10 11

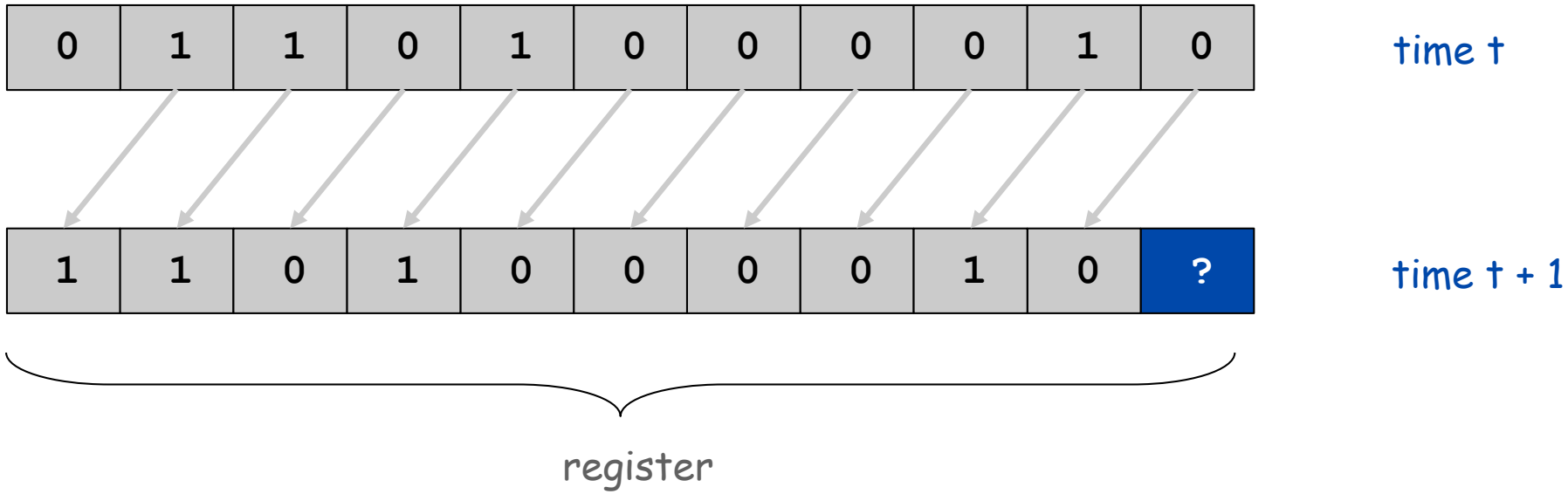
" Anyone who considers arithmetical methods of producing random digits is, of course, in a state of sin. "
– *Jon von Neumann (left)*
– *ENIAC (right)*



Shift Register

Shift register terminology.

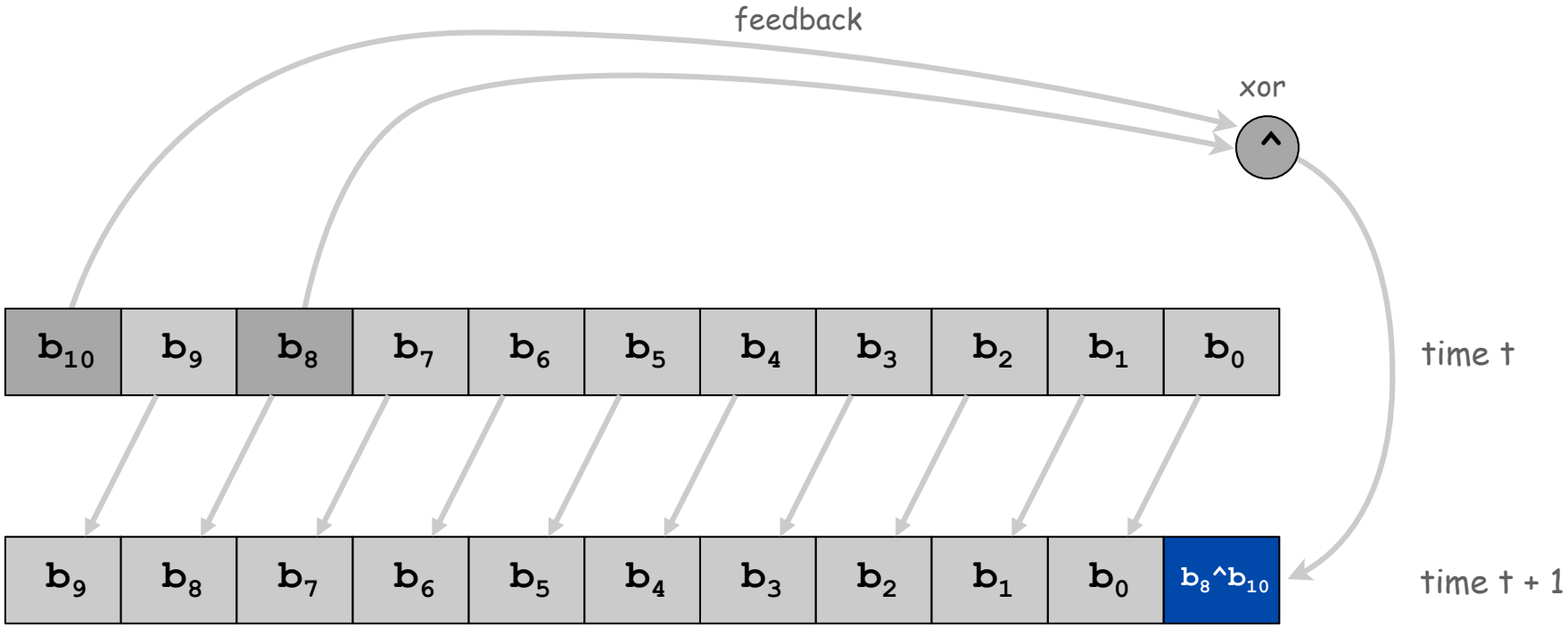
- Bit: 0 or 1.
- Cell: storage element that holds one bit.
- Register: sequence of cells.
- Seed: initial sequence of bits.
- Shift register: when clock ticks, bits propagate one position to left.



Linear Feedback Shift Register (LFSR)

{8, 10} linear feedback shift register.

- Shift register with 11 cells.
- Bit b_0 is is XOR of previous bits b_8 and b_{10}.
- Pseudo-random bit = b_0.



Random Numbers

Q. Are these 2000 numbers random? If not, what is the pattern?

```
11001001001111011011100101101011100110001011111101001000010011010010111100110010011111111011100
00010101100010000111010100110100001111001001100111011111110101000001000010001010010100011000
00101111000100100110101101111000110100110111001111010111100100010011101010111010000010100100001
00011010101011100000001011000001001110001011101101001010110011000011111110011000001111110001100
00011011110011101001111010011100100111011101110101010100000000001000000001010000001000100001
01010100100000001101000001110010001101110101110101000101000010100010010001010110101000011000001
00111100101110011100101111011100100101011101100001010111001000010111010010010100110110001111101
11011001010101111000000100110000101111100100100011101101011010110001100011101111011010100101110
00011001110011111101111000010100110010001111110101100001000111001010110111000011010110011100001
11110110110001011011101001101010011110000111001100110111111111010000000100100000101101000100011
00101011111100001000011001010011111000111000110110110111011011010101101100000110111000111010011
01101000110110010111011110010101001110000011101100011010111011100010101011010000011001000011111
11010011000100111110101110001000101101010100110000001111100001100011001111011111100101000011110
00100110110101111011000100101110101100101000111100010110011010011111100111000011110110011001010
11111110010000001110100001101001001110011011101111101010100010000001010100001000001001010001010
10001010011101000111010010110100110011001111111111100000000011000000011110000011001100011111111
10110000001011100001001011001011001111001111100111100011110011011001111101111100010100011010000
10111001010010111000110010110111110011010001111100101100011100111011011110101101001000110011010
01111111000100000110101000111000010110110010011011110111101001010010011000110111111011101000101
01001010000011000100011110101011001000001111010001100100101111101100100010111101010010010000011
01101001110110011101011111010001000100101010110000000011100000011011000011101110011010101111
10000010001100010101111010
```

A. No. This is output of {8, 10} LFSR with seed 01101000010!

LFSR Encryption

Base64 Encoding

char	dec	binary
A	0	000000
B	1	000001
…	…	…
w	22	010110
…	…	…

Encryption.

- Convert text message to N bits.
- Initialize LFSR with given seed
- Generate N bits with LFSR.
- Take bitwise XOR of two bitstrings.
- Convert binary back into text.

S	**E**	**N**	**D**	**M**	**O**	**N**	**E**	**Y**	message
010010	000100	001101	000011	001100	001110	001101	000100	011000	base64
110010	010011	110110	111001	011010	111001	100010	111111	010010	LFSR bits
100000	010111	111011	111010	010110	110111	101111	111011	001010	XOR
g	**X**	**7**	**6**	**W**	**3**	**v**	**7**	**K**	encrypted

LFSR Decryption

Base64 Encoding

char	dec	binary
A	0	000000
B	1	000001
…	…	…
M	12	001100
…	…	…

Decryption.

- Convert encrypted message to binary.
- Initialize identical LFSR with same seed
- Generate N bits with LFSR.
- Take bitwise XOR of two bitstrings.
- Convert back into text.

g	**X**	**7**	**6**	**W**	**3**	**v**	**7**	**K**	encrypted
100000	010111	111011	111010	010110	110111	101111	111011	001010	base64
110010	010011	110110	111001	011010	111001	100010	111111	010010	LFSR bits
010010	000100	001101	000011	001100	001110	001101	000100	011000	XOR
S	**E**	**N**	**D**	**M**	**O**	**N**	**E**	**Y**	message

Goods and Bads of LFSRs

Good.

- Easily computed with simple machine.
- Very simple encryption/decryption processes.
- Bits have many of the same properties as random bits.
- Scalable: 20 cells for 1 million bits; 30 cells for 1 billion bits.

  [but need theory of finite groups to know where to put taps]

a commercially available LFSR

Bad.

- Still need secure, independent way to distribute LFSR seed.
- The bits are not truly random.

  [bits in our 11-bit LFSR cycle after $2^{11} - 1 = 2047$ steps]
- Experts have cracked LFSR encryption.

  [need more complicated machines]

Other LFSR Applications

What else can we do with a LFSR?

- DVD encryption with CSS.
- DVD decryption with DeCSS!
- Subroutine in military cryptosystems.

```
/*     efdtt.c     Author:  Charles M. Hannum <root@ihack.net>            */
/*     Usage is:  cat title-key scrambled.vob | efdtt >clear.vob          */

#define m(i)(x[i]^s[i+84])<<

                    unsigned char x[5]      ,y,s[2048];main(
                    n){for( read(0,x,5     );read(0,s ,n=2048
                             ); write(1   ,s,n)         )if(s
                   [y=s      [13]%8+20]  /16%4  ==1      ){int
                   i=m(      1)17  ^256 +m(0)   8,k      =m(2)
                   0,j=      m(4)   17^ m(3)   9^k*     2-k%8
                   ^8,a     =0,c    =26;for   (s[y]    -=16;
                  --c;j  *=2)a=     a*2^i&    1,i=i /2^j&1
                  <<24;for(j=        127;     ++j<n;c=c>
                                     y)
                                     c

                            +=y=i^i/8^i>>4^i>>12,
                    i=i>>8^y<<17,a^=a>>14,y=a^a*8^a<<6,a=a
                  >>8^y<<9,k=s[j],k          ="7Wo~'G_\216"[k
                   &7]+2^"cr3sfw6v;*k+>/n."[k>>4]*2^k*257/
                         8,s[j]=k^(k&k*2&34)*6^c+~y
                                  ;}}
```

http://www.cs.cmu.edu/~dst/DeCSS/Gallery

Typical Exam Question (TEQ) on LFSRs 1

Give first 10 steps of {3, 4} LFSR with initial fill `00001`.

TEQ on LFSRs 2

Goal. Decrypt/encrypt 300 characters (1800 bits).

Challenge. Can we use an 11-bit LFSR?

A. Yes, no problem.

B. No, the bits it produces are not truly random.

C. No, need a longer LFSR.

D. No, experts have cracked LFSRs

LFSR and "General Purpose Computer"

Important properties.

- Built from simple components.
- Scales to handle huge problems.
- Requires a deep understanding to use effectively.

Basic Component	LFSR	Computer
control	start, stop, load	same
clock	regular pulse	2.8 GHz pulse
memory	11 bits	1 GB
input	seed	sequence of bits
computation	shift, XOR	logic, arithmetic, …
output	pseudo-random bits	Sequence of bits

Critical difference. General purpose machine can be programmed to simulate ANY abstract machine.

A Profound Idea

Programming. Can write a Java program to simulate the operations of any abstract machine.

- Basis for theoretical understanding of computation. [stay tuned]
- Basis for bootstrapping real machines into existence. [stay tuned]

Stay tuned. See Assignment 5.

```
public class LFSR
{
    private int seed[];
    private int tap;
    private int N;

    public LFSR(String seed, int tap) { … }

    public int step() { … }

    public static void main(String[] args)
    {
       LFSR lfsr = new LFSR("01101000010", 8);
       for (int i = 0; i < 2000; i++)
          StdOut.println(lfsr.step());
   }
}
```

```
% java LFSR
11001001001111011011100101101011
11001100010111111010010000100111
01001011110011001001111...
```

A Profound Question

Q. What is a random number?

LFSR does not produce random numbers.

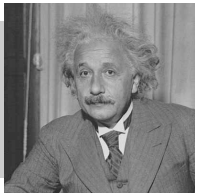
- It is a very simple deterministic machine.
- Not obvious how to distinguish the bits it produces from random.
- Experts have figured out how to do so.

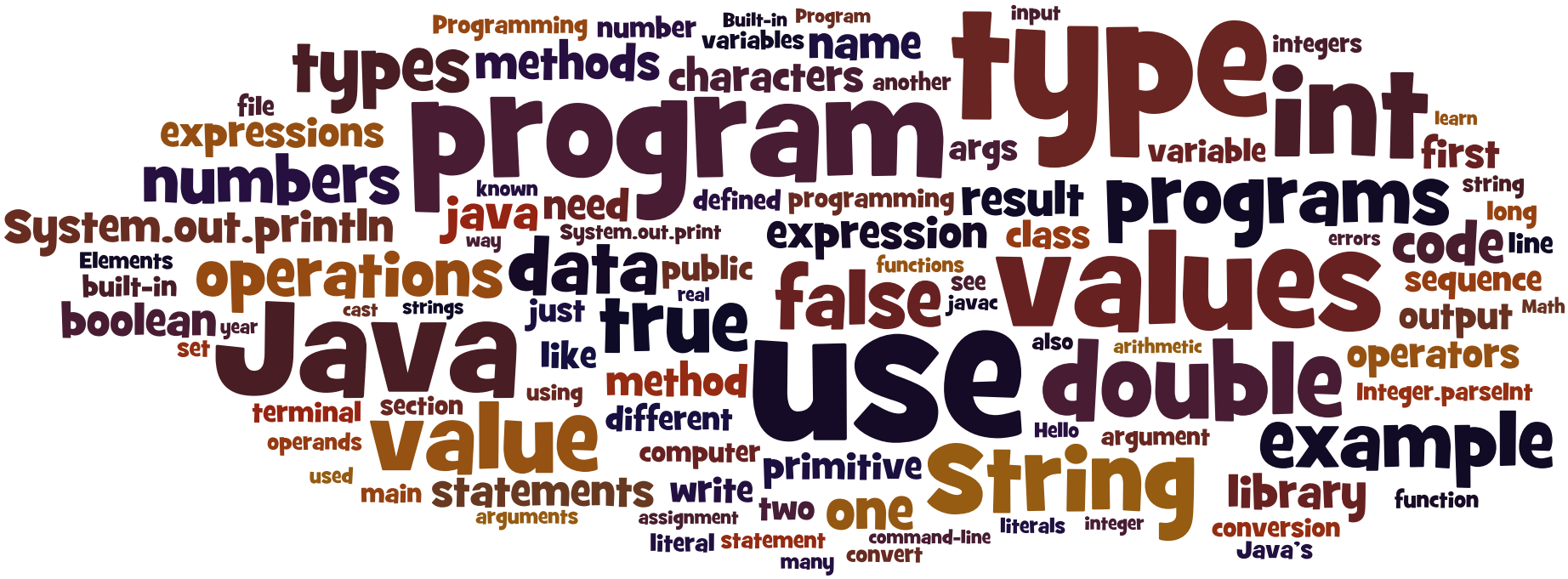
Q. Are random processes found in nature?

- Motion of cosmic rays or subatomic particles?
- Mutations in DNA?

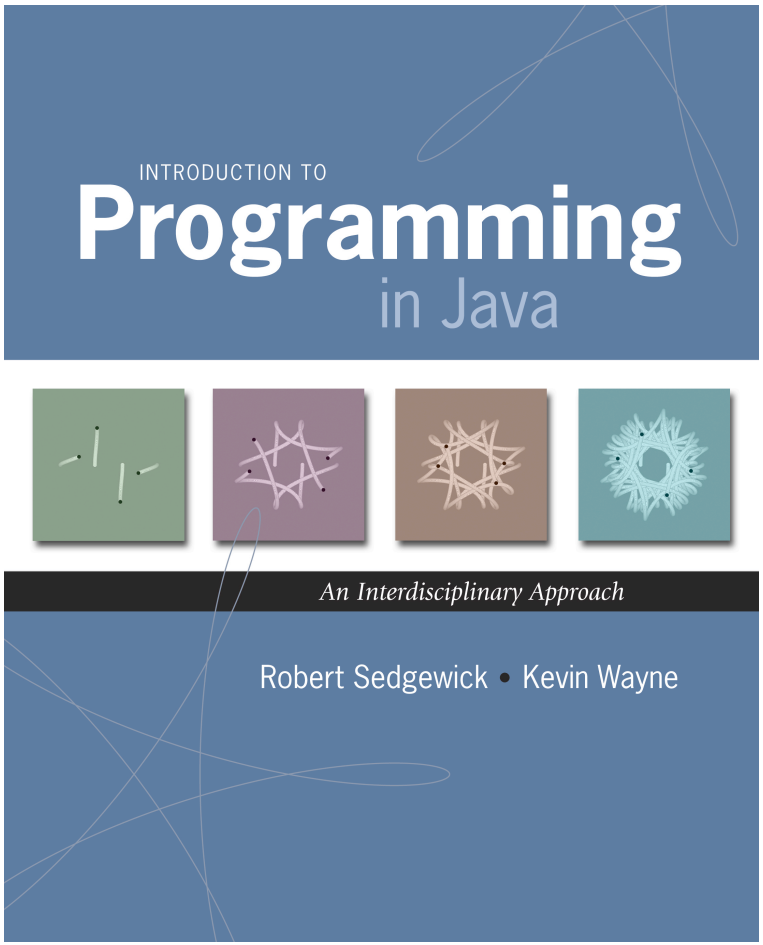
Q. Is the natural world a (not-so-simple) deterministic machine?

" God does not play dice. "
– Albert Einstein




Programming
number
Built-in
variables
Program
name
input
integers
types
methods
characters
another
type
int
file
expressions
program
args
variable
learn
first
numbers
known
java
need
defined
programming
result
programs
string
long
System.out.println
way
System.out.print
expression
class
errors
code
line
Elements
built-in
operations
data
public
functions
values
sequence
boolean
cast
strings
just
true
false
see
javac
real
output
Math
year
set
Java
like
use
also
arithmetic
double
operators
method
using
Integer.parseInt
terminal
section
different
Hello
argument
example
operands
value
computer
primitive
String
library
function
used
main
statements
write
two
one
assignment
literals
integer
conversion
arguments
literal
statement
command-line
Java's
many
convert

1.1 Your First Program

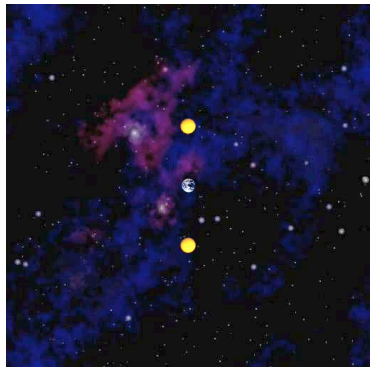


Why Programming?

Why programming? Need to tell computer what you want it to do.

Naive ideal. Natural language instructions.

"Please simulate the motion of N heavenly bodies, subject to Newton's laws of motion and gravity."

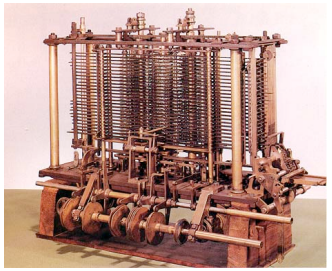


Prepackaged software solutions? Great, when what they do is what you want.

Programming. Enables you to make a computer do anything you want.

well, almost anything
[stay tuned]

Ada Lovelace



Analytic Engine

Languages

Machine languages. Tedious and error-prone.

Natural languages. Ambiguous; can be difficult to parse.

Kids Make Nutritious Snacks.
Red Tape Holds Up New Bridge.
Police Squad Helps Dog Bite Victim.
Local High School Dropouts Cut in Half.

[real newspaper headlines, compiled by Rich Pattis]

High-level programming languages. Acceptable tradeoff.

"Instead of imagining that our main task is to instruct a computer what to do, let us concentrate rather on explaining to human beings what we want a computer to do." – *Donald Knuth*

Why Program?

Why program?

- A natural, satisfying and creative experience.
- Enables accomplishments not otherwise possible.
- Opens new world of intellectual endeavor.

First challenge. Learn a programming language.

Next question. Which one?

Naive ideal. A single programming language.

Our Choice: Java

Java features.

- Widely used.
- Widely available.
- Embraces full set of modern abstractions.
- Variety of automatic checks for mistakes in programs.

Java economy.

$100 billion,
5 million developers

- Mars rover.
- Cell phones.
- Blu-ray Disc.
- Web servers.
- Medical devices.
- Supercomputing.
- ...

James Gosling
http://java.net/jag

Why Java?

Java features.

- Widely used.
- Widely available.
- Embraces full set of modern abstractions.
- Variety of automatic checks for mistakes in programs.

Facts of life.

- No language is perfect.
- We need to choose some language.

> *"There are only two kinds of programming languages: those people always [gripe] about and those nobody uses."*
> *– Bjarne Stroustrup*

Our approach.

- Minimal subset of Java.
- Develop general programming skills that are applicable to many languages

It's not about the language!

A Rich Subset of the Java Language

Built-In Types	
`int`	`double`
`long`	`String`
`char`	`boolean`

System
`System.out.println()`
`System.out.print()`
`System.out.printf()`

Math Library	
`Math.sin()`	`Math.cos()`
`Math.log()`	`Math.exp()`
`Math.sqrt()`	`Math.pow()`
`Math.min()`	`Math.max()`
`Math.abs()`	`Math.PI`

Flow Control	
`if`	`else`
`for`	`while`

Parsing
`Integer.parseInt()`
`Double.parseDouble()`

Primitive Numeric Types		
`+`	`-`	`*`
`/`	`%`	`++`
`--`	`>`	`<`
`<=`	`>=`	`==`
`!=`		

Boolean	
`true`	`false`
`||`	`&&`
`!`	

Punctuation	
`{`	`}`
`(`	`)`
`,`	`;`

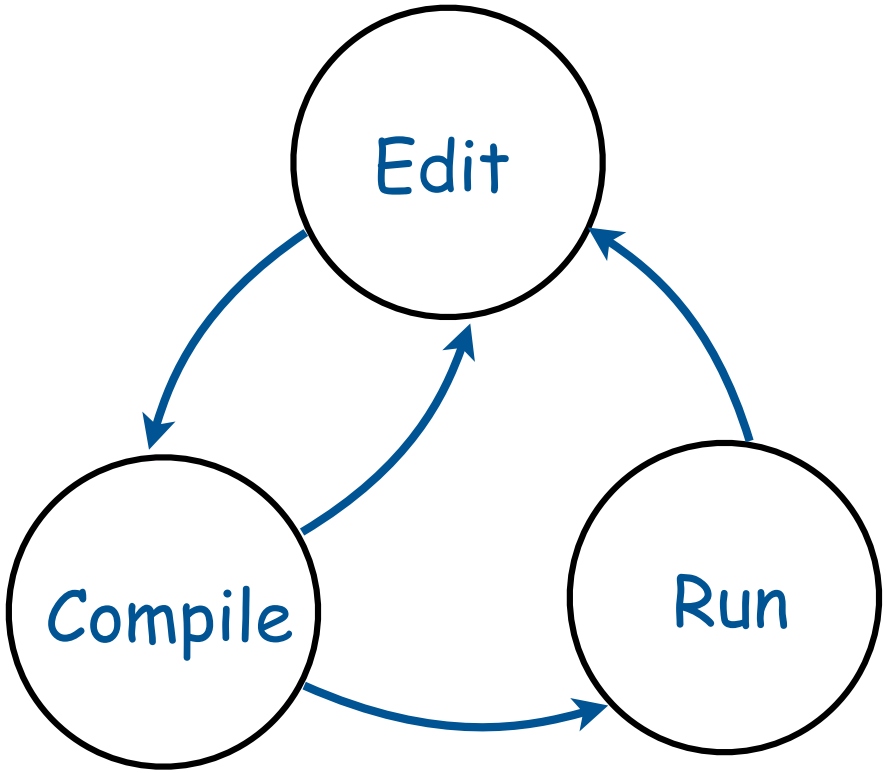
Assignment
`=`

String	
`+`	`""`
`length()`	`compareTo()`
`charAt()`	`matches()`

Arrays
`a[i]`
`new`
`a.length`

Objects	
`class`	`static`
`public`	`private`
`final`	`toString()`
`new`	`main()`

Program Development



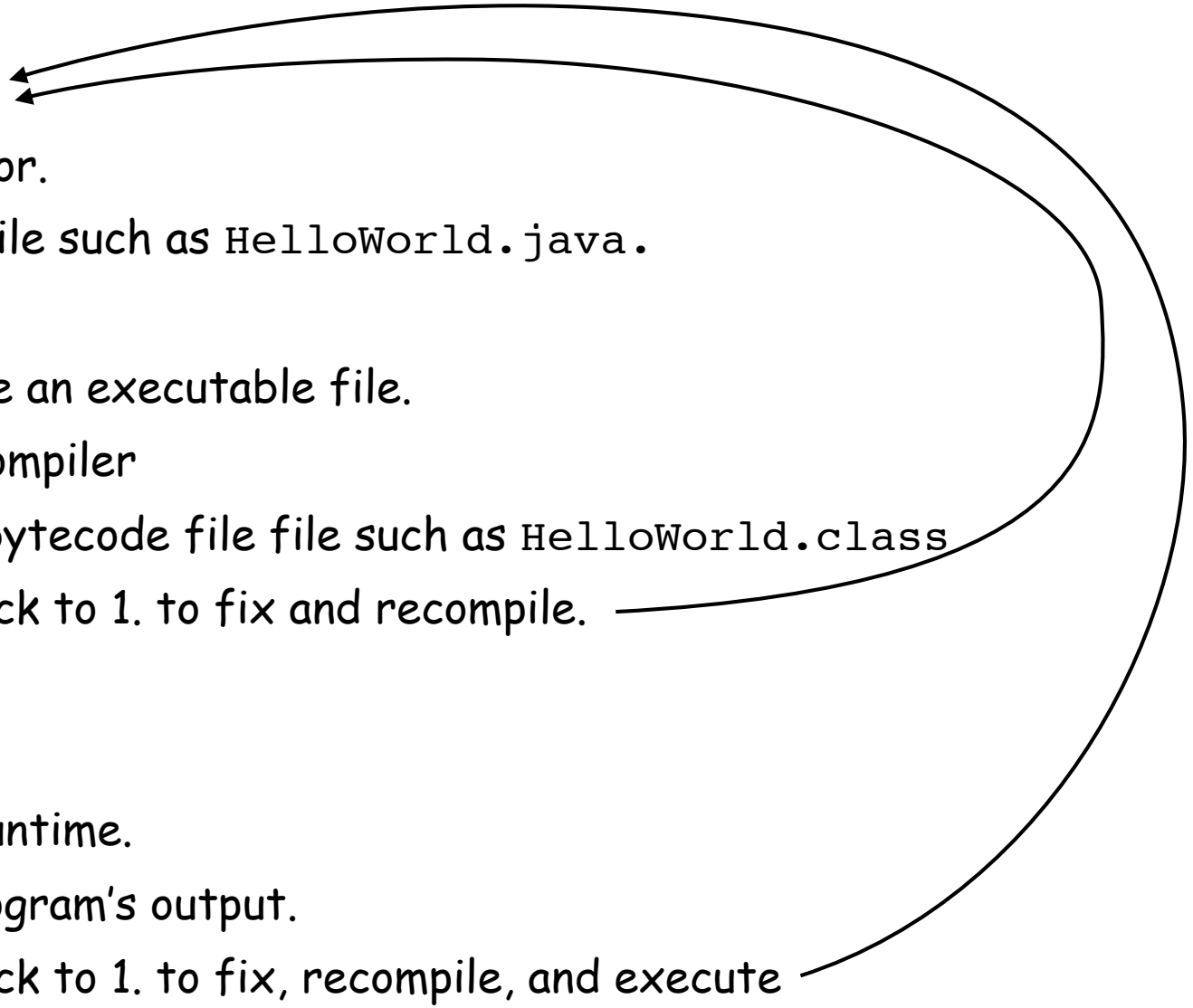
Program Development

Program development in Java (bare-bones)

1. Edit your program.
   - Use a text editor.
   - Result: a text file such as `HelloWorld.java`.

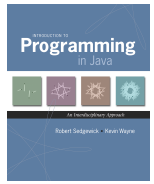
2. Compile it to create an executable file.
   - Use the Java compiler
   - Result: a Java bytecode file file such as `HelloWorld.class`
   - Mistake? Go back to 1. to fix and recompile.

3. Run your program.
   - Use the Java runtime.
   - Result: your program's output.
   - Mistake? Go back to 1. to fix, recompile, and execute



Program Development (virtual terminals)

Program development in Java (using virtual terminals).

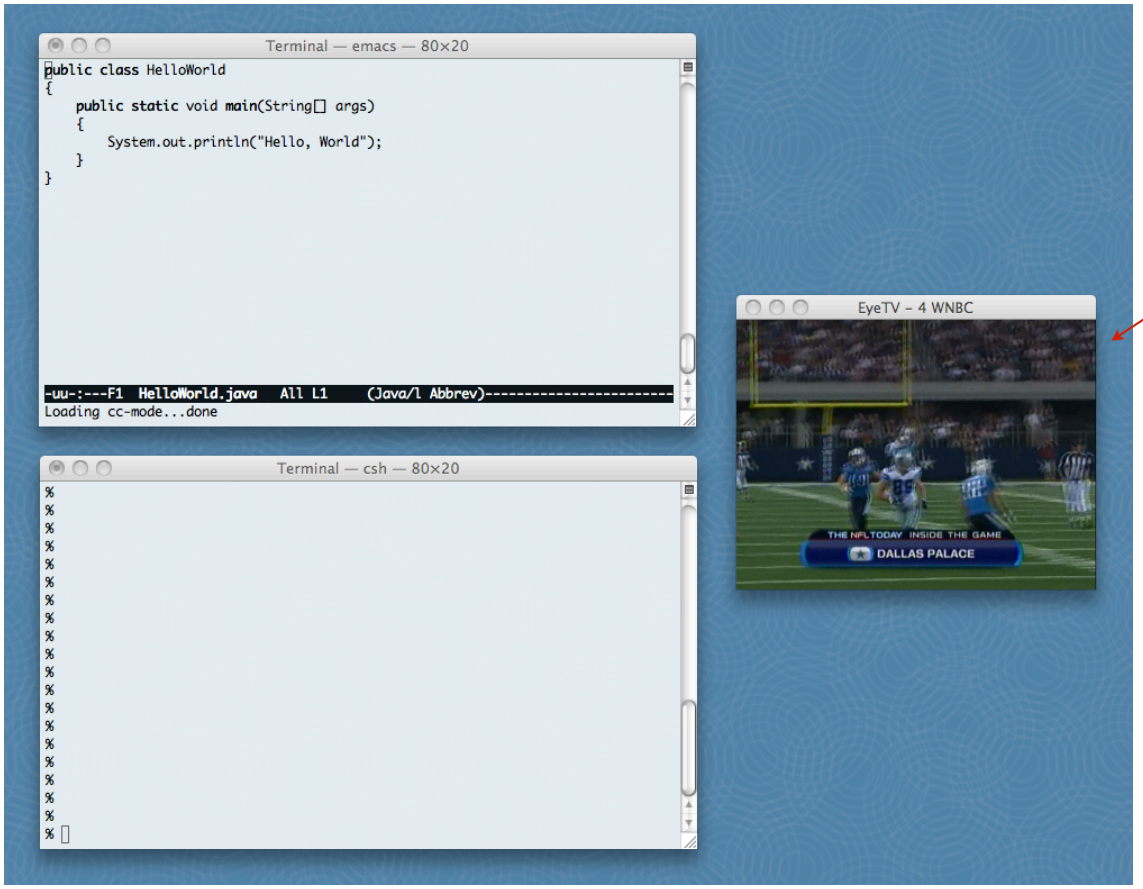


1. Edit your program using any text editor.
2. Compile it to create an executable file.
3. Run your program.

editor running in virtual terminal

second terminal for commands

virtual TV

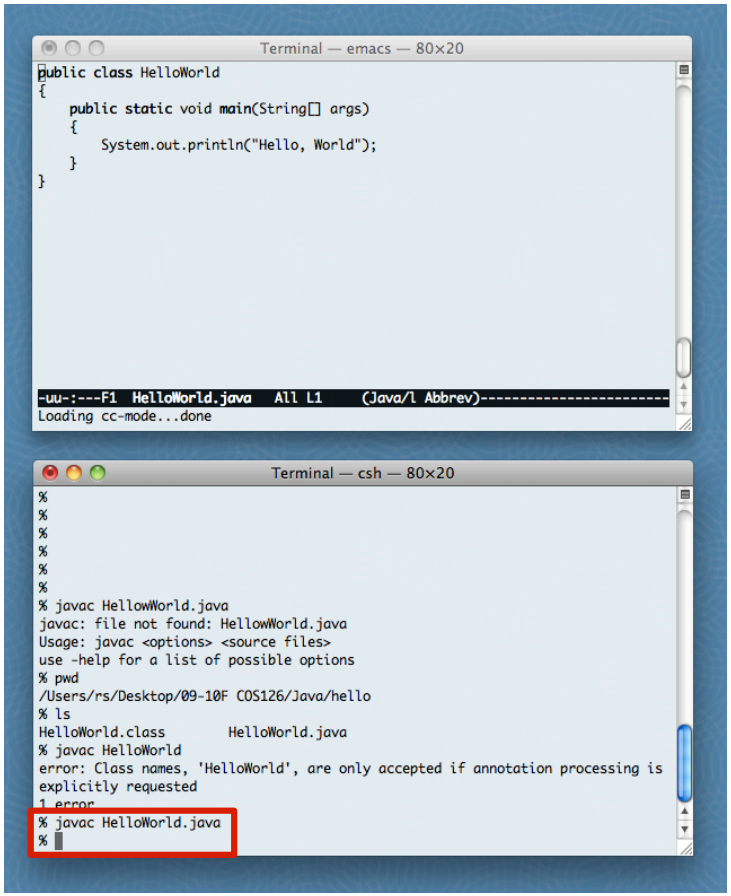


Program Development (virtual terminals)

Program development in Java (using virtual terminals).

1. Edit your program.
2. Compile it by typing `javac HelloWorld.java` at the command line.
3. Run your program.

creates `HelloWorld.class`



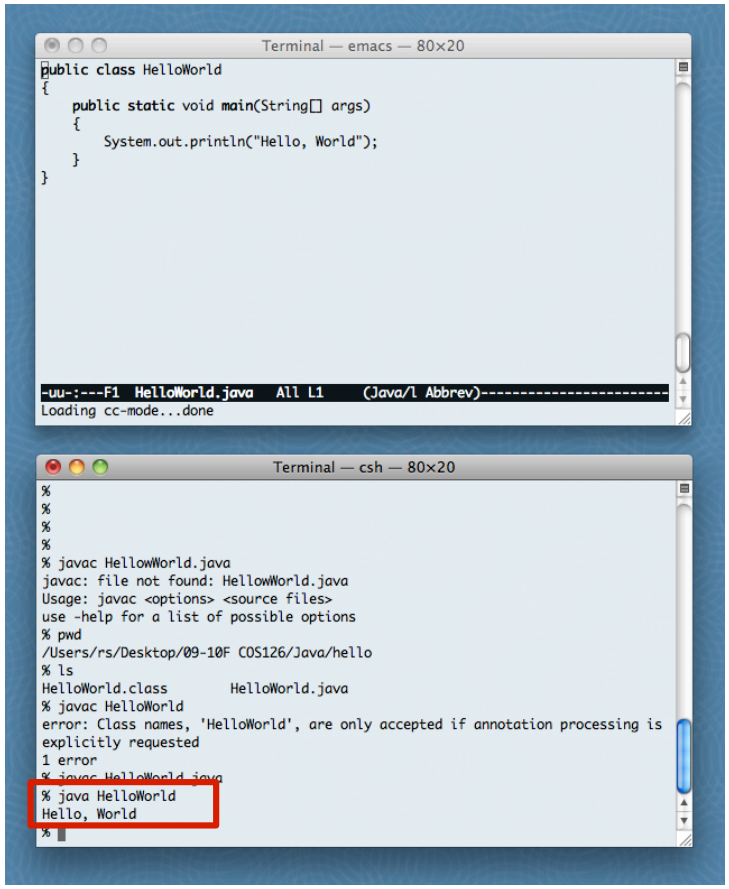
invoke Java compiler at command line

Program Development (virtual terminals)

Program development in Java (using virtual terminals).

1. Edit your program.
2. Compile it to create an executable file.
3. Run your program by typing `java HelloWorld` at the command line.

uses `HelloWorld.class`

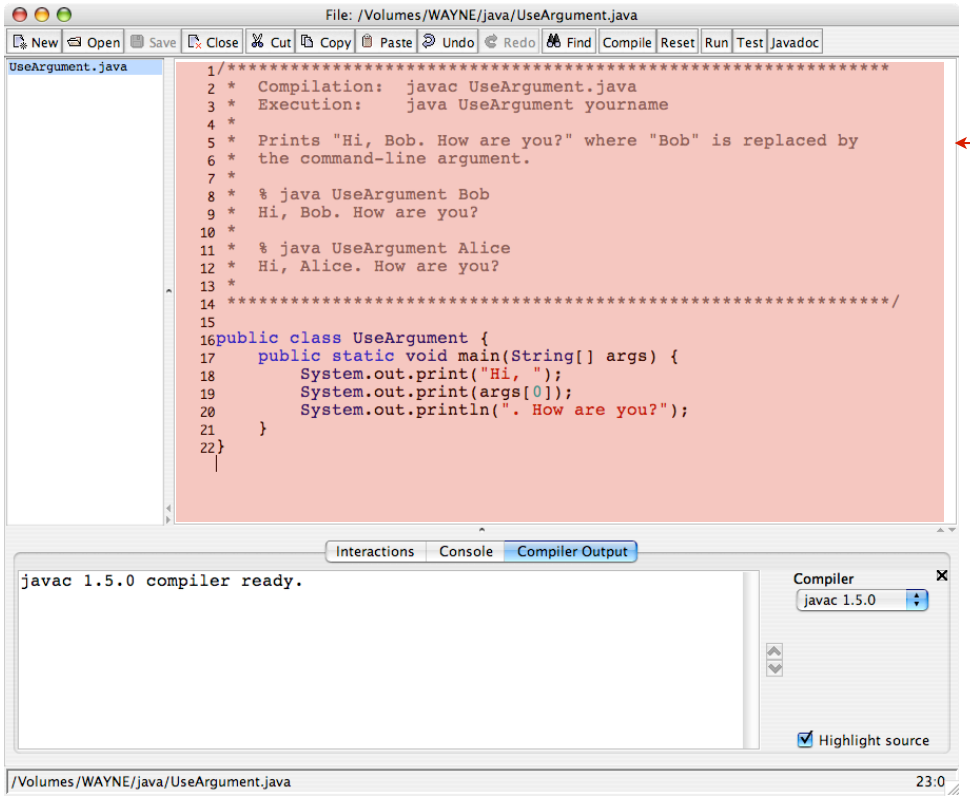


invoke Java runtime at command line

Program Development (using DrJava)

Program development in Java (using DrJava).

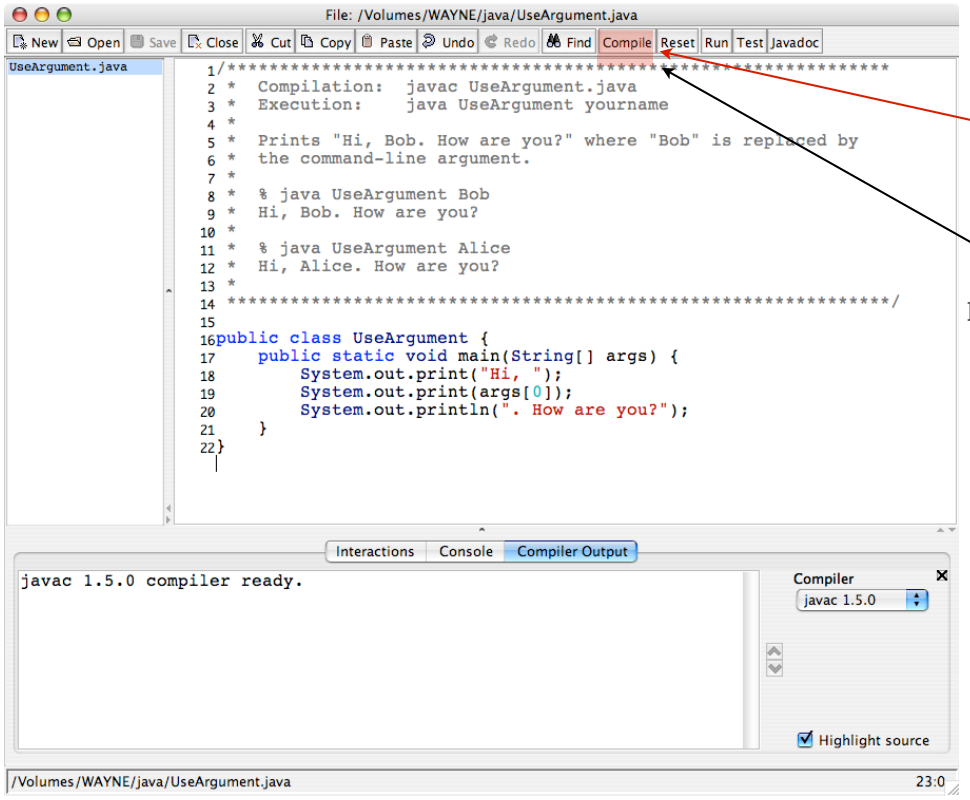
1. Edit your program using the built-in text editor.
2. Compile it to create an executable file.
3. Run your program.



Program Development (using DrJava)

Program development in Java (using DrJava).

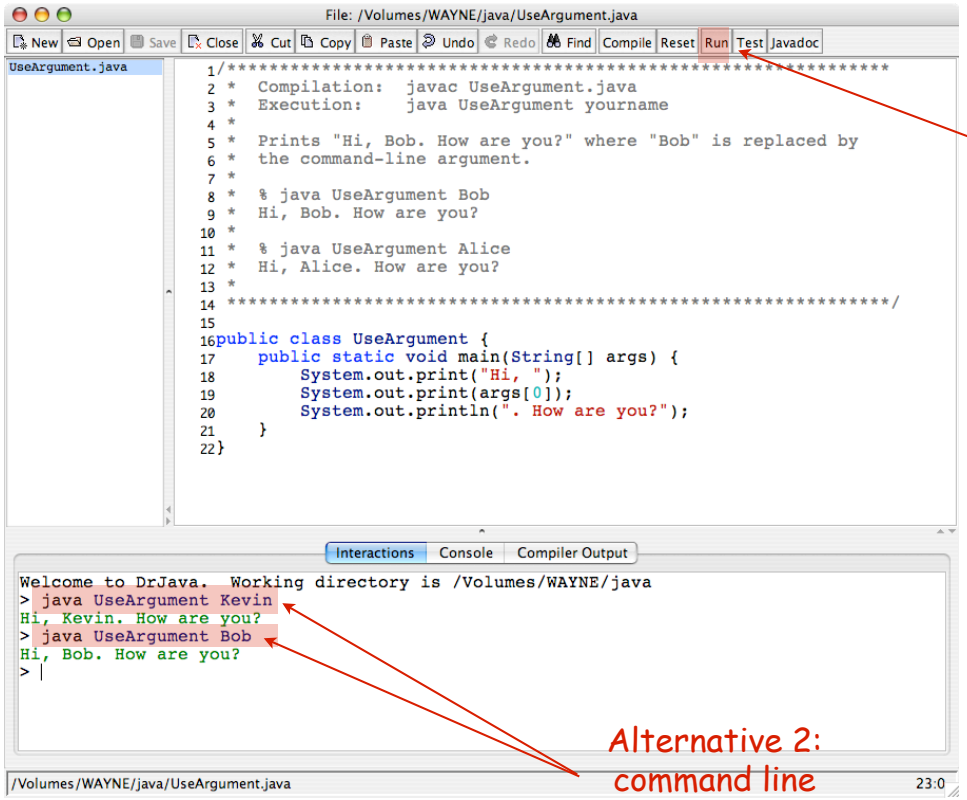
1. Edit your program.
2. Compile it by clicking the "compile" button.
3. Run your program.



Program Development (using DrJava)

Program development in Java (using DrJava).

1. Edit your program.
2. Compile it to create an executable file.
3. Run your program by clicking the "run" button or using the command line.



Note: Program Style

Three versions of the same program.

```
// java HelloWorld
public class HelloWorld
{
    public static void main(String[] args)
    {
        System.out.println("Hello, World");
    }
}
```

```
/*************************************************************************
 *  Compilation:  javac HelloWorld.java
 *  Execution:    java HelloWorld
 *
 *  Prints "Hello, World". By tradition, this is everyone's first program.
 *
 *  % java HelloWorld
 *  Hello, World
 *
 ************************************************************************/

public class HelloWorld {

    public static void main(String[] args) {
        System.out.println("Hello, World");
    }
}
```

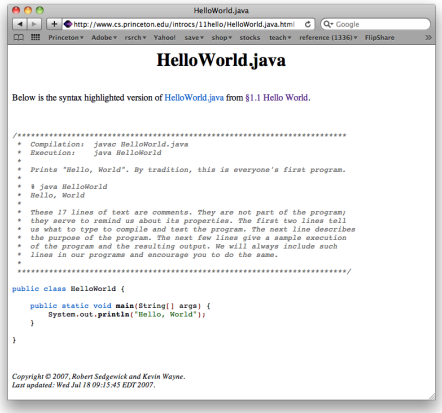
Fonts, color, comments, and extra space are not relevant to Java.

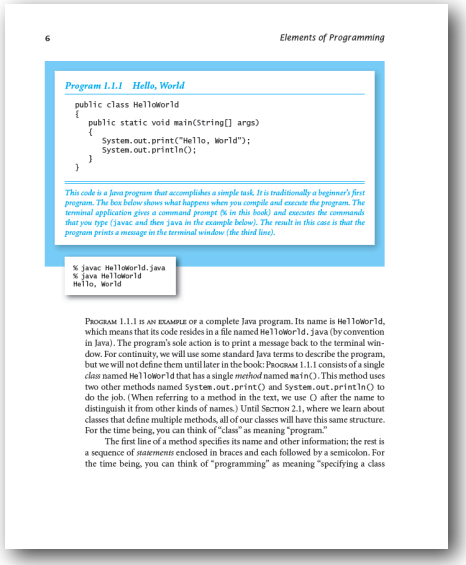

```
public class HelloWorld { public static void main(String[] args) { System.out.println("Hello, World"); } }
```

Note: Program Style

Different styles are appropriate in different contexts.

- DrJava
- Booksite
- Book
- COS 126 assignment





Enforcing consistent style can

- Stifle creativity.
- Confuse style rules with language rules.

Emphasizing consistent style can

- Make it easier to spot errors.
- Make it easier for others to read and use code.
- Enable development environment to provide useful visual cues.

Bottom line for COS 126: Life is easiest if you use DrJava style.

99% of program development

Debugging. Cyclic process of editing, compiling, and fixing mistakes (bugs).

You will make many mistakes as you write programs. It's normal.

> *As soon as we started programming, we found out to our surprise that it wasn't as easy to get programs right as we had thought. I can remember the exact instant when I realized that a large part of my life from then on was going to be spent in finding mistakes in my own programs.* – *Maurice Wilkes*

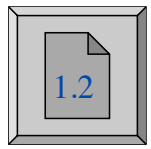
Program Development Environment. Software to support cycle of editing to fix mistakes, compiling programs, running programs, and examining output.

Examples: Terminal/editor, DrJava.

Naive ideal. "Please compile, execute, and debug my program".

Bad news. Even a computer can't find all the mistakes in your program.

profound idea
[stay tuned]



TEQ on Program Development

[easy if you did Exercise 1.1.2]

How do you cope with the following error messages?

A.
```
% javac HelloWorld.java
% java HelloWorld.java
Main method not public.
```

B.
```
% javac HelloWorld.java
HelloWorld.java:3: invalid method declaration; return type required
     public static main(String[] args)
                   ^
```

Program Development Environments: A Short History

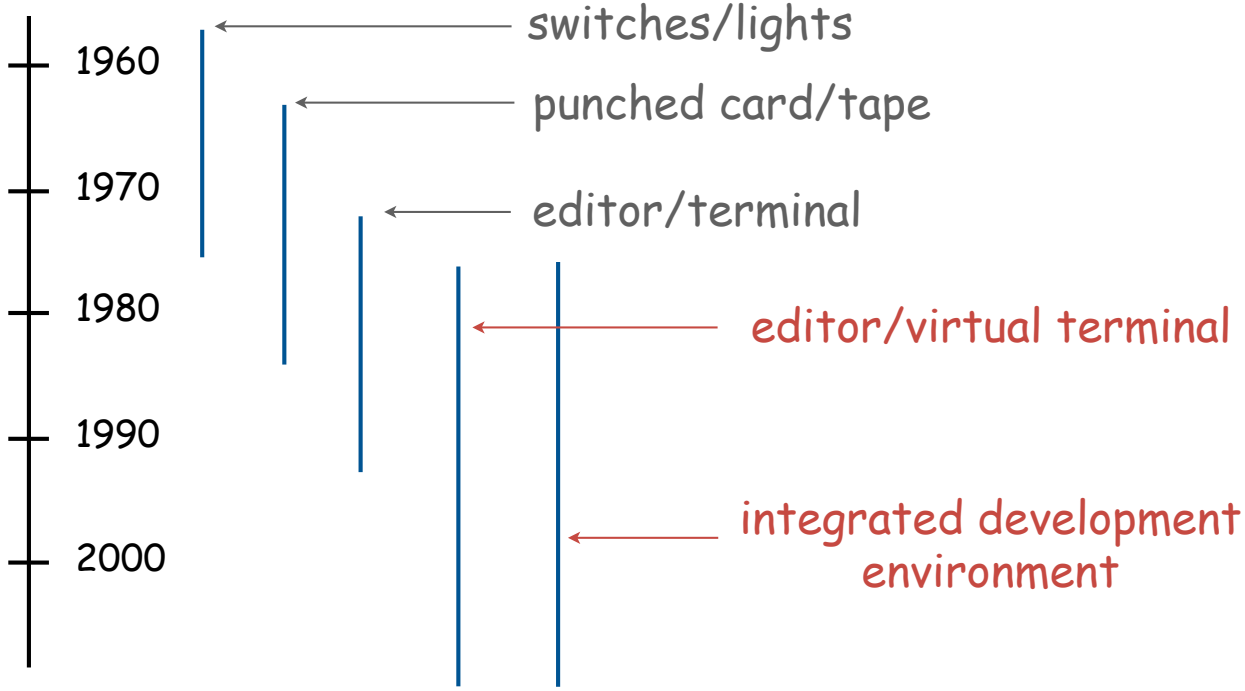
Historical context is important in computer science

- We regularly use old software.
- We regularly emulate old hardware.
- We depend upon old concepts and designs.

First requirement in any computer system: program development

Widely-used methods:

- switches/lights
- punched cards
- terminal
- editor/virtual terminal
- IDE



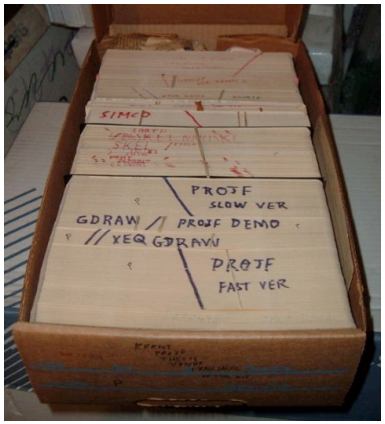
Switches and Lights

Use switches to enter binary program code, lights to read results

PDP-8, circa 1970

Punched Cards/Line Printer

Use punched cards for program code, line printer for output

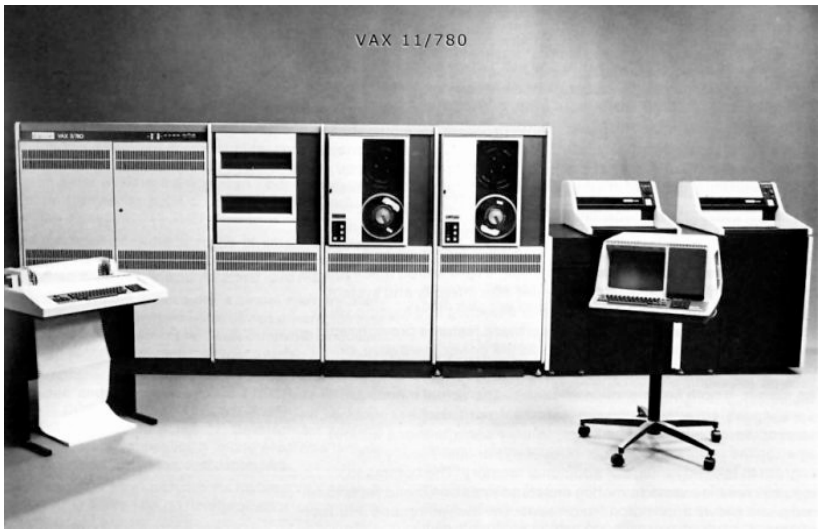


IBM System 360, circa 1975

Timesharing Terminal

Use terminal for editing program, reading output, and controlling computer

VAX 11/780 circa 1977



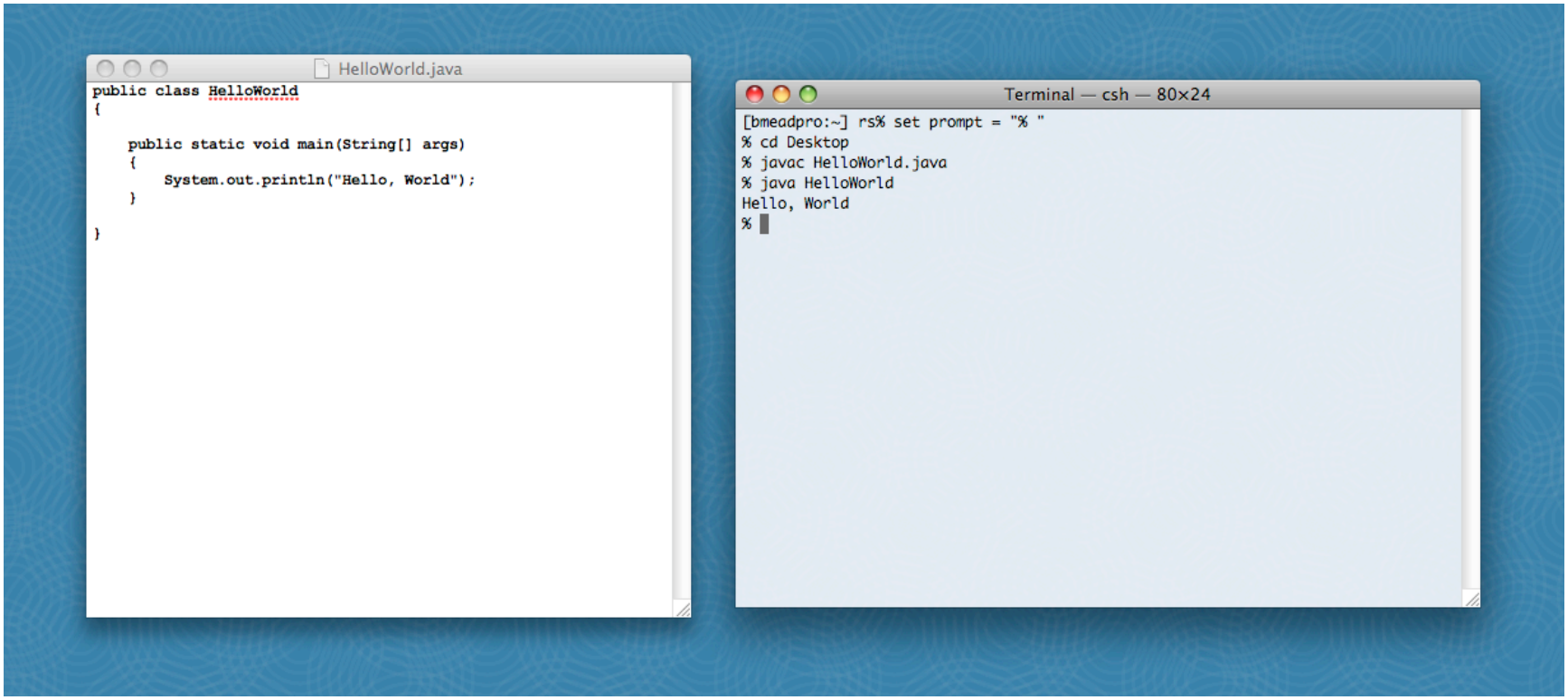
VT-100 terminal

Timesharing: allowed many people to simultaneously use a single machine.

Editor and Virtual Terminal on a Personal Computer

Use an editor to create and make changes to the program text.

Use a virtual terminal to invoke the compiler and run the executable code.



Pros:

- Works with any language.
- Useful for other tasks.
- Used by professionals.

Cons:

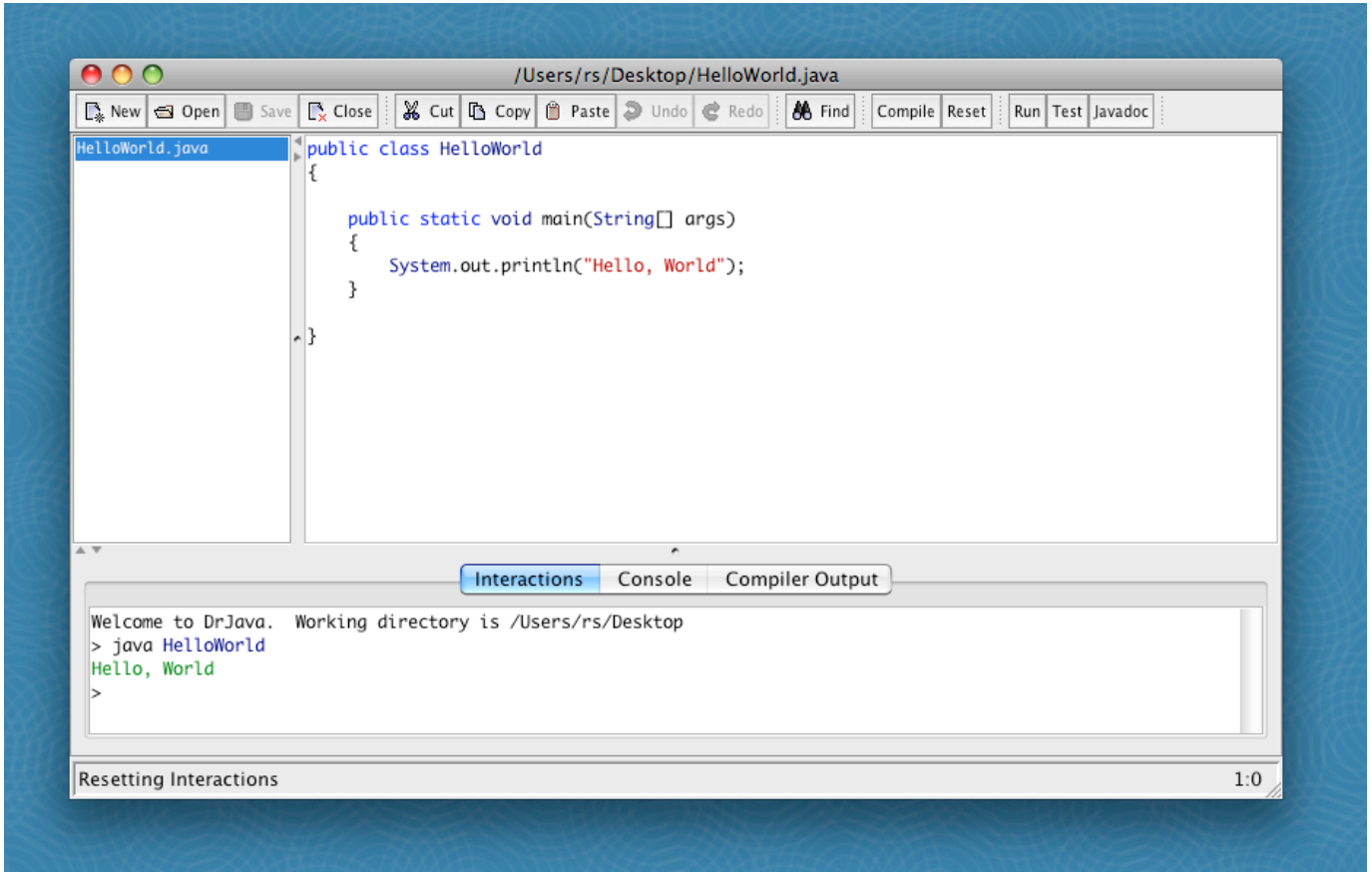
- Good enough for long programs?
- Dealing with two applications.

Integrated Development Environment

Use a customized application for all program development tasks.

Ex. drjava

http://drjava.org



Pros:

- Easy-to-use language-specific tools.
- System-independent (in principle).
- Used by professionals.

Cons:

- Overkill for short programs?
- Large application to learn and maintain.
- Skills may not transfer to other languages.

Lessons from Short History

First requirement in any computer system: program development

Programming is primarily a process of finding and fixing mistakes.

Program development environment must support cycle of editing to fix errors, compiling program, running program, and examining output.

Two approaches that have served for decades:

- editor and virtual terminal
- integrated development environment

Xerox Alto 1978

Macbook Air 2008

1.2 Built-in Types of Data

Built-in Data Types

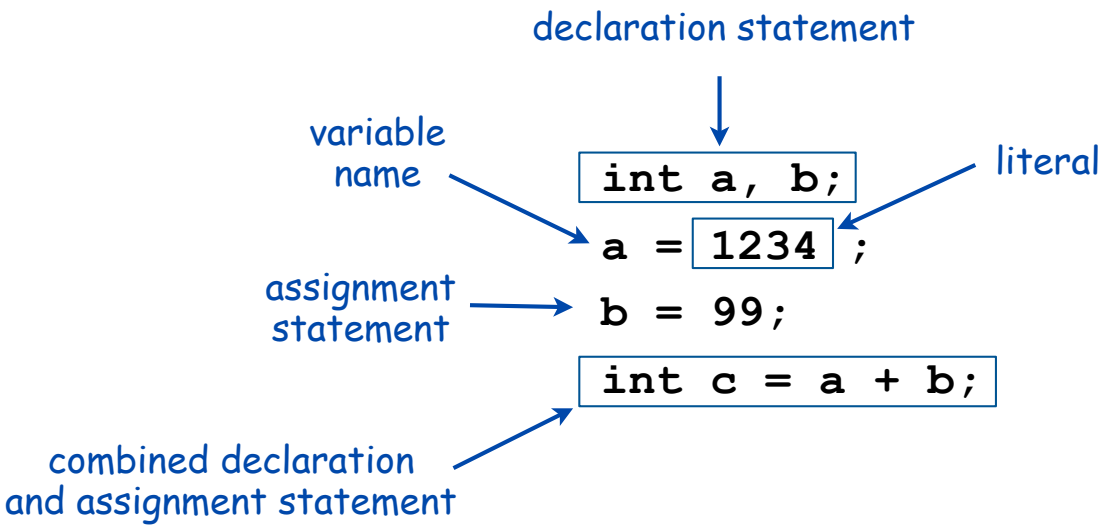
Data type. A set of values and operations defined on those values.

type	set of values	literal values	operations
`char`	characters	`'A'` `'@'`	compare
`String`	sequences of characters	`"Hello World"` `"CS is fun"`	concatenate
`int`	integers	`17` `12345`	add, subtract, multiply, divide
`double`	floating-point numbers	`3.1415` `6.022e23`	add, subtract, multiply, divide
`boolean`	truth values	`true` `false`	and, or, not

Basic Definitions

Variable. A name that refers to a value.

Assignment statement. Associates a value with a variable.



Trace

Trace. Table of variable values after each statement.

	a	b	t
`int a, b;`	undefined	undefined	undefined
`a = 1234;`	**1234**	undefined	undefined
`b = 99;`	1234	**99**	undefined
`int t = a;`	1234	99	**1234**
`a = b;`	**99**	99	1234
`b = t;`	99	**1234**	1234

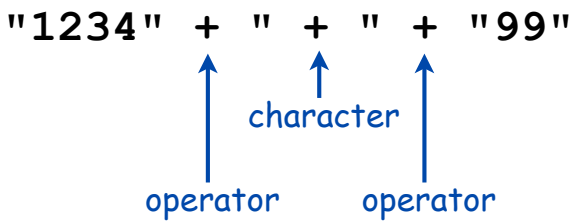
Text

`String` data type. Useful for program input and output.

values	sequences of characters
typical literals	`"Hello, " "1 " " * "`
operation	concatenate
operator	+

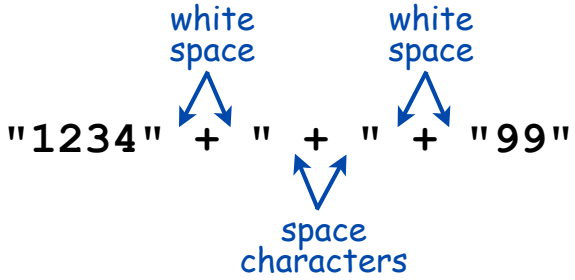
`String` data type

Important note: meaning of characters depends on context!



expression	value
`"Hi, " + "Bob"`	`"Hi, Bob"`
`"1" + " 2 " + "1"`	`"1 2 1"`
`"1234" + " + " + "99"`	`"1234 + 99"`
`"1234" + "99"`	`"123499"`

`String` concatenation examples



Example: Subdivisions of a Ruler

```
public class Ruler
{
   public static void main(String[] args)
   {
      String ruler1 = "1";
      String ruler2 = ruler1 + " 2 " + ruler1;
      String ruler3 = ruler2 + " 3 " + ruler2;
      String ruler4 = ruler3 + " 4 " + ruler3;
      System.out.println(ruler4);
   }
}
```

```
"1"
"1 2 1"
"1 2 1 3 1 2 1"
```

string concatenation

```
% java Ruler
1 2 1 3 1 2 1 4 1 2 1 3 1 2 1
```

Integers

`int` data type. Useful for calculations, expressing algorithms.

values	integers between -2^{31} and $+2^{31} - 1$				
typical literals		`1234` `99` `-99` `0` `1000000`			
operations	add	subtract	multiply	divide	remainder
operators	`+`	`-`	`*`	`/`	`%`

`int` data type

expression	value	comment
`5 + 3`	8	
`5 - 3`	2	
`5 * 3`	15	
`5 / 3`	1	no fractional part
`5 % 3`	2	remainder
`1 / 0`		run-time error
`3 * 5 - 2`	13	* has precedence
`3 * 5 / 2`	5	/ has precedence
`3 - 5 - 2`	-4	left associative
`( 3 - 5 ) - 2`	-4	better style

examples of `int` operations

Integer Operations

```
public class IntOps
{
   public static void main(String[] args)
   {
      int a = Integer.parseInt(args[0]);
      int b = Integer.parseInt(args[1]);
      int sum  = a + b;
      int prod = a * b;
      int quot = a / b;
      int rem  = a % b;
      System.out.println(a + " + " + b + " = " + sum);
      System.out.println(a + " * " + b + " = " + prod);
      System.out.println(a + " / " + b + " = " + quot);
      System.out.println(a + " % " + b + " = " + rem);
   }
}
```

command-line arguments

Java automatically converts `a`, `b` , and `rem` to type `String`

```
% javac IntOps.java
% java IntOps 1234 99
1234 + 99 = 1333
1234 * 99 = 122166
1234 / 99 = 12
1234 % 99 = 46
```

1234 = 12*99 + 46

Floating-Point Numbers

`double` data type. Useful in scientific applications.

values	approximations to real numbers				
typical literals	`3.14159`	`6.022e23`	`-3.0`	`2.0`	`1.4142135623730951`
operations	add	subtract	multiply	divide	remainder
operators	`+`	`-`	`*`	`/`	`%`

`double` data type

expression	value	
`3.141 + .03`	`3.171`	
`3.141 - .03`	`3.111`	
`6.02e23/2`	`3.01E+23`	
`5.0 / 3.0`	`1.6666666666666700`	
`10.0 % 3.141`	`0.577`	
`1.0 / 0.0`	`Infinity`	← special value
`Math.sqrt(2.0)`	`1.4142135623731000`	
`Math.sqrt(-1.0)`	`NaN`	← special value "not a number"

examples of `double` operations

Excerpts from Java's Math Library

public class Math

`double abs(double a)`	absolute value of a
`double max(double a, double b)`	maximum of a and b
`double min(double a, double b)`	minimum of a and b

also defined for `int`, `long`, and `float`

`double sin(double theta)`	sine function
`double cos(double theta)`	cosine function
`double tan(double theta)`	tangent function

inverse functions `asin()`, `acos()`, and `atan()` also available

In radians. Use `toDegrees()` and `toRadians()` to convert.

`double exp(double a)`	exponential (e^a)
`double log(double a)`	natural log ($\log_e a$, or ln a)
`double pow(double a, double b)`	raise a to the bth power (a^b)
`long round(double a)`	found to the nearest integer
`double random()`	random number in [0. 1)
`double sqrt(double a)`	square root of a
`double E`	value of e (constant)
`double PI`	value of p (constant)

Quadratic Equation

Ex. Solve quadratic equation $x^2 + bx + c = 0$.

$$\text{roots} = \frac{-b \pm \sqrt{b^2 - 4c}}{2}$$

```
public class Quadratic
{
   public static void main(String[] args)
   {
      // Parse coefficients from command-line.
      double b = Double.parseDouble(args[0]);
      double c = Double.parseDouble(args[1]);

      // Calculate roots.
      double discriminant = b*b - 4.0*c;
      double d = Math.sqrt(discriminant);
      double root1 = (-b + d) / 2.0;
      double root2 = (-b - d) / 2.0;

      // Print them out.
      System.out.println(root1);
      System.out.println(root2);
   }
}
```

Testing

Testing. Some valid and invalid inputs.

```
% java Quadratic -3.0 2.0
2.0
1.0

% java Quadratic -1.0 -1.0
1.618033988749895
-0.618033988749895

% java Quadratic 1.0 1.0
NaN
NaN

% java Quadratic 1.0 hello
java.lang.NumberFormatException: hello

% java Quadratic 1.0
java.lang.ArrayIndexOutOfBoundsException
```

command-line arguments

golden ratio

not a number

$x^2 - 3x + 2$

$x^2 - x - 1$

$x^2 + x + 1$

Booleans

`boolean` data type. Useful to control logic and flow of a program.

values	true or false		
literals	`true`	`false`	
operations	and	or	not
operators	`&&`	`||`	`!`

`boolean` data type

`a`	`!a`
`true`	`false`
`false`	`true`

`a`	`b`	`a && b`	`a || b`
`false`	`false`	`false`	`false`
`false`	`true`	`false`	`true`
`true`	`false`	`flase`	`true`
`true`	`true`	`true`	`true`

Truth-table definitions of `boolean` operations

Comparison Operators

Comparison operators.

- Two operands of the same type.
- Result: a value of type `boolean`.

op	meaning	true	false
`==`	equal	`2 == 2`	`2 == 3`
`!=`	not equal	`3 != 2`	`2 != 2`
`<`	less than	`2 < 13`	`2 < 2`
`<=`	less than or equal	`2 <= 2`	`3 <= 2`
`>`	greater than	`13 > 2`	`2 < 13`
`>=`	greater than or equal	`3 >= 2`	`2 >= 3`

comparison operators

non-negative discriminant?	`( b*b - 4.0*a*c ) >= 0.0`
beginning of a century?	`( year % 100 ) == 0`
legal month?	`( month >= 1 ) && ( month <= 12 )`

comparison examples

Leap Year

Q. Is a given year a leap year?

A. Yes if either (i) divisible by 400 or (ii) divisible by 4 but not 100.

```
public class LeapYear
{
   public static void main(String[] args)
   {
      int year = Integer.parseInt(args[0]);
      boolean isLeapYear;

      // divisible by 4 but not 100
      isLeapYear = (year % 4 == 0) && (year % 100 != 0);

      // or divisible by 400
      isLeapYear = isLeapYear || (year % 400 == 0);


      System.out.println(isLeapYear);
   }
}
```

```
% java LeapYear 2004
true

% java LeapYear 1900
false

% java LeapYear 2000
true
```

Type Conversion

Type conversion. Convert from one type of data to another.

- Automatic (done by Java when no loss of precision; or with strings).
- Explicitly defined by function call.
- Cast (write desired type within parens).

expression	type	value	
`"1234" + 99`	`String`	`"123499"`	automatic
`Integer.parseInt("123")`	`int`	`123`	explicit
`(int) 2.71828`	`int`	`2`	cast
`Math.round(2.71828)`	`long`	`3`	explicit
`(int) Math.round(2.71828)`	`int`	`3`	cast
`(int) Math.round(3.14159)`	`int`	`3`	cast
`11 * 0.3`	`double`	`3.3`	automatic
`(int) 11 * 0.3`	`double`	`3.3`	cast, automatic
`11 * (int) 0.3`	`int`	`0`	cast
`(int) (11 * 0.3)`	`int`	`3`	cast, automatic

TEQ on Type Conversion

[not difficult if you read Exercise 1.2.6]

What is the type and value of each of the following expression?

A. `( 7 / 2 ) * 2.0`

B. `( 7 / 2.0 ) * 2`

Type Conversion Example: Random Integer

Ex. Generate a pseudo-random number between `0` and `N-1`.

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

      System.out.println("random integer is " + n);
   }
}
```

`String` to `int` (function)

`double` between `0.0` and `1.0`

`double` to `int` (cast)

`int` to `double` (automatic)

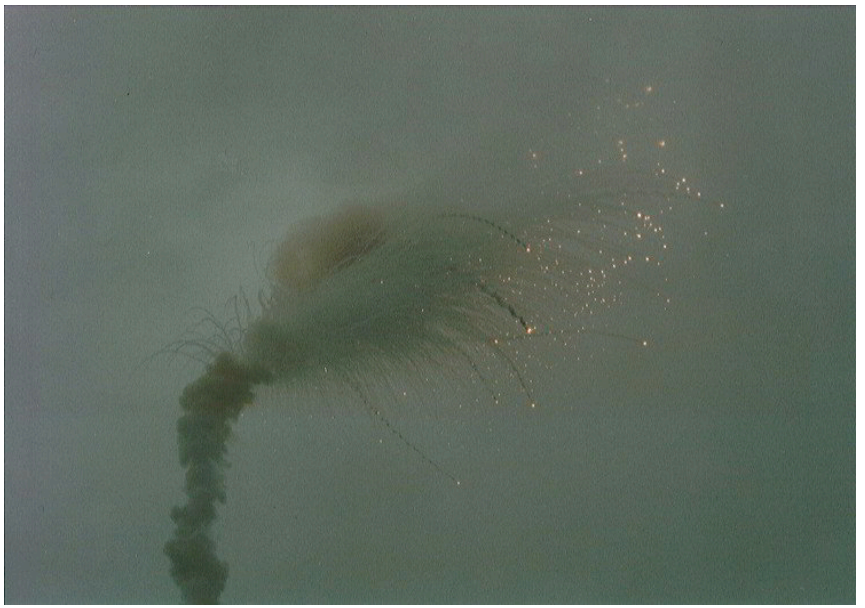
`int` to `String` (automatic)

```
% java RandomInt 6
random integer is 3

% java RandomInt 6
random integer is 0

% java RandomInt 10000
random integer is 3184
```

Summary

A data type is a set of values and operations on those values.

- `String` text processing, input and output.
- `double`, `int` mathematical calculation.
- `boolean` decision making.



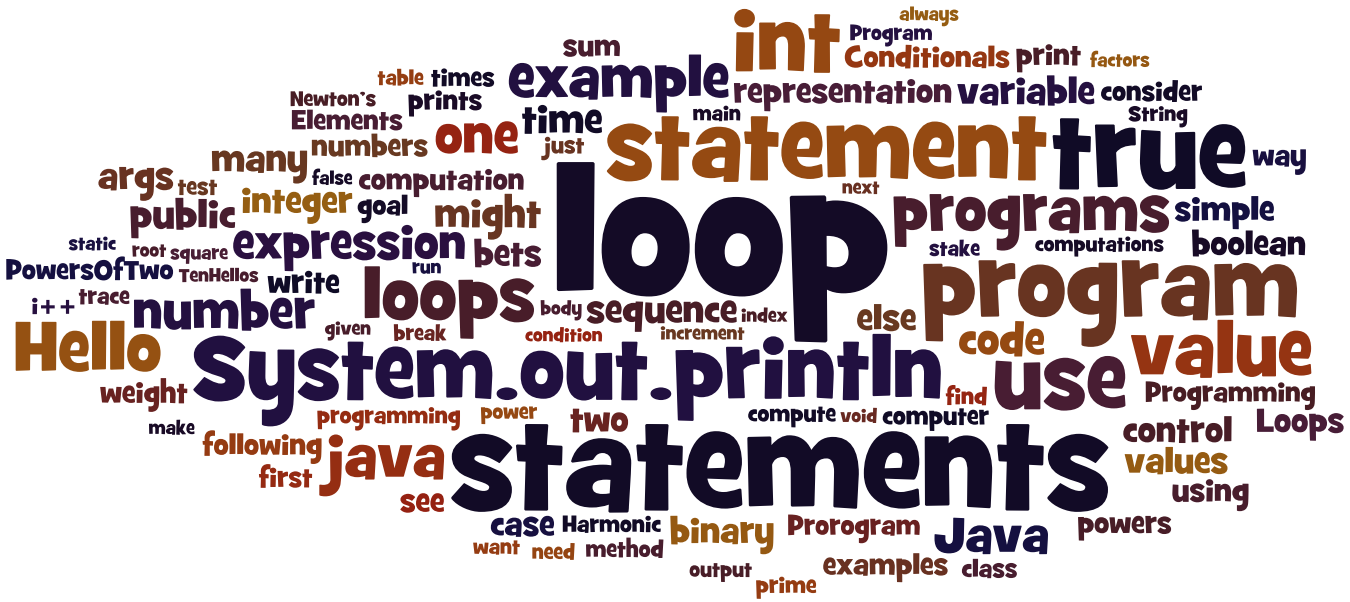
Example of bad type conversion

Be aware. In Java you must:

- Declare type of values.
- Convert between types when necessary.

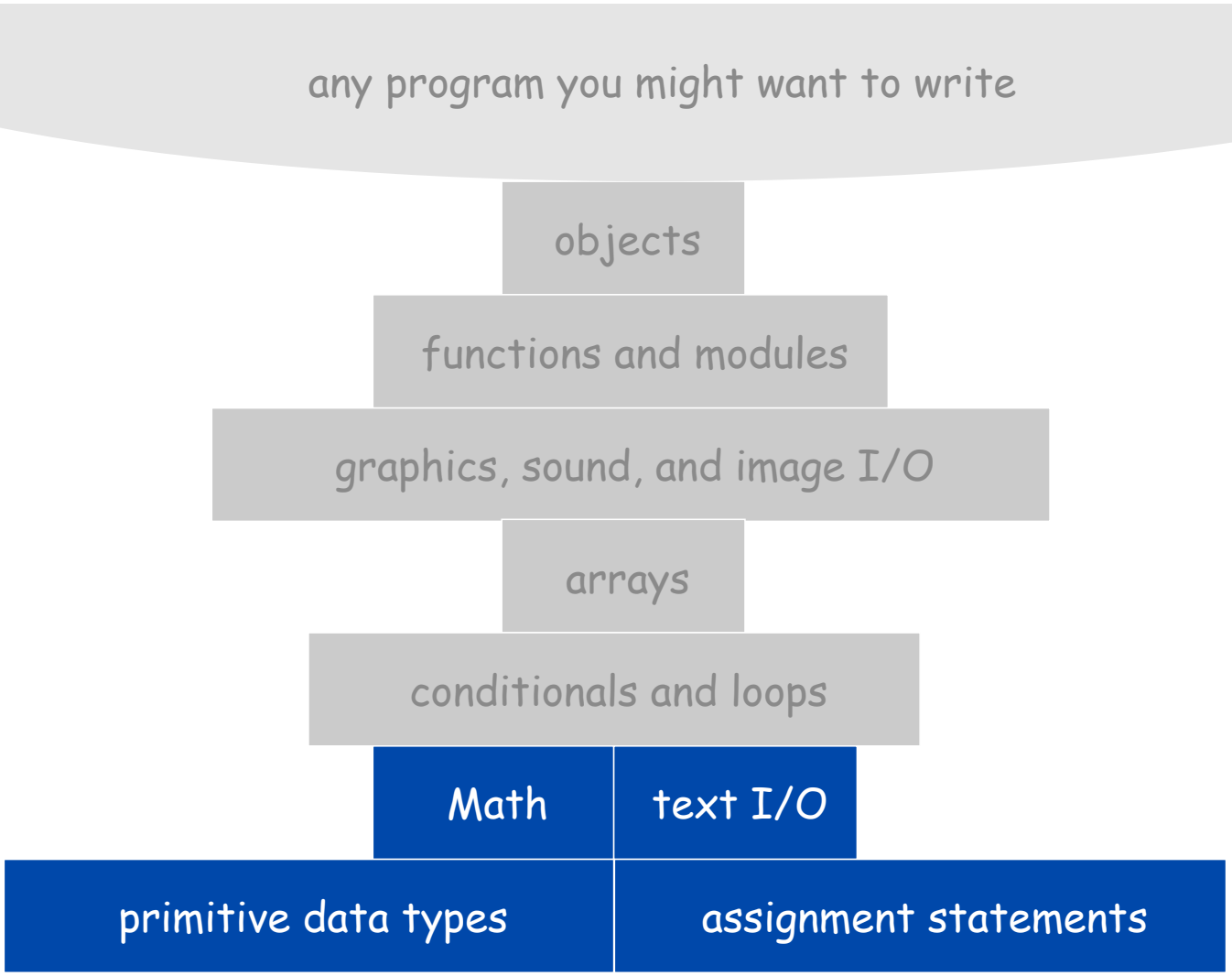
Why do we need types?

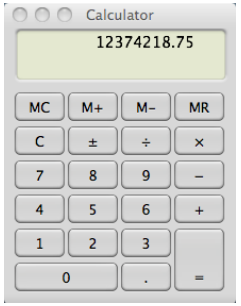
- Type conversion must be done at some level.
- Compiler can help do it correctly.
- Example: In 1996, Ariane 5 rocket exploded after takeoff because of bad type conversion.


always
Program
int
sum
Conditionals
print
factors
table
times
example
representation
variable
consider
Newton's
prints
main
String
Elements
time
statement
true
numbers
one
just
way
many
next
args
test
false
computation
integer
goal
might
loop
programs
simple
public
static
root
square
expression
bets
stake
computations
boolean
PowersOfTwo
TenHellos
run
write
program
i++
trace
loops
body
sequence
index
else
number
given
break
condition
increment
code
value
Hello
System.out.println
use
weight
find
Programming
programming
power
two
compute
void
computer
Loops
make
following
java
statements
control
first
values
see
using
case
Harmonic
binary
Prorogram
Java
powers
want
need
method
output
examples
class
prime

1.3 Conditionals and Loops

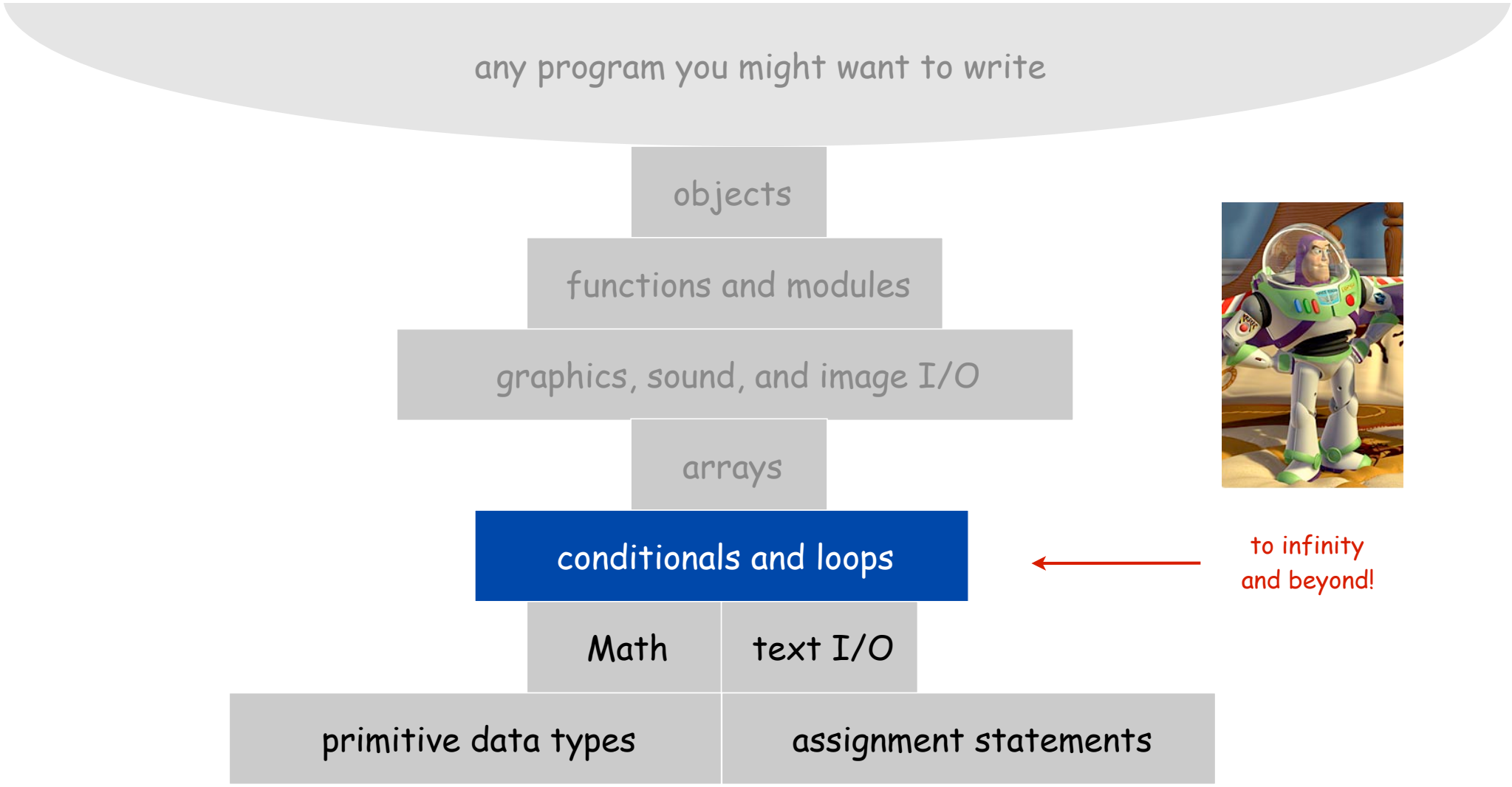
A Foundation for Programming





last lecture; equivalent to a calculator

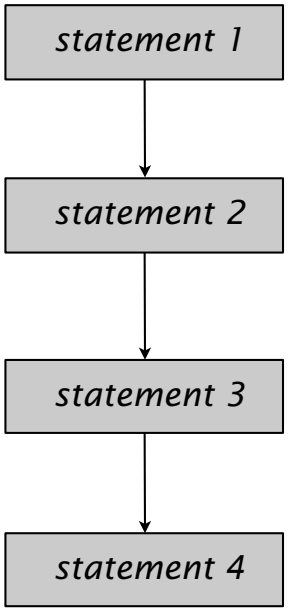
A Foundation for Programming



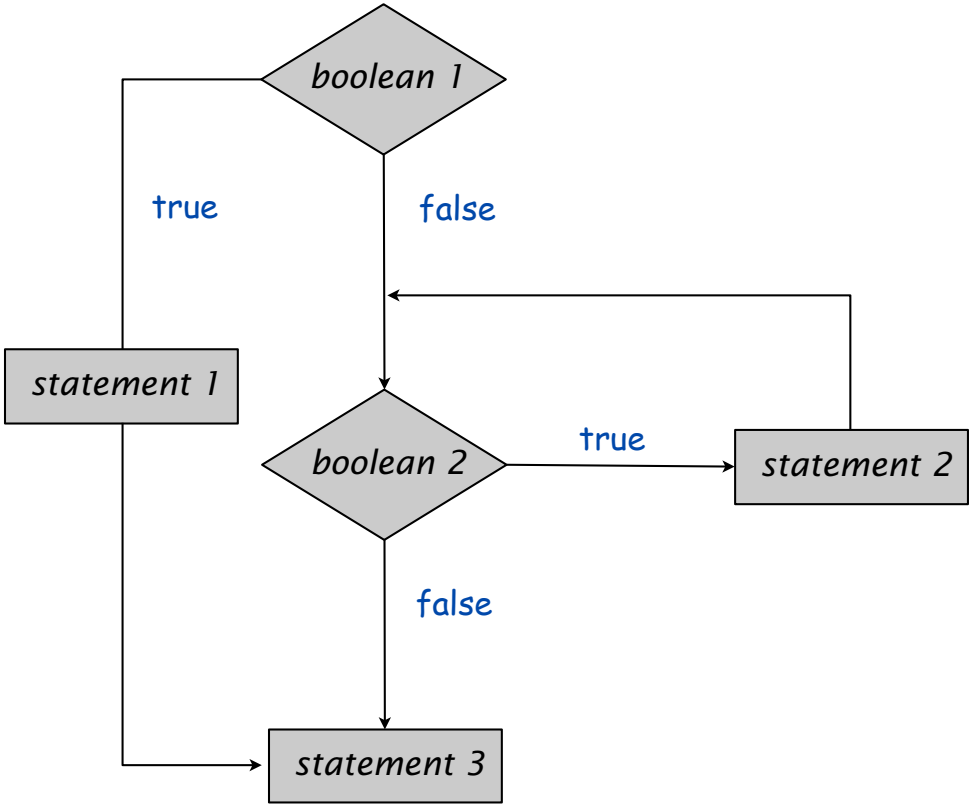
Conditionals and Loops

Control flow.

- Sequence of statements that are actually executed in a program.
- Conditionals and loops: enable us to choreograph control flow.



straight-line control flow



control flow with conditionals and loops

Conditionals

If Statement

The `if` statement. A common branching structure.

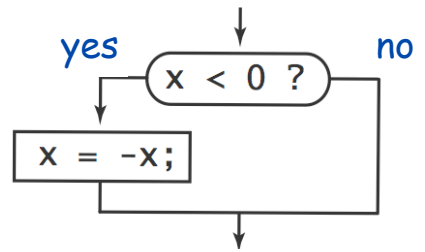
- Evaluate a `boolean` expression.
- If `true`, execute some statements.
- `else` option: If `false`, execute other statements.

boolean expression

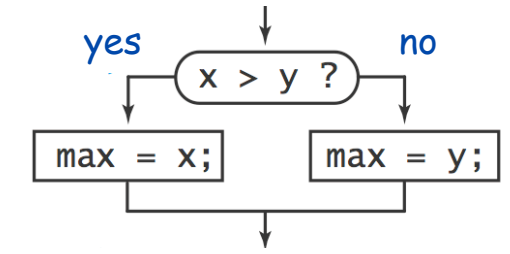
sequence of statements

```
if ( x > y )
{
   int t = x;
   x = y;
   y = t;
}
```

```
if (x < 0) x = -x;
```




```
if (x > y) max = x;
else       max = y;
```



If Statement

Ex. Take different action depending on value of variable.

```
public class Flip
{
   public static void main(String[] args)
   {
      if (Math.random() < 0.5)
           System.out.println("Heads");
      else System.out.println("Tails");
   }
}
```

```
% java Flip
Heads
% java Flip
Heads
% java Flip
Tails

% java Flip
Heads
```

If Statement Examples

```
if (x > 0) x = -x;
```

absolute value

```
if (x > y) max = x;
else       max = y;
```

maximum

```
if (x > y)
{
   int t = x;
   x = y;
   y = t;
}
```

2-sort

x > y before

x	y	t
1234	99	undefined
1234	99	1234
99	99	1234
99	1234	1234

x < y after

```
if (den == 0) System.out.println("Division by zero");
else          System.out.println("Quotient = " + num/den);
```

error check for division operation

```
double discriminant = b*b - 4.0*c;
if (discriminant < 0.0)
{
   System.out.println("No real roots");
}
else
{
   System.out.println((-b + Math.sqrt(discriminant))/2.0);
   System.out.println((-b - Math.sqrt(discriminant))/2.0);
}
```

error check for quadratic formula

Loops

While Loop

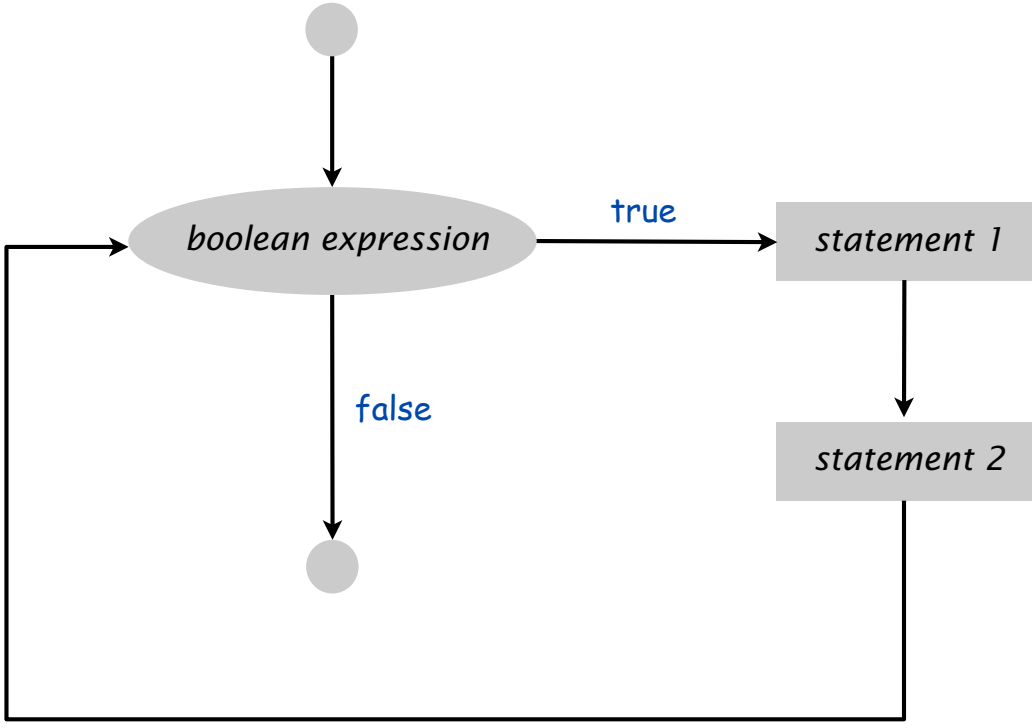
The `while` loop. A common repetition structure.

- Check a boolean expression.
- Execute a sequence of statements.
- Repeat.

loop continuation condition

```
while (boolean expression)
{
    statement 1;
    statement 2;
}
```

loop body



While Loop Example: Powers of Two

Ex. Print powers of 2 that are ≤ 2^n.

- Increment `i` from `0` to `n`.
- Double `v` each time.

```
int i = 0;
int v = 1;
while (i <= n)
{
   System.out.println(v);
   i = i + 1;
   v = 2 * v;
}
```

i	v	i <= n
0	1	true
1	2	true
2	4	true
3	8	true
4	16	true
5	32	true
6	64	true
7	128	false

n = 6

```
1
2
4
8
16
32
64
```

Powers of Two (full program)

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

      // last power of two to print
      int n = Integer.parseInt(args[0]);


      int i = 0;  // loop control counter
      int v = 1;  // current power of two
      while (i <= n)
      {
         System.out.println(v);
         i = i + 1;
         v = 2 * v;
      }
   }
}
```

print ith power of two

```
% java PowersOfTwo 4
1
2
4
8
% java PowersOfTwo 6
1
2
4
8
16
32
64
```

TEQ on While Loops

Anything wrong with the following code?

```
public class PowersOfTwo {
   public static void main(String[] args) {
      int N = Integer.parseInt(args[0]);
      int i = 0;  // loop control counter
      int v = 1;  // current power of two
      while (i <= N)
         System.out.println(v);
         i = i + 1;
         v = 2 * v;
   }
}
```

While Loop Example: Square Root

Goal. Implement `Math.sqrt()`.

```
% java Sqrt 60481729
7777.0
```

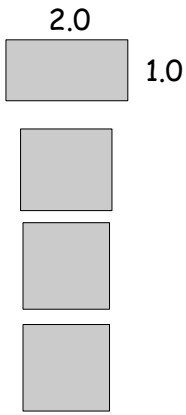
Newton-Raphson method to compute the square root of c:

- Initialize $t_0 = c$.
- Repeat until $t_i = c / t_i$, up to desired precision:

  set t_{i+1} to be the average of t_i and c / t_i.

i	t_i	$2/t_i$	average
0	2.0	1.0	1.5
1	1.5	1.3333333	1.4166667
2	1.4166667	1.4117647	1.4142157
3	1.4142157	1.4142114	1.4142136
4	1.4142136	1.4142136	

computing the square root of 2 to seven places



"A wonderful square root. Let's hope it can be used for the good of mankind."

While Loop Example: Square Root

Goal. Implement `Math.sqrt()`.

Newton-Raphson method to compute the square root of c:

- Initialize $t_0 = c$.
- Repeat until $t_i = c / t_i$, up to desired precision:

  set t_{i+1} to be the average of t_i and c / t_i.

```
public class Sqrt
{
   public static void main(String[] args)
   {
      double EPS = 1E-15;
      double c = Double.parseDouble(args[0]);
      double t = c;
      while (Math.abs(t - c/t) > t*EPS)
      {  t = (c/t + t) / 2.0;  }
      System.out.println(t);
   }
}
```

error tolerance

```
% java Sqrt 2.0
1.414213562373095
```

15 decimal digits of accuracy in 5 iterations

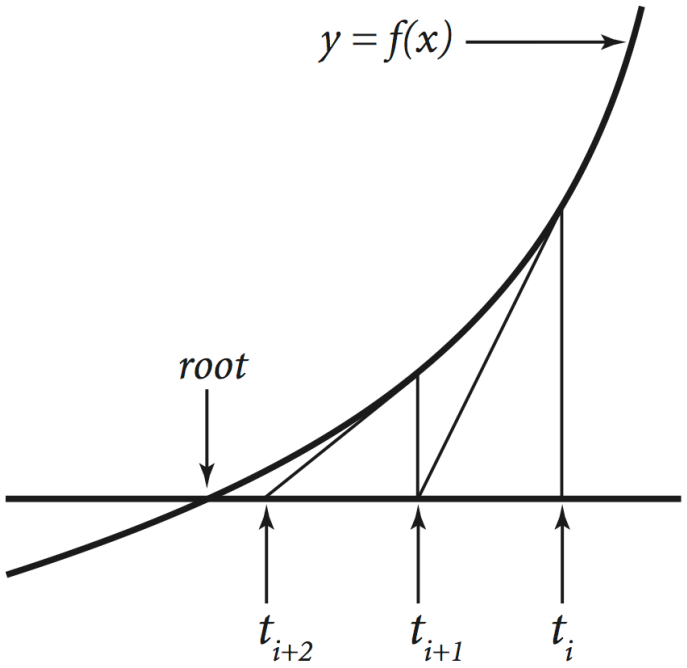
Newton-Raphson Method

Square root method explained (some math omitted).

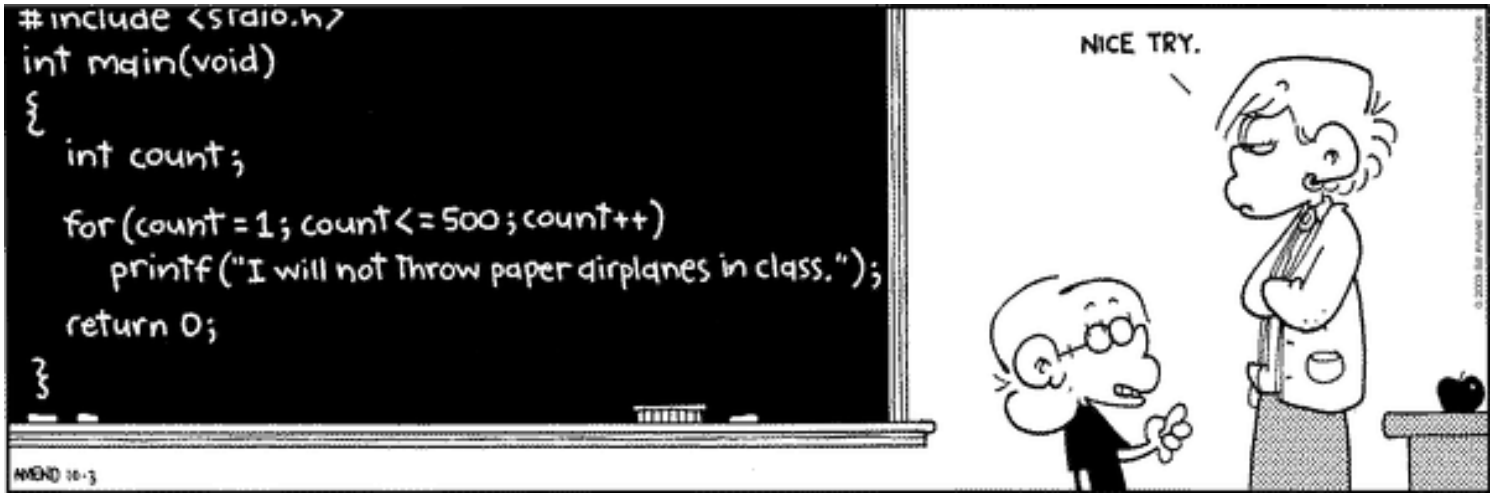
- Goal: find root of function f(x).
- Start with estimate t_0.

$f(x) = x^2 - c$ to compute $\sqrt{c}$

- Draw line tangent to curve at $x = t_i$.
- Set t_{i+1} to be x-coordinate where line hits x-axis.
- Repeat until desired precision.



The For Loop



Copyright 2004, FoxTrot by Bill Amend
www.ucomics.com/foxtrot/2003/10/03

The For Loop

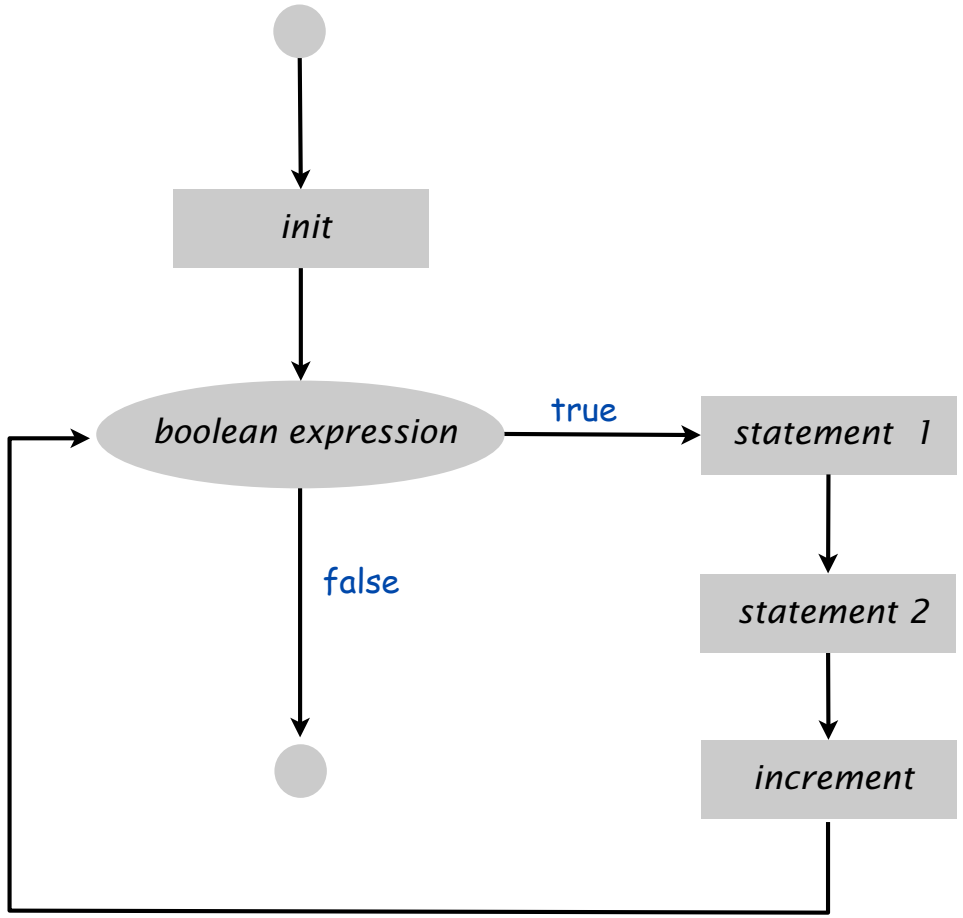
The `for` loop. Another common repetition structure.

- Execute initialization statement.
- Check boolean expression.
- Execute sequence of statements.
- Execute increment statement.
- Repeat.

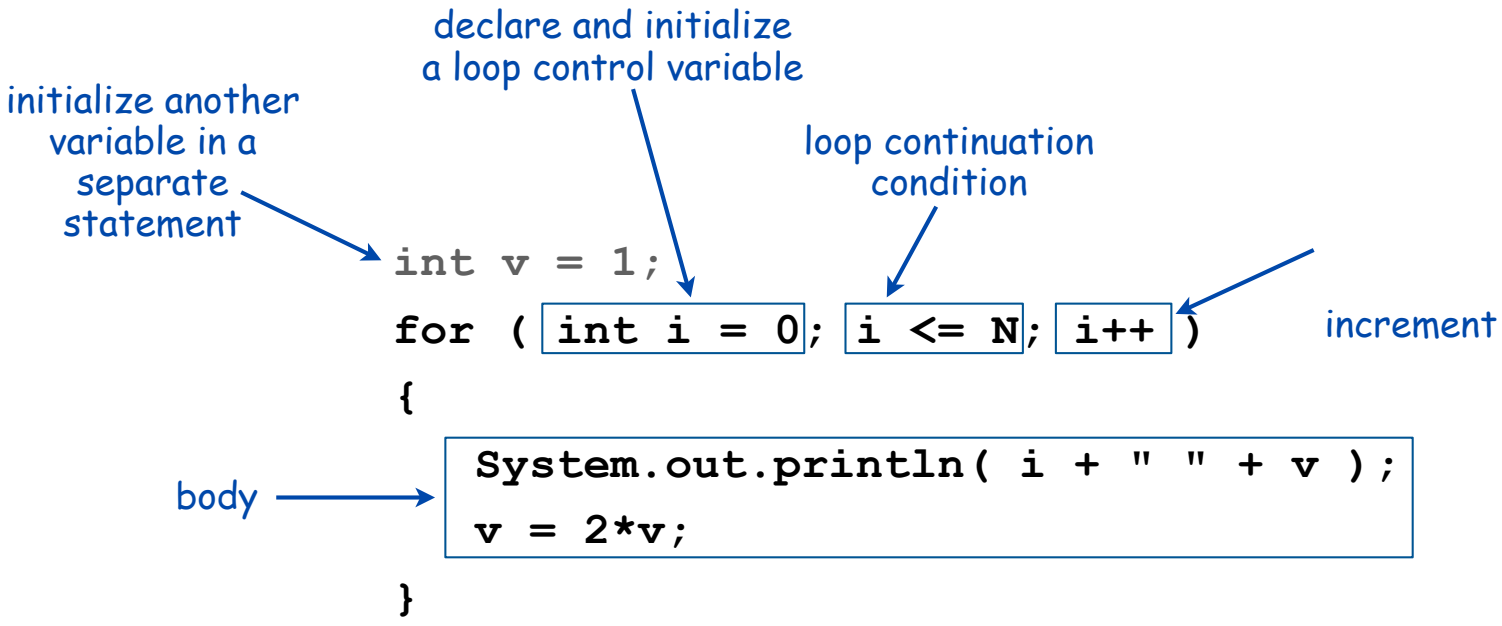
loop continuation condition

```
for (init; boolean expression; increment)
{
    statement 1;
    statement 2;
}
```

body



Anatomy of a `for` Loop



prints table of powers of two

Anatomy of a `for` Loop

```
int v = 1;
for ( int i = 0; i <= N; i++ )
{
   System.out.println( i + " " + v );
   v = 2*v;
}
```

v	i	output
1		
1	0	
1	0	0 1
2	0	
2	1	
2	1	1 2
4	1	
4	2	
4	2	2 4
8	2	
8	3	
8	3	3 8

Every `for` loop has an equivalent `while` loop

```
int v = 1;
int i = 0;
while ( i <= N; )
{
   System.out.println( i + " " + v );
   v = 2*v;
   i++;
}
```

Why `for` loops? Can provide more compact and understandable code.

For Loops: Subdivisions of a Ruler

Create subdivision of a ruler.

- Initialize `ruler` to empty string.
- For each value `i` from `1` to `N`:

  sandwich two copies of `ruler` on either side of `i`.

```
public class Ruler
{
   public static void main(String[] args)
   {
      int N = Integer.parseInt(args[0]);
      String ruler = " ";
      for (int i = 1; i <= N; i++)
        ruler = ruler + i + ruler;
      System.out.println(ruler);
   }
}
```

i	ruler
1	" 1 "
2	" 1 2 1 "
3	" 1 2 1 3 1 2 1 "

end-of-loop trace

For Loops: Subdivisions of a Ruler

```
% java Ruler 1
 1

% java Ruler 2
 1 2 1

% java Ruler 3
 1 2 1 3 1 2 1

% java Ruler 4
 1 2 1 3 1 2 1 4 1 2 1 3 1 2 1

% java Ruler 5
 1 2 1 3 1 2 1 4 1 2 1 3 1 2 1 5 1 2 1 3 1 2 1 4 1 2 1 3 1 2 1

% java Ruler 100
Exception in thread "main"
java.lang.OutOfMemoryError
```

$2^{100} - 1$ = 1,267,650,600,228,229,401,496,703,205,375 integers in output

Observation. Loops can produce a huge amount of output!

Loop Examples

```
int sum = 0;
for (int i = 1; i <= N; i++)
   sum += i;
System.out.println(sum);
```

compute sum (1 + 2 + 3 + . . . + N)

```
int sum = 0;
for (int i = 1; i <= N; i++)
   product += i;
System.out.println(product);
```

compute N! (1 * 2 * 3 * . . . * N)

```
for (int i = 0; i <= N; i++)
   System.out.println(i + " " + 2*Math.PI*i/N);
```

print a table of function values

```
int v = 1;
while (v <= N/2)
   v = 2*v;
System.out.println(v);
```

print largest power of 2 less than or equal to N

TEQ on For Loops

[easy if you read Exercise 1.3.13]

What does the following program print?

```
public class Mystery
{
   public static void main(String[] args)
   {
      int f = 0, g = 1;
      for (int i = 0; i <= 10; i++)
      {
         System.out.println(f);
         f = f + g;
         g = f - g;
      }
   }
}
```

Nesting

Nesting Conditionals and Loops

Nesting. Use a conditional or a loop within a conditional or a loop

- Enables complex control flows.
- Adds to challenge of debugging.

Any "statement" within a conditional or loop may itself be a conditional or a loop statement

```
for (int i = 0; i < trials; i++)
{
   int t = stake;
   while (t > 0 && t < goal)
      if (Math.random() < 0.5)  t++;
      else                      t--;
   if (t == goal) wins++;
}
```


if-else statement within a while loop within a for loop

Nested If Statements

Ex. Pay a certain tax rate depending on income level.

Income	Rate
0 - 47,450	22%
47,450 – 114,650	25%
114,650 – 174,700	28%
174,700 – 311,950	33%
311,950 -	35%

5 mutually exclusive alternatives

Nested If Statements

Use nested if statements to handle multiple alternatives

```
if (income <  47450) rate = 0.22;
else
   {
      if (income < 114650) rate = 0.25;
      else
         {
            if (income < 174700) rate = 0.28;
            else
               {
                  if (income < 311950) rate = 0.33;
                  else                 rate = 0.35;
               }
         }
    }
```

Nested If-Else Statements

Need all those braces? Not always:

```
if      (income <  47450) rate = 0.22;
else if (income < 114650) rate = 0.25;
else if (income < 174700) rate = 0.28;
else if (income < 311950) rate = 0.33;
else                      rate = 0.35;
```

is shorthand for

```
if (income <  47450) rate = 0.22;
else
   {
      if (income < 114650) rate = 0.25;
      else
         {
            if (income < 174700) rate = 0.28;
            else
               {
                  if (income < 311950) rate = 0.33;
                  else                 rate = 0.35;
               }
         }
    }
```

but BE CAREFUL when nesting if-else statements (see Q&A p. 75).

TEQ on If-Else

Anything wrong with the following code?

```
double rate = 0.35;
if (income <  47450) rate = 0.22;
if (income < 114650) rate = 0.25;
if (income < 174700) rate = 0.28;
if (income < 311950) rate = 0.33;
```

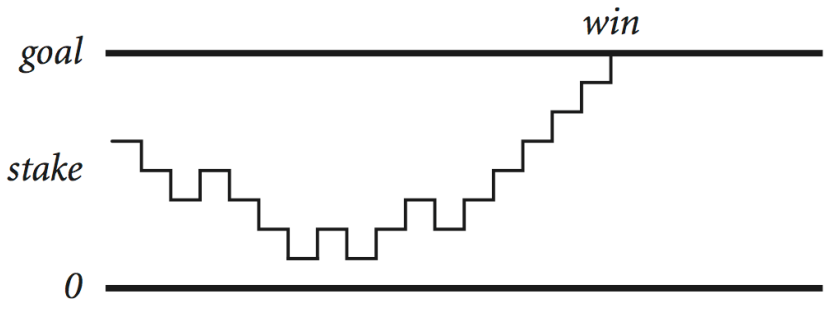
Nesting Example: Gambler's Ruin

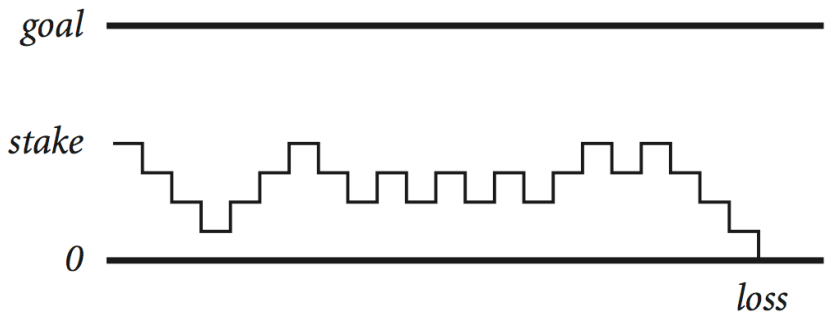
Gambler's ruin. Gambler starts with $stake and places $1 fair bets until going broke or reaching $goal.

- What are the chances of winning?
- How many bets will it take?

One approach. Monte Carlo simulation.

- Flip digital coins and see what happens.
- Repeat and compute statistics.





Nesting Example: Gambler's Ruin Simulation

```
public class Gambler
{
    public static void main(String[] args)
    {
      // Get parameters from command line.
      int stake  = Integer.parseInt(args[0]);
      int goal   = Integer.parseInt(args[1]);
      int trials = Integer.parseInt(args[2]);

      // Count wins among args[2] trials.
      int wins   = 0;
      for (int i = 0; i < trials; i++)
      {
         // Do one gambler's ruin experiment.
         int t = stake;
         while (t > 0 && t < goal)
         {
            // flip coin and update
            if (Math.random() < 0.5) t++;
            else                     t--;
         }
         if (t == goal) wins++;
      }
      System.out.println(wins + " wins of " + trials);
   }
}
```

if statement
within a while loop
within a for loop

Digression: Simulation and Analysis

stake goal trials

```
% java Gambler 5 25 1000
191 wins of 1000

% java Gambler 5 25 1000
203 wins of 1000

% java Gambler 500 2500 1000
197 wins of 1000
```

after a substantial wait….

Fact. Probability of winning = stake ÷ goal.

Fact. Expected number of bets = stake × desired gain.

Ex. 20% chance of turning $500 into $2500, 500/2500 = 20%

but expect to make one million $1 bets. 500*(2500 - 500) = 1,000,000

Remark. Both facts can be proved mathematically.

For more complex scenarios, computer simulation is often the best plan of attack.

Control Flow Summary

Control flow.

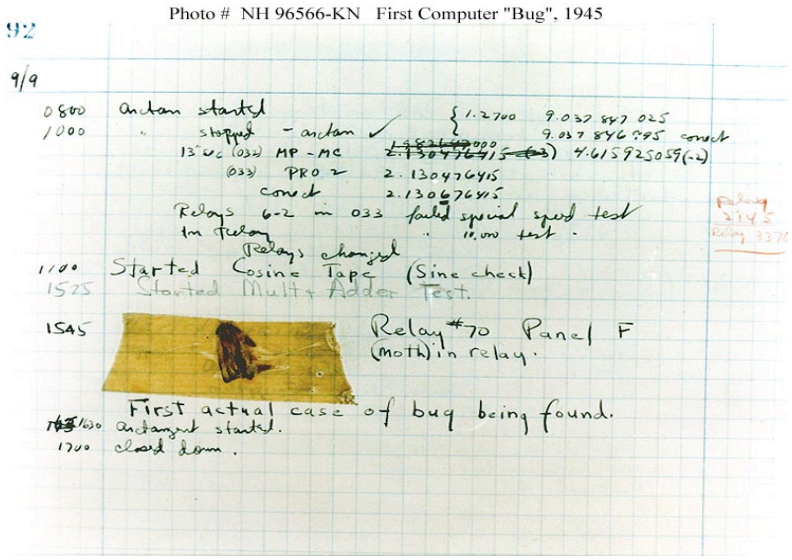
- Sequence of statements that are actually executed in a program.
- Conditionals and loops: enables us to choreograph the control flow.

Control Flow	Description	Examples
Straight-line programs	All statements are executed in the order given.	
Conditionals	Certain statements are executed depending on the values of certain variables.	`if` `if-else`

Loops	Certain statements are executed repeatedly until certain conditions are met.	`while`
		`for`
		`do-while`

Debugging

Admiral Grace Murray Hopper



http://www.history.navy.mil/photos/images/h96000/h96566kc.htm

99% of program development

Debugging. Cyclic process of editing, compiling, and fixing errors.

- Always a logical explanation.
- What would the machine do?
- Explain it to the teddy bear.

You will make many mistakes as you write programs. It's normal.

Good news: Can use computer to test program.

Bad news: Conditionals/loops open up huge number of possibilities.

Really bad news: Cannot use computer to automatically find all bugs.

stay tuned

Debugging Example

Factor. Given an integer $N > 1$, compute its prime factorization.

$3{,}757{,}208 = 2^3 \times 7 \times 13^2 \times 397$

$98 = 2 \times 7^2$

$17 = 17$

$11{,}111{,}111{,}111{,}111 = 2{,}071{,}723 \times 5{,}363{,}222{,}357$

Note: 1 is not prime. (else it would have to be in every factorization)

Application. Break RSA cryptosystem (factor 200-digit numbers).

Debugging: 99% of Program Development

Programming. A process of finding and fixing mistakes.

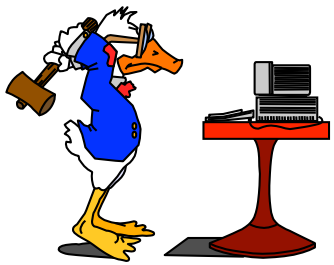
- Compiler error messages help locate syntax errors.
- Run program to find semantic and performance errors.

```
public class Factors
{
   public static void main(String[] args)
   {
      long N = Long.parseLong(args[0])
      for (i = 0; i < N; i++)
      {
         while (N % i == 0)
            System.out.print(i + " ")
            N = N / i
      }
   }
}
```

Check whether i is a factor.

if i is a factor print it and divide it out

This program has bugs!



Debugging: Syntax Errors

Syntax error. Illegal Java program.

- Compiler error messages help locate problem.
- Goal: no errors and a file named `Factors.class`.

```
public class Factors
{
   public static void main(String[] args)
   {
      long N = Long.parseLong(args[0])
      for (i = 0; i < N; i++)
      {
         while (N % i == 0)
            System.out.print(i + " ")
            N = N / i
      }
   }
}
```

Debugging: Syntax Errors

Syntax error. Illegal Java program.

- Compiler error messages help locate problem.
- Goal: no errors and a file named `Factors.class`.

```
public class Factors
{
   public static void main(String[] args)
   {
      long N = Long.parseLong(args[0])
      for (i = 0; i < N; i++)
      {
         while (N % i == 0)
            System.out.print(i + " ")
            N = N / i
      }
   }
}
```

```
% javac Factors.java
Factors.java:6: ';' expected
      for (i = 2; i < N; i++)
      ^
1 error  <-- the FIRST error
```

Debugging: Syntax Errors

Syntax error. Illegal Java program.

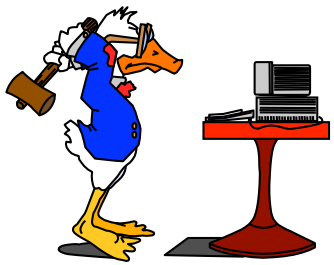
- Compiler error messages help locate problem.
- Goal: no errors and a file named `Factors.class`.

```
public class Factors
{
   public static void main(String[] args)
   {
      long N = Long.parseLong(args[0]) ;
      for ( int i = 0; i < N; i++ )
      {
         while (N % i == 0)
            System.out.print(i + " ") ;
            N = N / i ;
      }
   }
}
```

need to declare variable i

need terminating semicolons

Syntax (compile-time) errors



Debugging: Semantic Errors

Semantic error. Legal but wrong Java program.

- Run program to identify problem.
- Add print statements if needed to produce trace.

```
public class Factors
{
   public static void main(String[] args)
   {
      long N = Long.parseLong(args[0]);
      for (int i = 0; i < N; i++)
      {
         while (N % i == 0)
            System.out.print(i + " ");
            N = N / i;
      }
   }
}
```

```
% javac Factors.java
% java Factors          <-- oops, need argument
Exception in thread "main"
java.lang.ArrayIndexOutOfBoundsException: 0
        at Factors.main(Factors.java:5)
```

you will see this message!

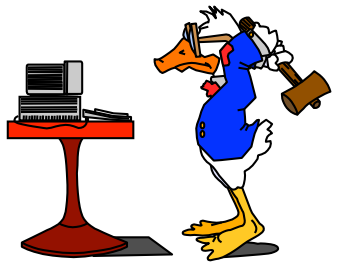
Debugging: Semantic Errors

Semantic error. Legal but wrong Java program.

- Run program to identify problem.
- Add print statements if needed.

```
public class Factors
{
   public static void main(String[] args)
   {
      long N = Long.parseLong(args[0]);
      for (int i = 0; i < N; i++)
      {
         while (N % i == 0)
            System.out.print(i + " ");
            N = N / i;
      }
   }
}
```

```
% javac Factors.java
% % java Factors 98
Exception in thread "main"
java.lang.ArithmeticException: / by zero
         at Factors.main(Factors.java:8)
```



Debugging: Semantic Errors

Semantic error. Legal but wrong Java program.

- Run program to identify problem.
- Add print statements if needed.

```
public class Factors
{
   public static void main(String[] args)
   {
      long N = Long.parseLong(args[0]);
      for (int i = 2; i < N; i++)
      {
         while (N % i == 0)
            System.out.print(i + " ");
            N = N / i;
      }
   }
}
```

need to start at 2 since
0 and 1 cannot be factors

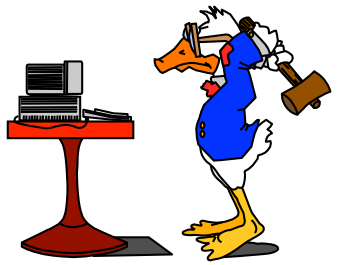
Debugging: Semantic Errors

Semantic error. Legal but wrong Java program.

- Run program to identify problem.
- Add print statements if needed.

```
public class Factors
{
   public static void main(String[] args)
   {
      long N = Long.parseLong(args[0]);
      for (int i = 2; i < N; i++)
      {
         while (N % i == 0)
            System.out.print(i + " ");
            N = N / i;
      }
   }
}
```

```
% javac Factors.java
% java Factors 98
2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
??? infinite loop
```



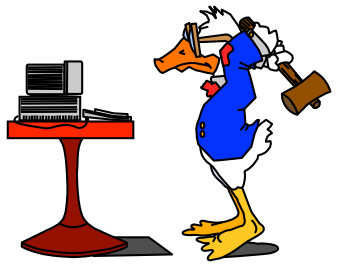
Debugging: Semantic Errors

Semantic error. Legal but wrong Java program.

- Run program to identify problem.
- Add print statements if needed.

```
public class Factors
{
   public static void main(String[] args)
   {
      long N = Long.parseLong(args[0])
      for (int i = 2; i < N; i++)
      {
         while (N % i == 0)
          { System.out.print(i + " ");
            N = N / i;  }
      }
   }
}
```

Semantic (run-time) error:
indents do not imply braces



Debugging: The Beat Goes On

Success? Program factors 98 = 2 2 7.

- Time to try it for other inputs.
- Add trace to find and fix (minor) problems.

```
public class Factors
{
   public static void main(String[] args)
   {
      long N = Long.parseLong(args[0])
      for (int i = 2; i < N; i++)
      {  // Check whether i is a factor.
         while (N % i == 0)
         {  // If so, print and divide.
            System.out.print(i + " ");
            N = N / i;
         }
      }
   }
}
```

```
% java Factors 98
2 7 7 %        <-- need newline
% java Factors 5
%              <-- ??? no output
% java Factors 6
2 %            <-- ??? where's the 3?
```

Debugging: The Beat Goes On

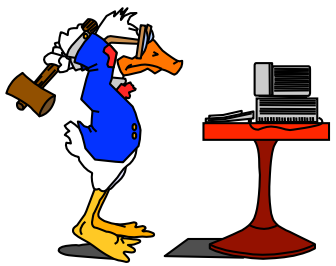
Success? Program factors 98 = 2 2 7.

- Time to try it for other inputs.
- Add trace to find and fix (minor) problems.

```
public class Factors
{
   public static void main(String[] args)
   {
      long N = Long.parseLong(args[0])
      for (int i = 2; i < N; i++)
      {
         while (N % i == 0)
         {
            System.out.println(i + " ");
            N = N / i;
         }
         System.out.println("TRACE " + i + " " + N);
      }
   }
}
```

```
% javac Factors.java
% java Factors 5
TRACE 2 5
TRACE 3 5
TRACE 4 5
% java Factors 6
2
TRACE 2 3
```

AHA!
Print out N
after for loop
(if it is not 1)



Debugging: Success?

Success? Program seems to work.

- Remove trace to try larger inputs.
- [stay tuned].

```
public class Factors
{
   public static void main(String[] args)
   {
      long N = Long.parseLong(args[0])
      for (int i = 2; i < N; i++)
      {  // Check whether i is a factor.
         while (N % i == 0)
         {  // If so, print and divide.
            System.out.print(i + " ");
            N = N / i;
         }
      }
      if (N > 1) System.out.println(N);
      else       System.out.println();
   }
}
```

```
% java Factors 5
TRACE 2 5
TRACE 3 5
TRACE 4 5
% javac Factors.java
% java Factors 5
5
% java Factors 6
2 3
% java Factors 98
2 7 7
% java Factors 3757208
2 2 2 7 13 13 397
```

???
%$%@$#!!
forgot to recompile

Corner case: print largest factor (and new line)

Time to add comments (if not earlier).

Debugging Your Program

Debugging Your Program. [summary]

1. Create the program.

2. Compile it.

   Compiler says: That's not a legal program.

   Back to step 1 to fix your errors of syntax.

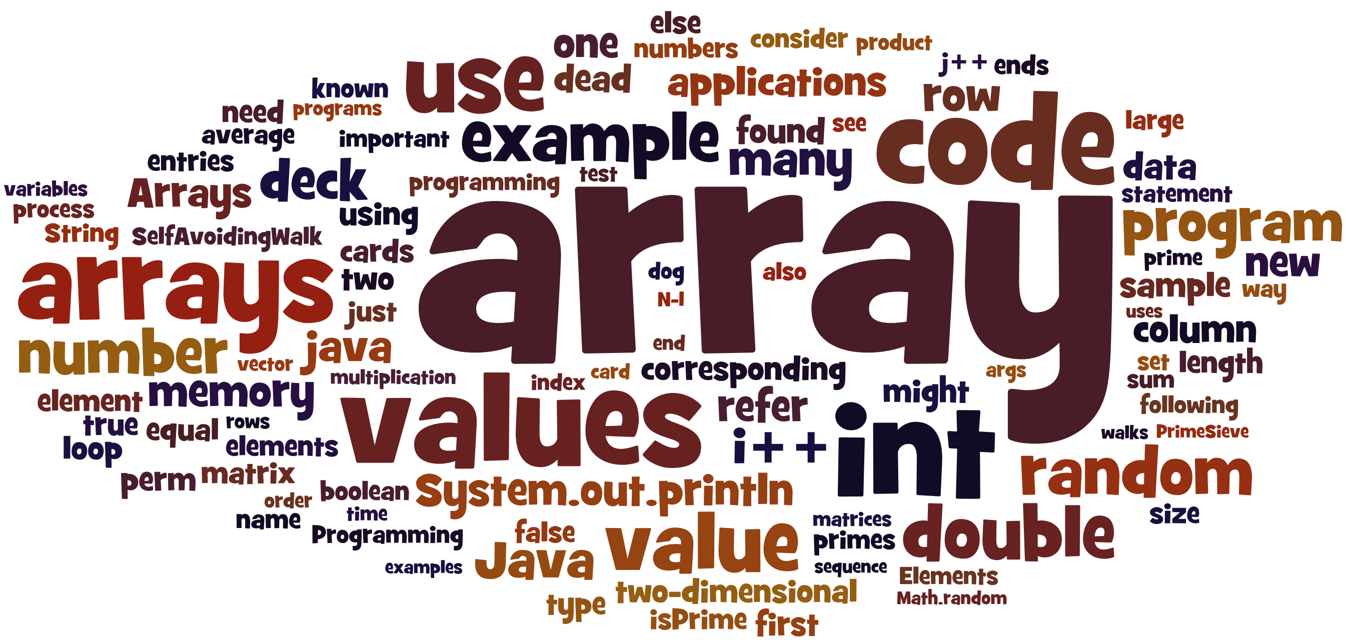
3. Execute it.

   Result is bizarrely (or subtly) wrong.

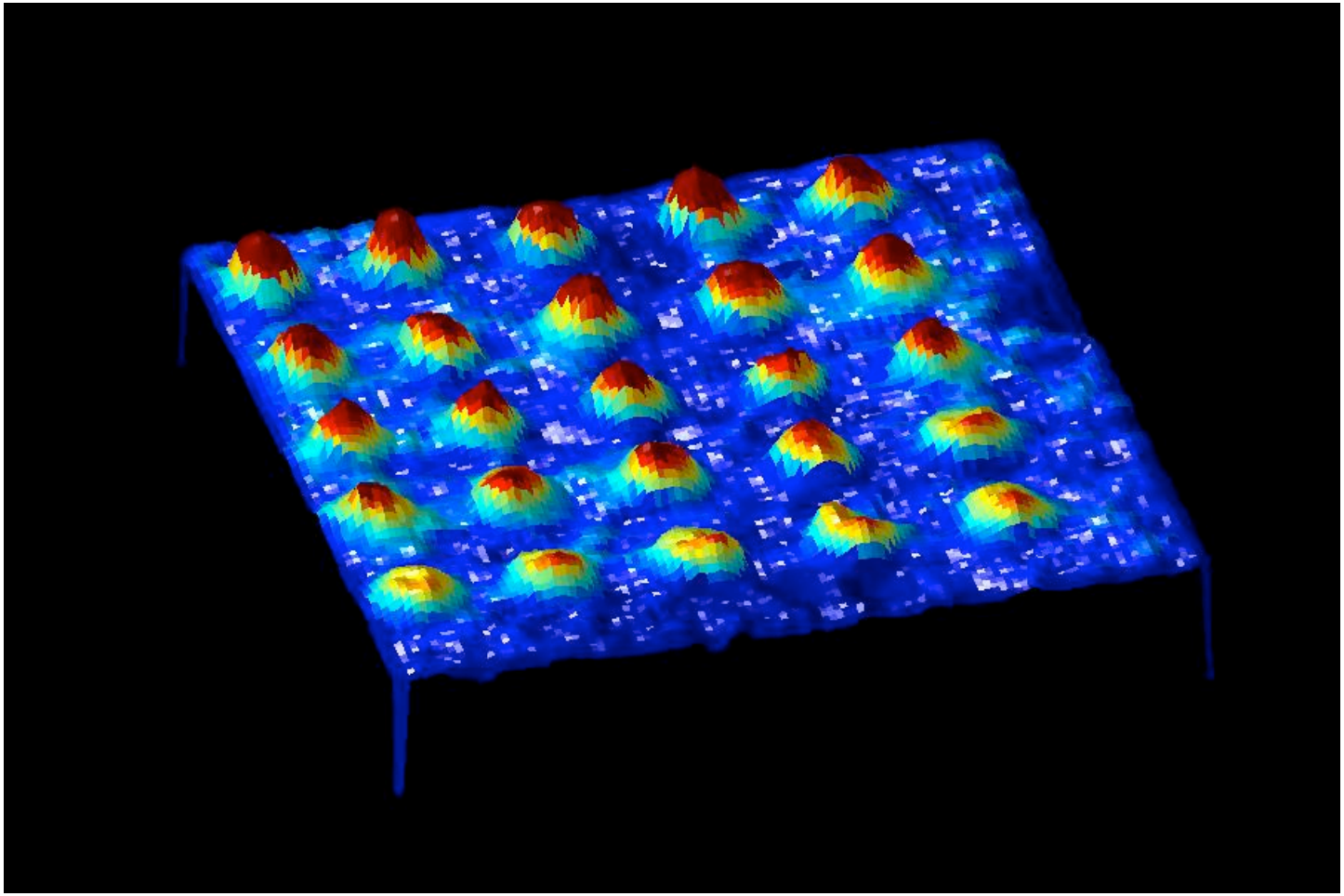
   Back to step 1 to fix your errors of semantics.

4. Enjoy the satisfaction of a working program!

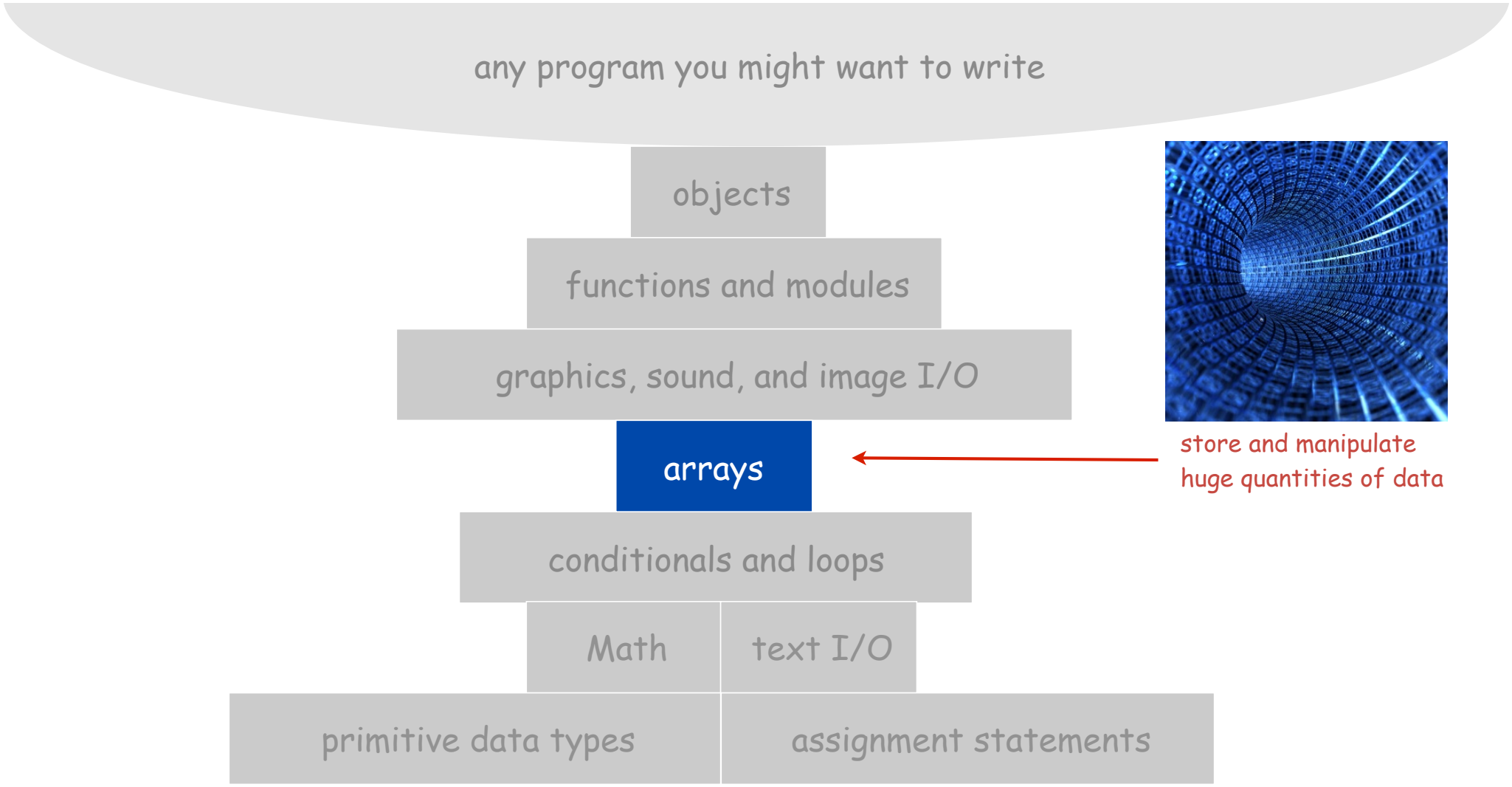
   [but stay tuned for more debugging]


else
one
numbers
consider
product
use
dead
applications
j++
ends
known
programs
row
need
code
found
see
large
average
important
example
many
entries
data
deck
test
variables
Arrays
programming
statement
process
using
program
String
SelfAvoidingWalk
array
cards
prime
new
arrays
two
dog
also
sample
way
just
N-1
uses
column
number
java
end
vector
multiplication
index
card
corresponding
args
set
length
element
memory
sum
might
refer
following
true
rows
values
equal
elements
i++
int
walks
PrimeSieve
loop
random
perm
matrix
order
boolean
System.out.println
size
time
name
matrices
false
double
Programming
value
primes
Java
examples
sequence
Elements
two-dimensional
type
Math.random
isPrime
first

1.4 Arrays



A Foundation for Programming



Arrays

This lecture. Store and manipulate huge quantities of data.

Array. Indexed sequence of values of the same type.

Examples.

- 52 playing cards in a deck.
- 5 thousand undergrads at Princeton.
- 1 million characters in a book.
- 10 million audio samples in an MP3 file.
- 4 billion nucleotides in a DNA strand.
- 73 billion Google queries per year.
- 50 trillion cells in the human body.
- 6.02×10^{23} particles in a mole.

index	value
0	wayne
1	doug
2	rs
3	maia
4	mona
5	cbienia
6	wkj
7	mkc

Many Variables of the Same Type

Goal. 10 variables of the same type.

```
// Tedious and error-prone code.
double a0, a1, a2, a3, a4, a5, a6, a7, a8, a9;
a0 = 0.0;
a1 = 0.0;
a2 = 0.0;
a3 = 0.0;
a4 = 0.0;
a5 = 0.0;
a6 = 0.0;
a7 = 0.0;
a8 = 0.0;
a9 = 0.0;
...
a4 = 3.0;
...
a8 = 8.0;
...
double x = a4 + a8;
```

Many Variables of the Same Type

Goal. 10 variables of the same type.

```
// Easy alternative.
double[] a = new double[10];
...
a[4] = 3.0;
...
a[8] = 8.0;
...
double x = a[4] + a[8];
```

declares, creates, and initializes
[stay tuned for details]

Many Variables of the Same Type

Goal. 1 million variables of the same type.

```
// Scales to handle large arrays.
double[] a = new double[1000000];
...
a[234567] = 3.0;
...
a[876543] = 8.0;
...
double x = a[234567] + a[876543];
```

Arrays in Java

Java has special language support for arrays.

- To make an array: declare, create, and initialize it.
- To access element `i` of array named `a`, use `a[i]`.
- Array indices start at `0`.

```
int N = 1000;
double[] a;                      // declare the array
a = new double[N];               // create the array
for (int i = 0; i < N; i++)      // initialize the array
   a[i] = 0.0;                   // all to 0.0
```

Arrays in Java

Java has special language support for arrays.

- To make an array: declare, create, and initialize it.
- To access element `i` of array named `a`, use `a[i]`.
- Array indices start at `0`.

```
int N = 1000;
double[] a;                      // declare the array
a = new double[N];               // create the array
for (int i = 0; i < N; i++)      // initialize the array
   a[i] = 0.0;                   // all to 0.0
```

Compact alternative: Declare, create, and initialize in one statement.

- Default: all entries automatically set to 0.

```
int N = 1000;
double[] a = new double[N];
```

- Alternative: entries initialized to given literal values.

```
double[] x = { 0.3, 0.6, 0.1 };
```

Sample Array Code: Vector Dot Product

Dot product. Given two vectors `x[]` and `y[]` of length `N`, their dot product is the sum of the products of their corresponding components.

```
double[] x = { 0.3, 0.6, 0.1 };
double[] y = { 0.5, 0.1, 0.4 };

double sum = 0.0;
for (int i = 0; i < N; i++)
   sum += x[i]*y[i];
```

i	x[i]	y[i]	x[i]*y[i]	sum
				0
0	.30	.50	.15	.15
1	.60	.10	.06	.21
2	.10	.40	.04	.25
				.25

Array Processing Examples

```
double[] a = new double[N];
for (int i = 0; i < N; i++)
   a[i] = Math.random();
```

create an array with N random values

```
double[] b = new double[N];
for (int i = 0; i < N; i++)
   b[i] = a[i];
```

copy to another array

```
for (int i = 0; i < N; i++)
   System.out.println(a[i]);
```

print the array values, one per line

```
double sum = 0.0;
for (int i = 0; i < N; i++)
   sum += a[i];
double average = sum / N;
```

compute the average of the array values

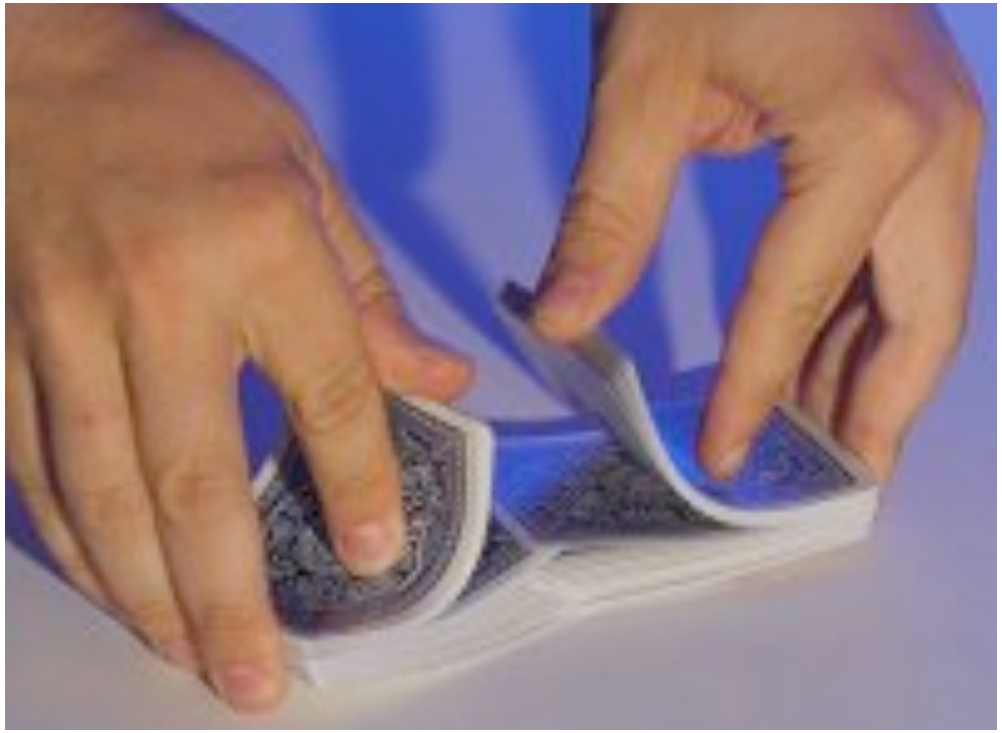
```
double max = Double.NEGATIVE_INFINITY;
for (int i = 0; i < N; i++)
   if (a[i] > max) max = a[i];
```

find the maximum of the array values

```
for (int i = 0; i < N/2; i++)
{
   double temp = b[i];
   b[i] = b[N-1-i];
   b[N-i-1] = temp;
}
```

reverse the elements within the array

Shuffling a Deck



Setting Array Values at Compile Time

Ex. Print a random card.

```
String[] rank =
{
   "2", "3", "4", "5", "6", "7", "8", "9",
   "10", "Jack", "Queen", "King", "Ace"
};


String[] suit =
{
   "Clubs", "Diamonds", "Hearts", "Spades"
};

int i = (int) (Math.random() * 13);  // between 0 and 12
int j = (int) (Math.random() * 4);   // between 0 and 3

System.out.println(rank[i] + " of " + suit[j]);
```

TEQ on Arrays 1

The following code sets array values to the 52 card values and prints them.

What order are they printed?

```
String[] deck = new String[52];
for (int i = 0; i < 13; i++)
   for (int j = 0; j < 4; j++)
      deck[4*i + j] = rank[i] + " of " + suit[j];

for (int i = 0; i < 52; i++)
   System.out.println(deck[i]);
```

typical array processing code changes values at runtime

A.
```
2 of clubs
2 of diamonds
2 of hearts
2 of spades
3 of clubs
...
```

B.
```
2 of clubs
3 of clubs
4 of clubs
5 of clubs
6 of clubs
...
```

Shuffling

Goal. Given an array, rearrange its elements in random order.

Shuffling algorithm.

- In iteration `i`, pick random card from `deck[i]` through `deck[N-1]`, with each card equally likely.
- Exchange it with `deck[i]`.

```
int N = deck.length;
for (int i = 0; i < N; i++)
{
   int r = i + (int) (Math.random() * (N-i));
   String t = deck[r];
   deck[r] = deck[i];
   deck[i] = t;
}
```

swap idiom

between i and N-1

Shuffling a Deck of Cards

```
public class Deck
{
   public static void main(String[] args)
   {
      String[] suit = { "Clubs", "Diamonds", "Hearts", "Spades" };
      String[] rank = { "2", "3", "4", "5", "6", "7", "8", "9",
                        "10", "Jack", "Queen", "King", "Ace"    };
      int SUITS = suit.length;
      int RANKS = rank.length;
      int N = SUITS * RANKS;

      String[] deck = new String[N];
      for (int i = 0; i < RANKS; i++)
         for (int j = 0; j < SUITS; j++)
            deck[SUITS*i + j] = rank[i] + " of " + suit[j];

      for (int i = 0; i < N; i++)
      {
         int r = i + (int) (Math.random() * (N-i));
         String t = deck[r];
         deck[r] = deck[i];
         deck[i] = t;
      }

      for (int i = 0; i < N; i++)
         System.out.println(deck[i]);
   }
}
```

avoid "hardwired" constants like 52, 4, and 13.

build the deck

shuffle

print shuffled deck

Shuffling a Deck of Cards

```
% java Deck
5 of Clubs
Jack of Hearts
9 of Spades
10 of Spades
9 of Clubs
7 of Spades
6 of Diamonds
7 of Hearts
7 of Clubs
4 of Spades
Queen of Diamonds
10 of Hearts
5 of Diamonds
Jack of Clubs
Ace of Hearts
...
5 of Spades
```

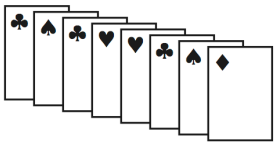
```
% java Deck
10 of Diamonds
King of Spades
2 of Spades
3 of Clubs
4 of Spades
Queen of Clubs
2 of Hearts
7 of Diamonds
6 of Spades
Queen of Spades
3 of Spades
Jack of Diamonds
6 of Diamonds
8 of Spades
9 of Diamonds
...
10 of Spades
```

Coupon Collector

Coupon Collector Problem

Coupon collector problem. Given `N` different card types, how many do you have to collect before you have (at least) one of each type?

assuming each possibility is equally likely for each card that you collect



Simulation algorithm. Repeatedly choose an integer `i` between `0` and `N-1`. Stop when we have at least one card of every type.

Q. How to check if we've seen a card of type `i`?

A. Maintain a boolean array so that `found[i]` is `true` if we've already collected a card of type `i`.

Coupon Collector: Java Implementation

```
public class CouponCollector
{
   public static void main(String[] args)
   {
      int N = Integer.parseInt(args[0]);
      int cardcnt = 0;   // number of cards collected
      int valcnt = 0;    // number of distinct cards

      // Do simulation.
      boolean[] found = new boolean[N];
      while (valcnt < N)
      {
         int val = (int) (Math.random() * N);
         cardcnt++;
         if (!found[val])
         {
            valcnt++;
            found[val] = true;
         }
      }

      // all N distinct cards found
      System.out.println(cardcnt);
   }
}
```

type of next card (between 0 and N-1)

Coupon Collector: Debugging

Debugging. Add code to print contents of all variables.

val	found						valcnt	cardcnt
	0	1	2	3	4	5		
	F	F	F	F	F	F	0	0
2	F	F	T	F	F	F	1	1
0	T	F	T	F	F	F	2	2
4	T	F	T	F	T	F	3	3
0	T	F	T	F	T	F	3	4
1	T	T	T	F	T	F	4	5
2	T	T	T	F	T	F	4	6
5	T	T	T	F	T	T	5	7
0	T	T	T	F	T	T	5	8
1	T	T	T	F	T	T	5	9
3	T	T	T	T	T	T	6	10

Challenge. Debugging with arrays requires tracing many variables.

Coupon Collector: Mathematical Context

Coupon collector problem. Given N different possible cards, how many do you have to collect before you have (at least) one of each type?

Fact. About $N (1 + 1/2 + 1/3 + \ldots + 1/N) \sim N \ln N$

see ORF 245 or COS 341

Ex. N = 30 baseball teams. Expect to wait ≈ 120 years before all teams win a World Series.

under idealized assumptions

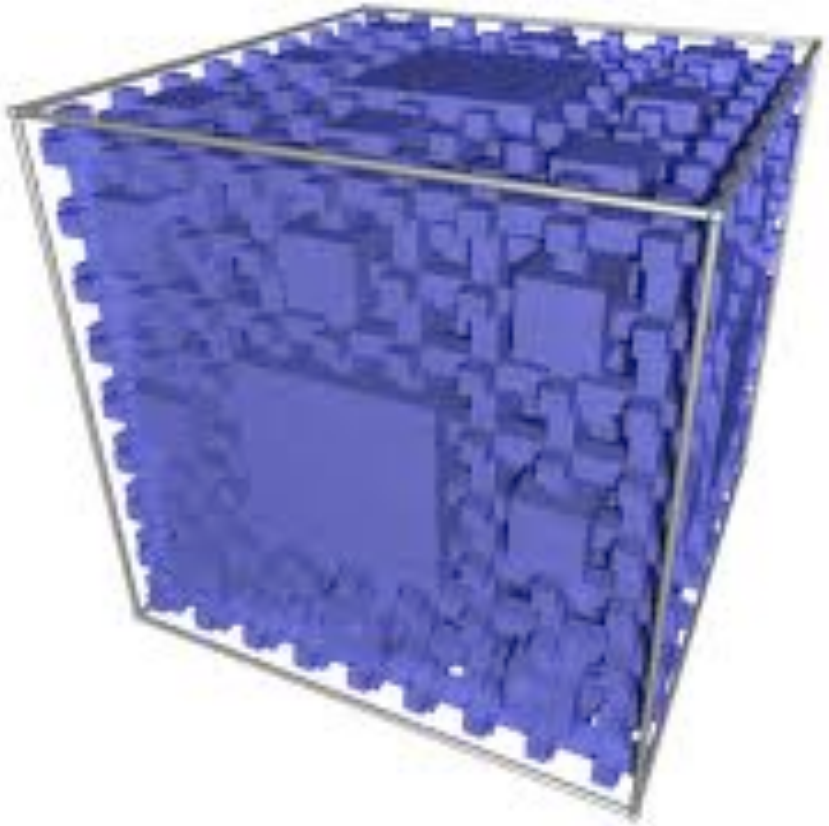
Coupon Collector: Scientific Context

Q. Given a sequence from nature, does it have same characteristics as a random sequence?

A. No easy answer - many tests have been developed.

Coupon collector test. Compare number of elements that need to be examined before all values are found against the corresponding answer for a random sequence.

Multidimensional Arrays



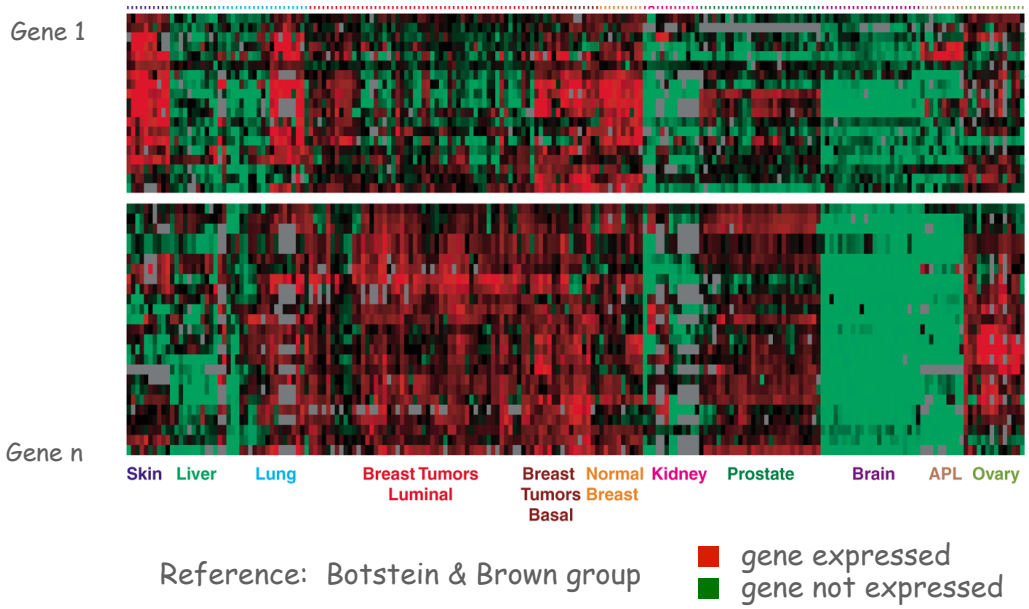
Two Dimensional Arrays

Two dimensional arrays.

- Table of data for each experiment and outcome.
- Table of grades for each student and assignments.
- Table of grayscale values for each pixel in a 2D image.

Mathematical abstraction. Matrix.

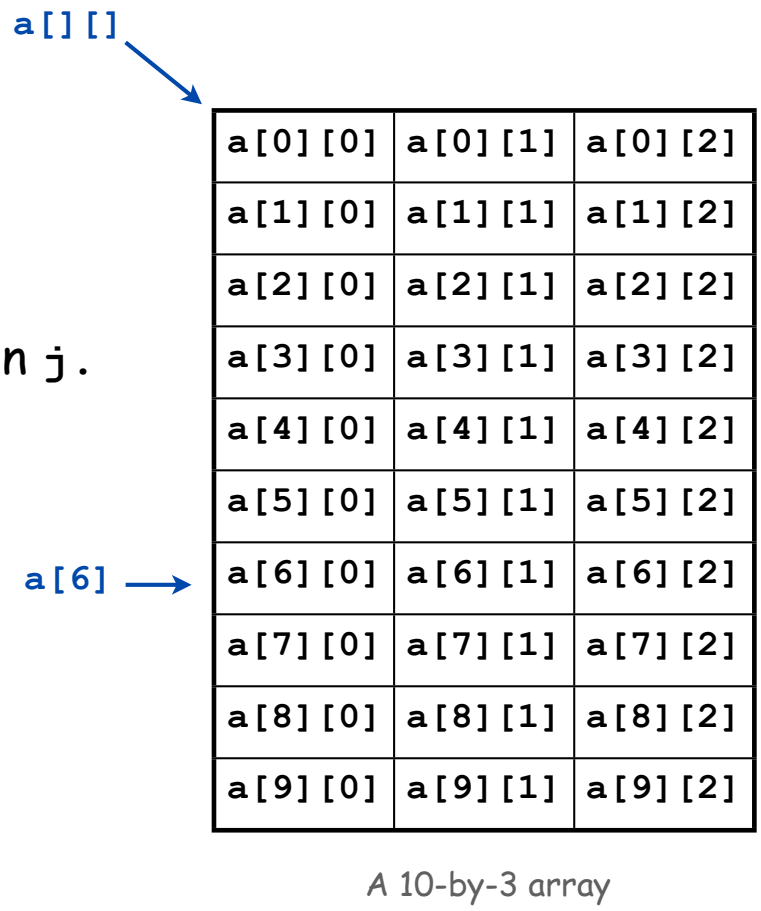
Java abstraction. 2D array.



Two Dimensional Arrays in Java

Declare, create, initialize. Like 1D, but add another pair of brackets.

```
int M = 10;
int N = 3;
double[][] a = new double[M][N];
```



A 10-by-3 array

Array access.

Use `a[i][j]` to access entry in row `i` and column `j`.

Indices start at `0`.

Initialize.

This code is implicit (sets all entries to 0).

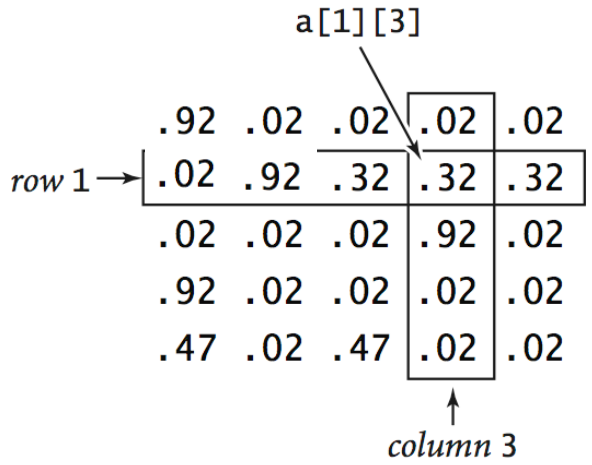
```
for (int i = 0; i < M; i++)
   for (int j = 0; j < N; j++)
      a[i][j] = 0.0;
```

Warning. This implicit code might slow down your program for big arrays.

Setting 2D Array Values at Compile Time

Initialize 2D array by listing values.

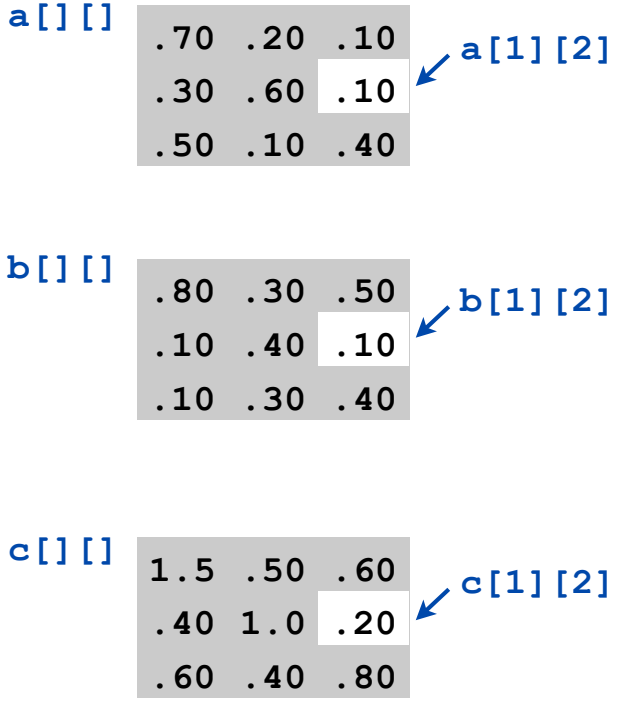
```
double[][] p =
{
   { .02, .92, .02, .02, .02 },
   { .02, .02, .32, .32, .32 },
   { .02, .02, .02, .92, .02 },
   { .92, .02, .02, .02, .02 },
   { .47, .02, .47, .02, .02 },
};
```



Matrix Addition

Matrix addition. Given two N-by-N matrices `a` and `b`, define `c` to be the N-by-N matrix where `c[i][j]` is the sum `a[i][j] + b[i][j]`.

```
double[][] c = new double[N][N];
for (int i = 0; i < N; i++)
   for (int j = 0; j < N; j++)
      c[i][j] = a[i][j] + b[i][j];
```

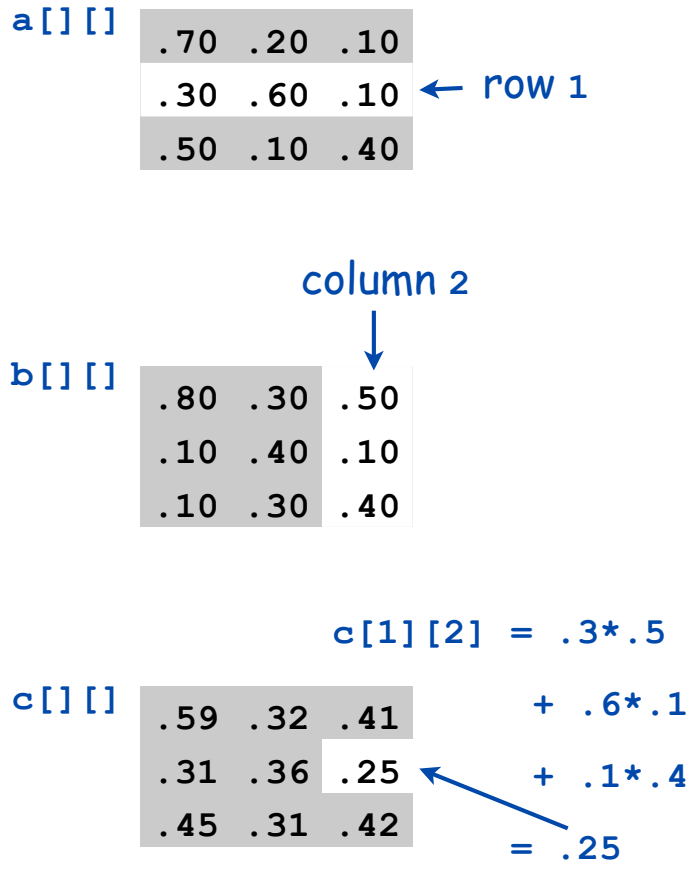


Matrix Multiplication

Matrix multiplication. Given two N-by-N matrices `a` and `b`, define `c` to be the N-by-N matrix where `c[i][j]` is the dot product of the i^{th} row of `a` and the j^{th} row of `b`.

all values initialized to 0

```
double[][] c = new double[N][N];
for (int i = 0; i < N; i++)
   for (int j = 0; j < N; j++)
      for (int k = 0; k < N; k++)
         c[i][j] += a[i][k] * b[k][j];
```



TEQ on Arrays 2

How many multiplications to multiply two N-by-N matrices?

```
double[][] c = new double[N][N];
for (int i = 0; i < N; i++)
   for (int j = 0; j < N; j++)
      for (int k = 0; k < N; k++)
         c[i][j] += a[i][k] * b[k][j];
```

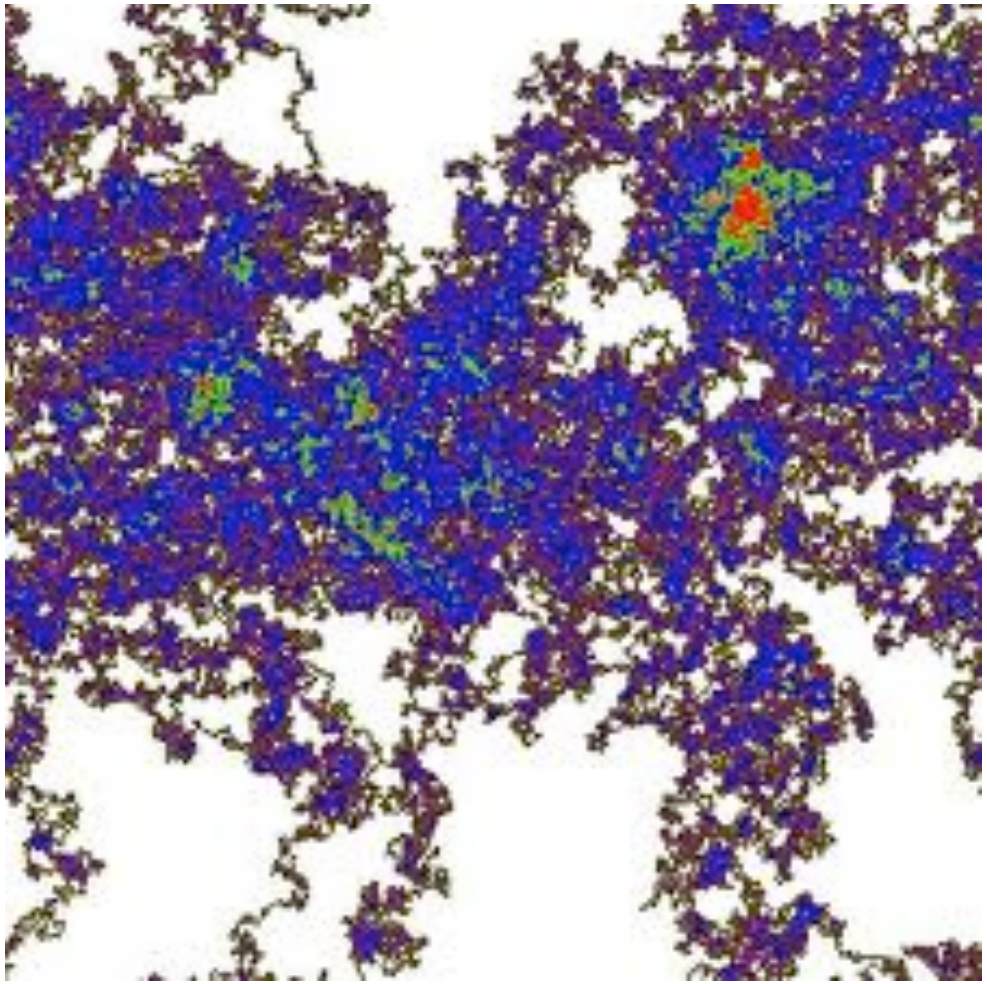
A. N

B. N^2

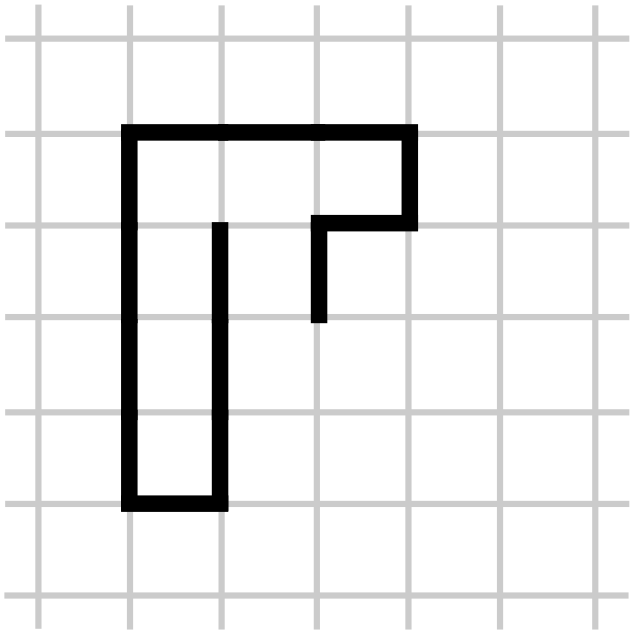
C. N^3

D. N^4

Application: 2D Random Walks



Application: Self-Avoiding Walks

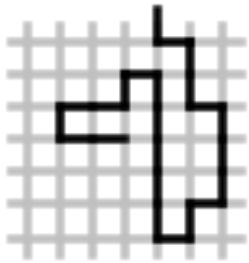


Self-Avoiding Walk

Model.

- N-by-N lattice.
- Start in the middle.
- Randomly move to a neighboring intersection, avoiding all previous intersections.
- Two possible outcomes: escape and dead end

dead end



escape

Applications. Polymers, statistical mechanics, etc.

Q. What fraction of time will you escape in an 5-by-5 lattice?

Q. In an N-by-N lattice?

Q. In an N-by-N-by-N lattice?

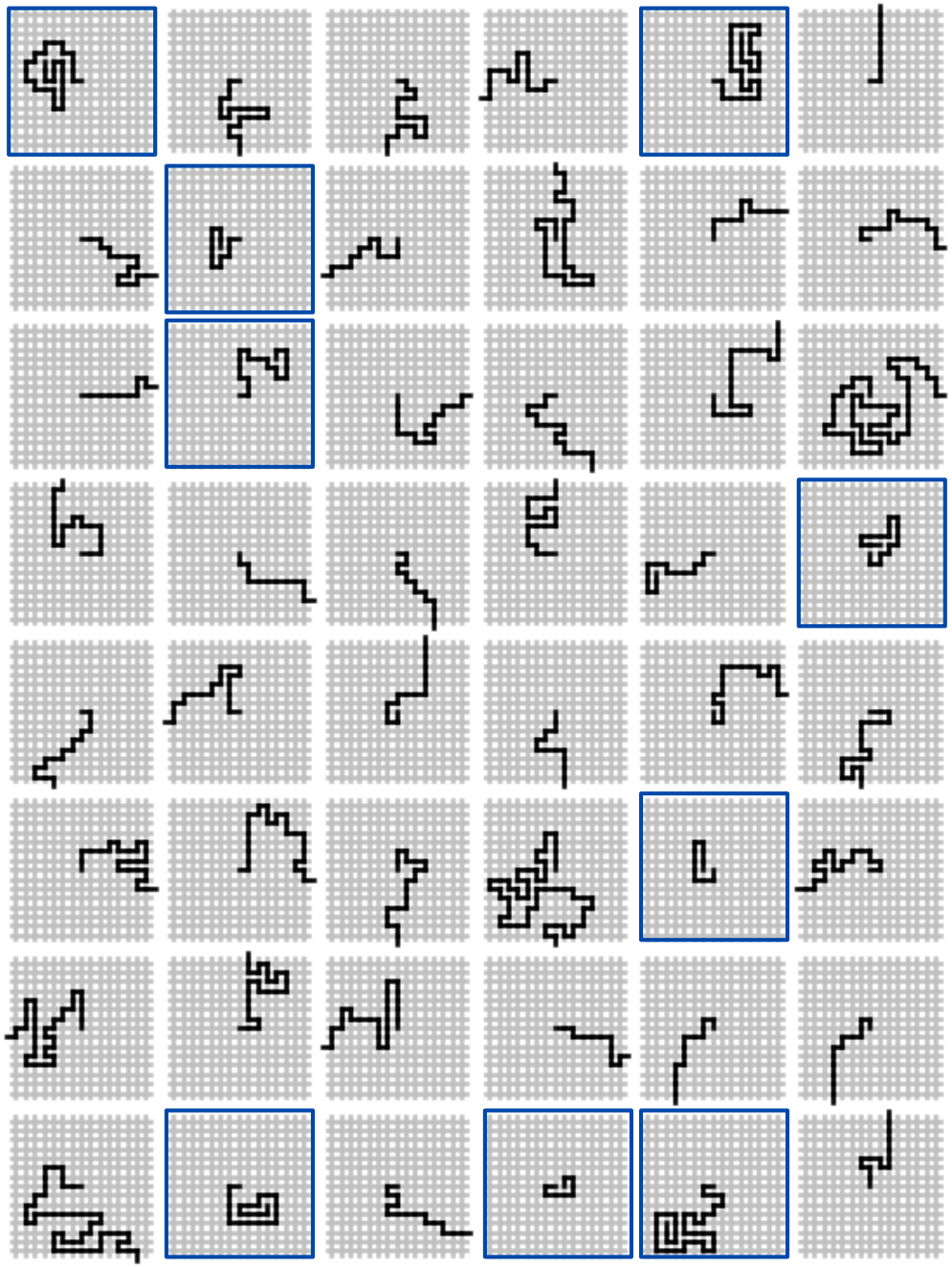
Self-Avoiding Walk: Implementation

```
public class SelfAvoidingWalk
{
   public static void main(String[] args)
   {
      int N = Integer.parseInt(args[0]);    // lattice size
      int T = Integer.parseInt(args[1]);    // number of trials
      int deadEnds = 0;                     // trials ending at dead end

      for (int t = 0; t < T; t++)
      {
         boolean[][] a = new boolean[N][N];  // intersections visited
         int x = N/2, y = N/2;               // current position

         while (x > 0 && x < N-1 && y > 0 && y < N-1)
         {

            if (a[x-1][y] && a[x+1][y] && a[x][y-1] && a[x][y+1])
            {  deadEnds++;  break; }

            a[x][y] = true;                 // mark as visited

            double r = Math.random();
            if      (r < 0.25) { if (!a[x+1][y]) x++; }
            else if (r < 0.50) { if (!a[x-1][y]) x--; }
            else if (r < 0.75) { if (!a[x][y+1]) y++; }
            else if (r < 1.00) { if (!a[x][y-1]) y--; }
         }
      }
      System.out.println(100*deadEnds/T + "% dead ends");
   }
}
```

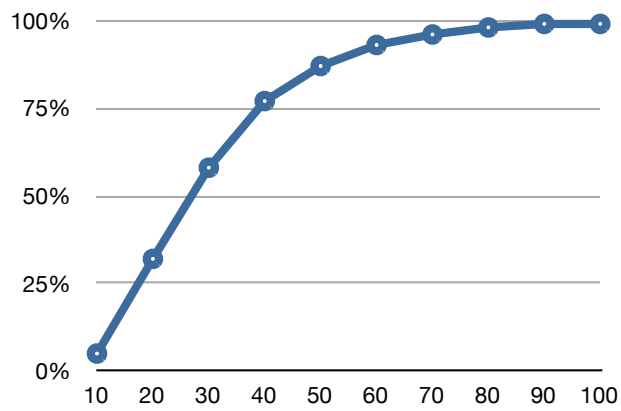
dead end

take a random step to a new intersection

Self-Avoiding Walks



```
% java SelfAvoidingWalk 10 100000
5% dead ends
% java SelfAvoidingWalk 20 100000
32% dead ends
% java SelfAvoidingWalk 30 100000
58% dead ends
% java SelfAvoidingWalk 40 100000
77% dead ends
% java SelfAvoidingWalk 50 100000
87% dead ends
% java SelfAvoidingWalk 60 100000
93% dead ends
% java SelfAvoidingWalk 70 100000
96% dead ends
% java SelfAvoidingWalk 80 100000
98% dead ends
% java SelfAvoidingWalk 90 100000
99% dead ends
% java SelfAvoidingWalk 100 100000
99% dead ends
```

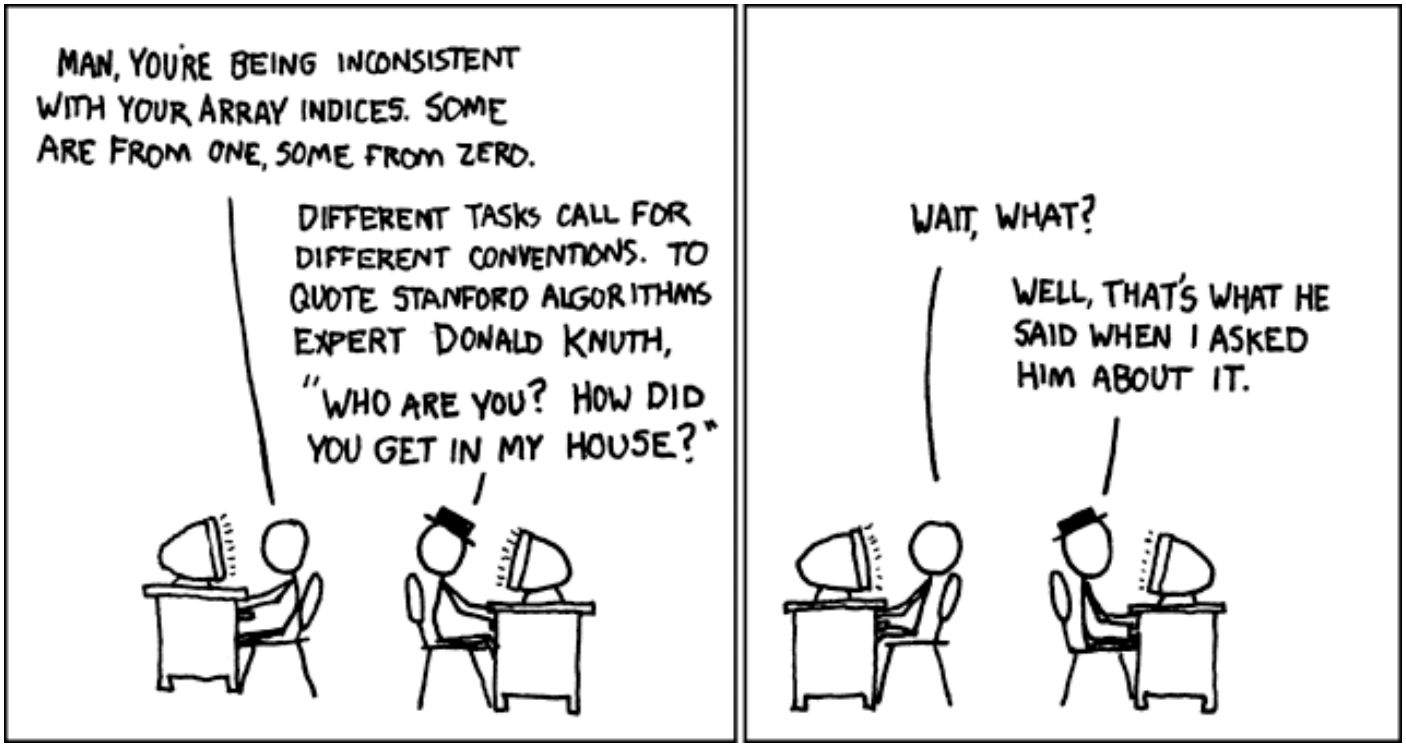


Summary

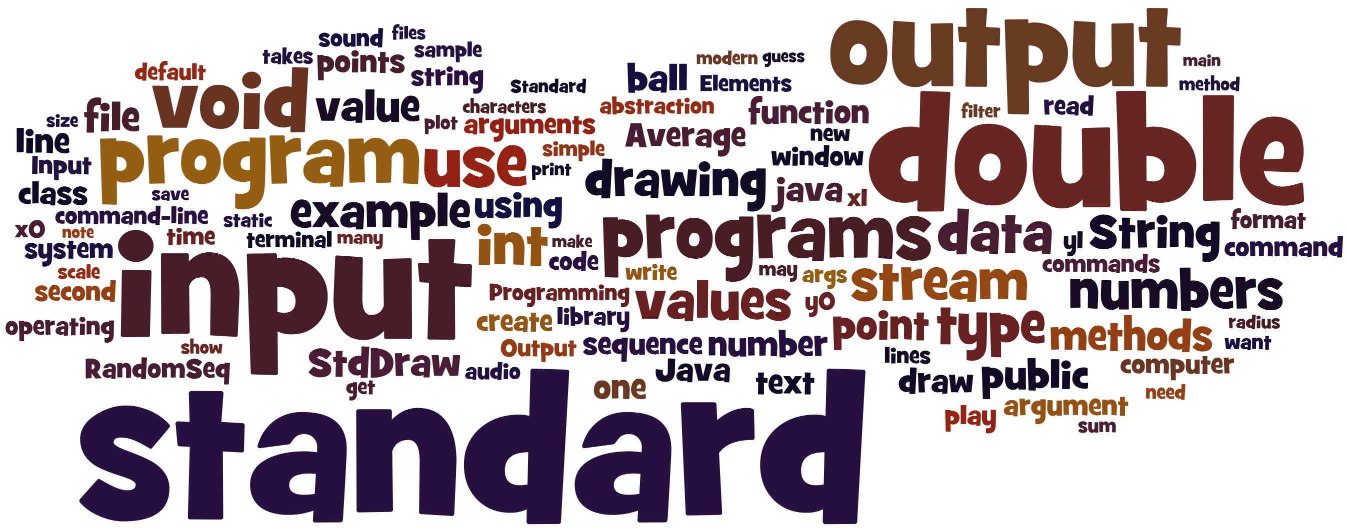
Arrays.

- Organized way to store huge quantities of data.
- Almost as easy to use as primitive types.
- Can directly access an element given its index.

Ahead. Reading in large quantities of data from a file into an array.

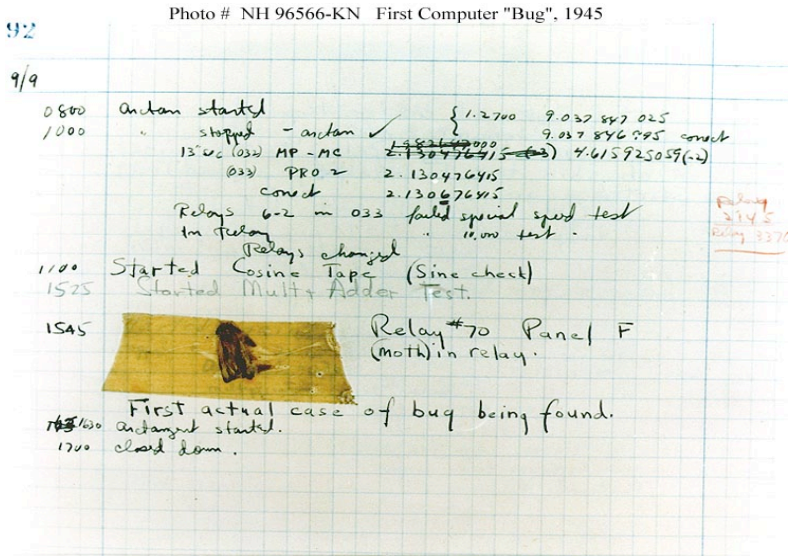


http://imgs.xkcd.com/comics/donald_knuth.png


sound
files
takes
sample
points
string
default
void
value
Standard
ball
Elements
modern
guess
output
main
method
size
file
plot
characters
arguments
abstraction
function
filter
read
line
program
use
simple
Average
new
Input
print
window
double
class
save
example
using
drawing
java
xl
command-line
static
xO
note
many
terminal
make
programs
data
yl
String
format
system
time
int
code
command
scale
write
may
args
commands
second
input
Programming
values
stream
numbers
yo
operating
create
library
point
type
methods
radius
show
Output
sequence
number
lines
want
RandomSeq
StdDraw
audio
computer
get
one
Java
text
draw
public
need
play
argument
sum
standard

Debugging [continued]

Admiral Grace Murray Hopper



http://www.history.navy.mil/photos/images/h96000/h96566kc.htm

Debugging Your Program

Debugging Your Program. [summary]

1. Create the program.

2. Compile it.
   Compiler says: That's not a legal program.
   Back to step 1 to fix your errors of syntax.

3. Execute it.
   Result is bizarrely (or subtly) wrong.
   Back to step 1 to fix your errors of semantics.

4. Enjoy the satisfaction of a working program!
   [but stay tuned for more debugging]

Debugging: Performance Errors

Performance error. Correct program, but too slow.

- Are all iterations of inner loop necessary?
- Improve or change underlying algorithm.

```
public class Factors
{
   public static void main(String[] args)
   {
      long N = Long.parseLong(args[0])
      for (int i = 2; i < N; i++)
      {  // Check whether i is a factor.
         while (N % i == 0)
         {  // If so, print and divide.
            System.out.print(i + " ");
            N = N / i;
         }
      }
      if (N > 1) System.out.println(N);
      else       System.out.println();
   }
}
```

```
% java Factors 11111111
11 73 101 137
% java Factors 11111111111
21649 513239
% java Factors 11111111111111
11 239 4649 909091
% java Factors 1111111111111111
2071723
%
```

very long wait
(with a surprise ending)

Debugging: Performance Errors

Performance error. Correct program, but too slow.

- Are all iterations of inner loop necessary?
- Improve or change underlying algorithm.

```
public class Factors
{
   public static void main(String[] args)
   {
      long N = Long.parseLong(args[0]);
      for (int i = 2; i < N/i; i++)
      {  // Check whether i is a factor.
         while (N % i == 0)
         {  // If so, print and divide.
            System.out.print(i + " ");
            N = N / i;
         }
      }
      if (N > 1) System.out.println(N);
      else       System.out.println();
   }
}
```

Fixes performance error: terminate when i*i > N since no larger factors left

```
% java Factors 11111111
11 73 101 137
% java Factors 11111111111
21649 513239
% java Factors 11111111111111
11 239 4649 909091
% java Factors 1111111111111111
2071723 5363222357
%
```

Program Development: Analysis

Q. How large an integer can I factor?

```
% java Factors 3757208
2 2 2 7 13 13 397


% java Factors 92011111169755555703
92011111169755555703
```

after a few minutes of computing....

in largest factor ⟶

digits	(i <= N)	(i*i <= N)
3	instant	instant
6	0.15 seconds	instant
9	77 seconds	instant
12	21 hours †	0.16 seconds
15	2.4 years †	2.7 seconds
18	2.4 millennia †	92 seconds

† estimated, using analytic number theory

Note. Can't break RSA this way (experts are still trying)

Debugging Your Program

Debugging Your Program. [summary]

1. Create the program.

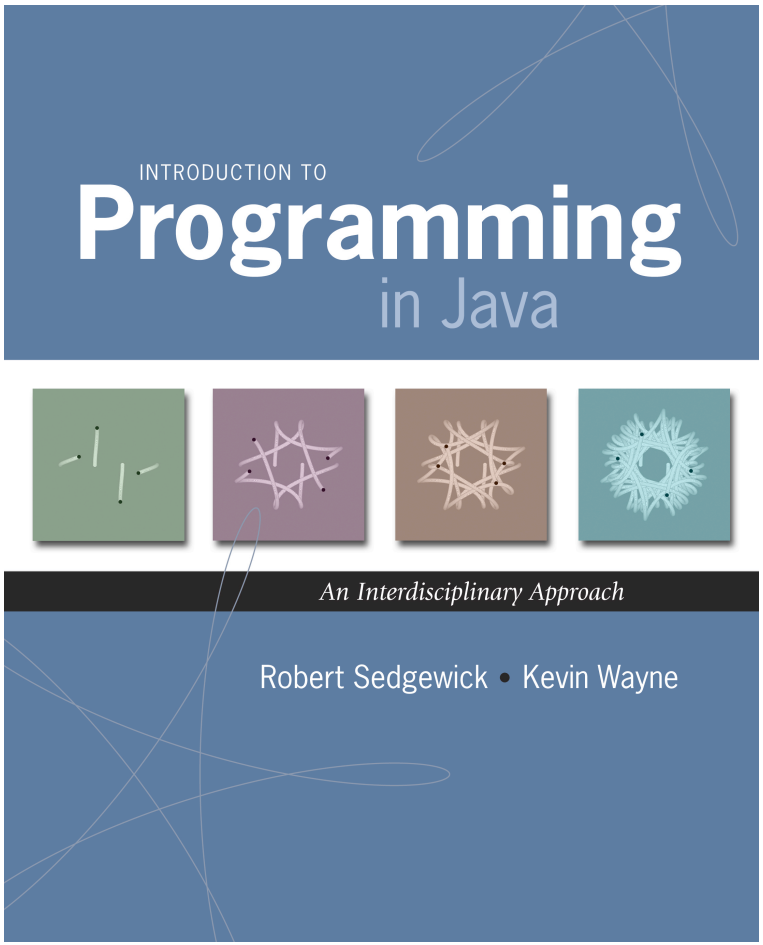
2. Compile it.
   Compiler says: That's not a legal program.
   Back to step 1 to fix your errors of syntax.

3. Execute it.
   Result is bizarrely (or subtly) wrong.
   Back to step 1 to fix your errors of semantics.

4. Enjoy the satisfaction of a working program!

5. Too slow? Back to step 1 to try a different algorithm.

1.5 Input and Output



Input and Output

Input devices.

Keyboard

Mouse

Hard drive

Network

Digital camera

Microphone

Output devices.

Display

Speakers

Hard drive

Network

Printer

MP3 Player

Goal. Java programs that interact with the outside world.

Input and Output

Input devices.


Keyboard


Mouse


Hard drive


Network


Digital camera


Microphone

Output devices.


Display


Speakers


Hard drive


Network


Printer

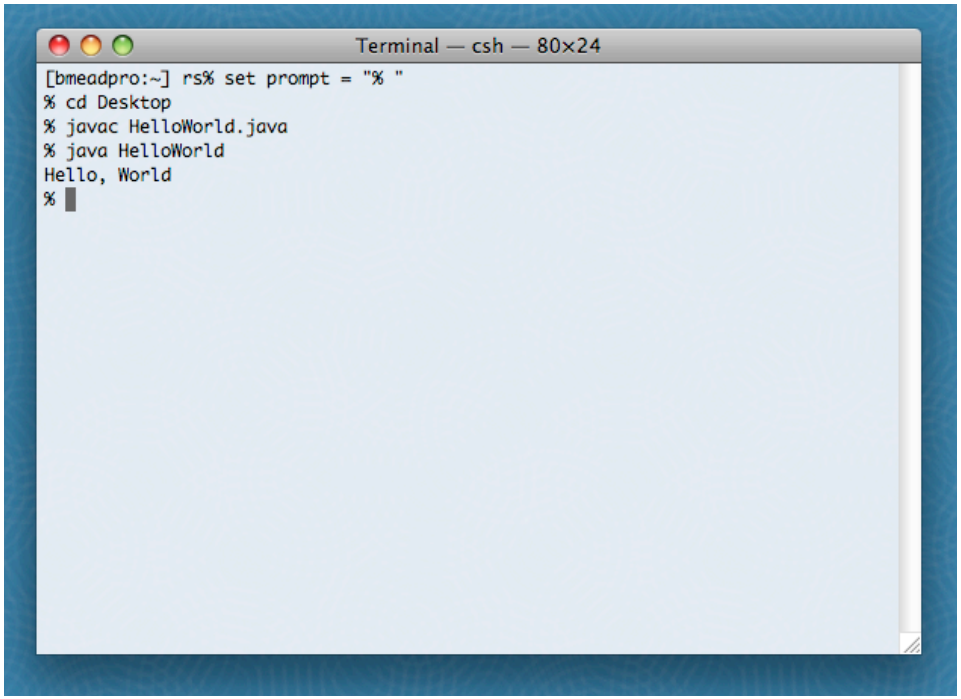

MP3 Player

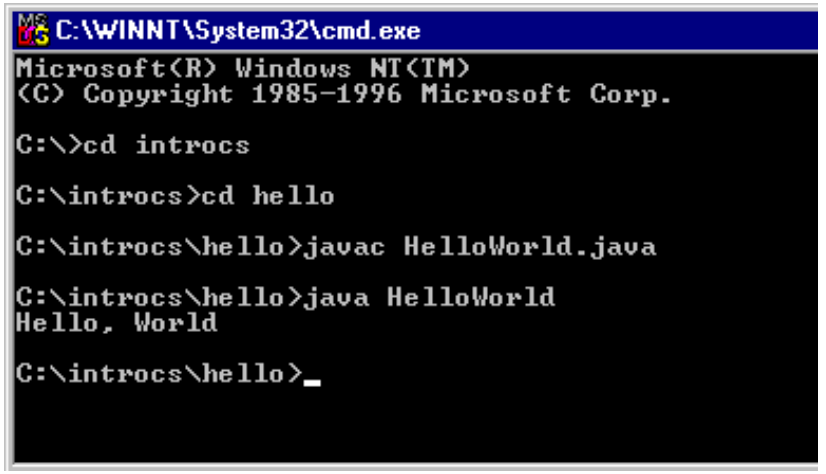
Our approach.

- Define Java libraries of functions for input and output.
- Use operating system (OS) to connect Java programs to:
  file system, each other, keyboard, mouse, display, speakers.

Terminal

Terminal. Application for typing commands to control the operating system.





Microsoft Windows

Command-Line Input and Standard Output

Command-line input. Read an integer `N` as command-line argument.

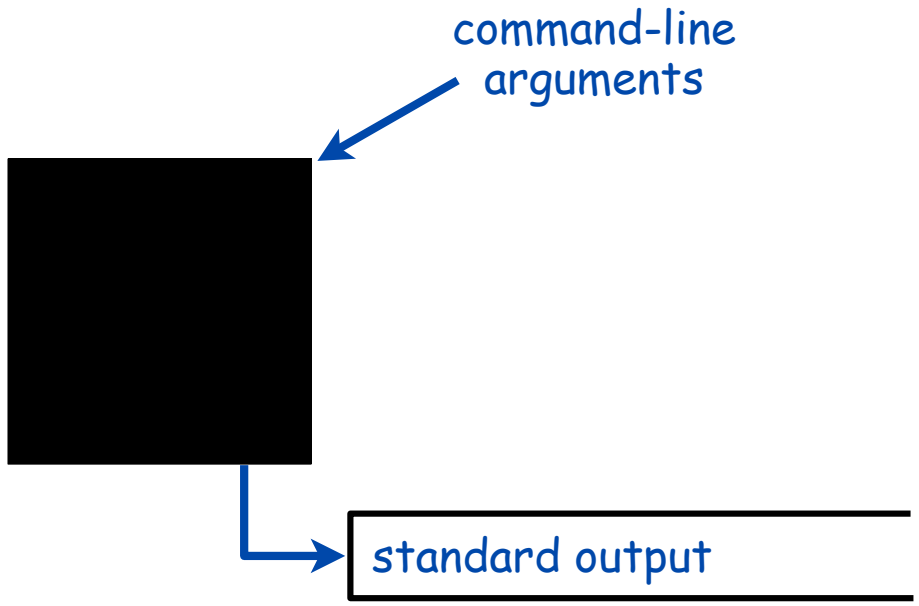
Standard output.

- Flexible OS abstraction for output.
- In Java, output from `System.out.println()` goes to standard output.
- By default, standard output is sent to Terminal.

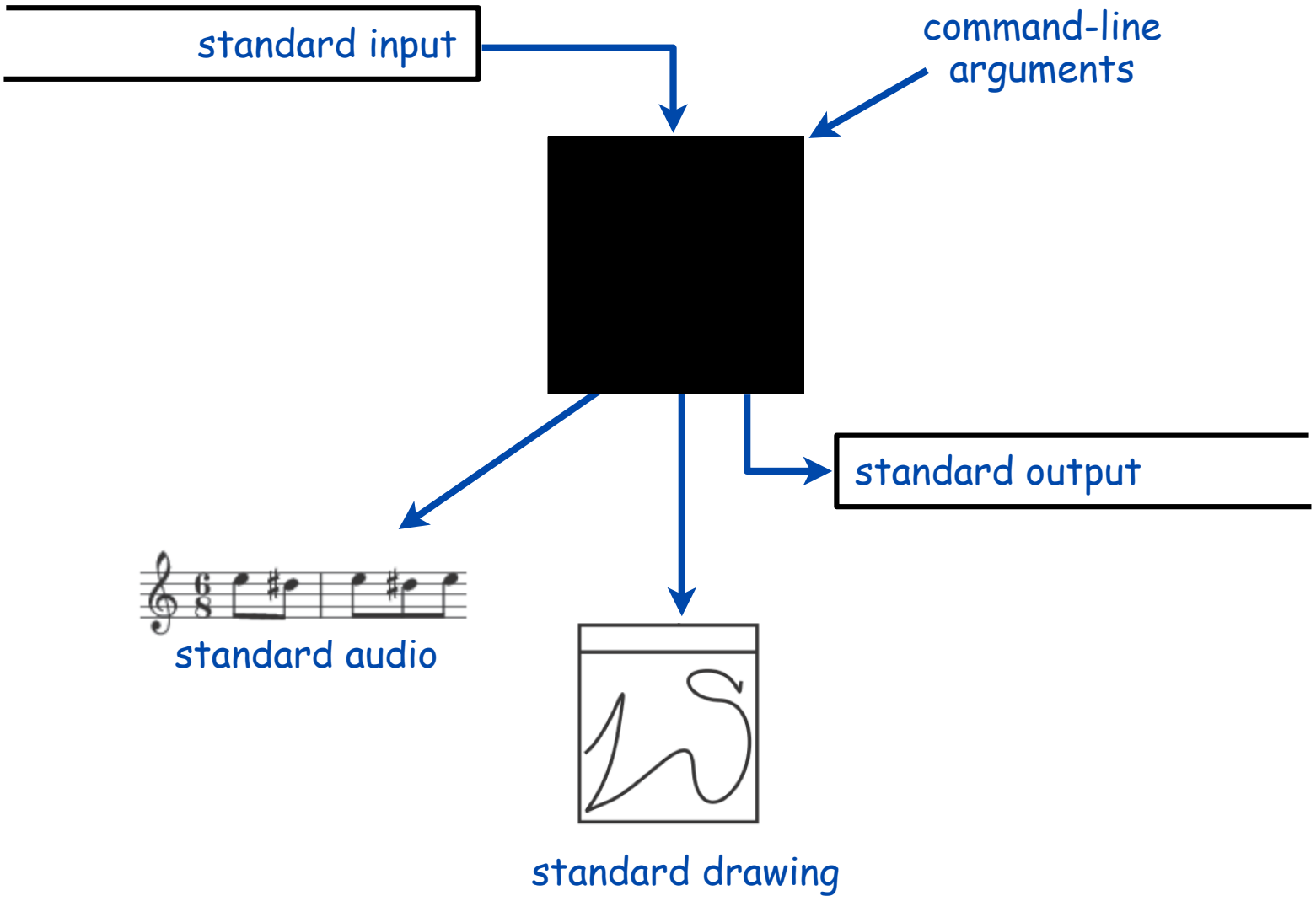
```
public class RandomSeq
{
   public static void main(String[] args)
   {
      int N = Integer.parseInt(args[0]);
      for (int i = 0; i < N; i++)
         System.out.println(Math.random());
   }
}
```

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

Old Bird's Eye View



New Bird's Eye View



Standard Input and Output

Command-Line Input vs. Standard Input

Command-line inputs.

- Useful for providing a few user values (arguments) to a program.
- Not practical for a large or unspecified number of user inputs.
- Input entered before program begins execution.

Standard input.

- Flexible OS abstraction for input.
- Useful for providing an unlimited amount of data to a program.
- By default, standard input is received from Terminal window.
- Input entered while program is executing.

Standard Input and Output

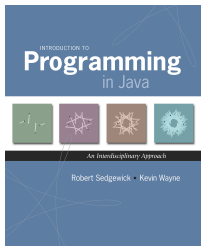
Standard input. `StdIn` library has methods to read text input.

Standard output. `StdOut` library has methods to write text output.

public class StdIn	
boolean isEmpty()	`true` if no more values, `false` otherwise
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 readLine()	read the rest of the line
String readAll()	read the rest of the text

public class StdOut	
void print(String s)	print `s`
void println(String s)	print `s`, followed by a newline
void println()	print a new line
void printf(String f, ...)	formatted print

libraries developed for this course (and also broadly useful)



Standard IO Warmup

To use. Download `StdIn.java` and `StdOut.java` from booksite, and put in working directory (or use classpath).

see booksite

```
public class Add
{
   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 Add
Type the first integer: 1
Type the second integer: 2
Their sum is 3
```

Standard IO Example: Averaging A Stream of Numbers

Average. Read in a stream of numbers, and print their average.

```
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
```

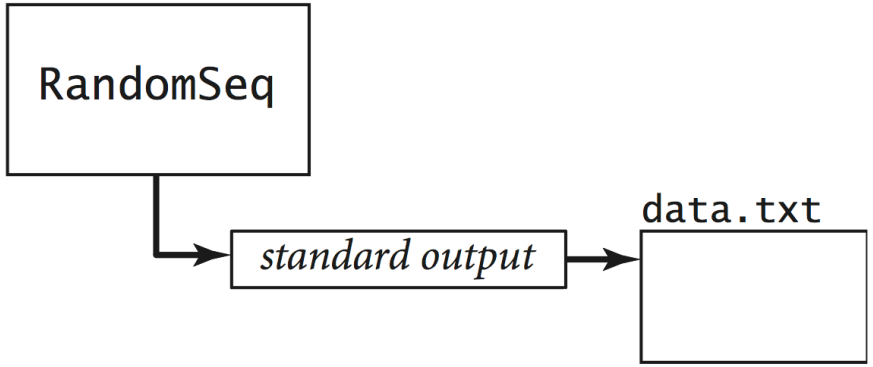
Key point. Program does not limit amount of data.

<Ctrl-d> is OS X/Linux/Unix/DrJava EOF
<Ctrl-z> is Windows analog

Redirection and Piping

Redirecting Standard Output

Redirecting standard output. Use OS directive to send standard output to a file for permanent storage (instead of terminal window).

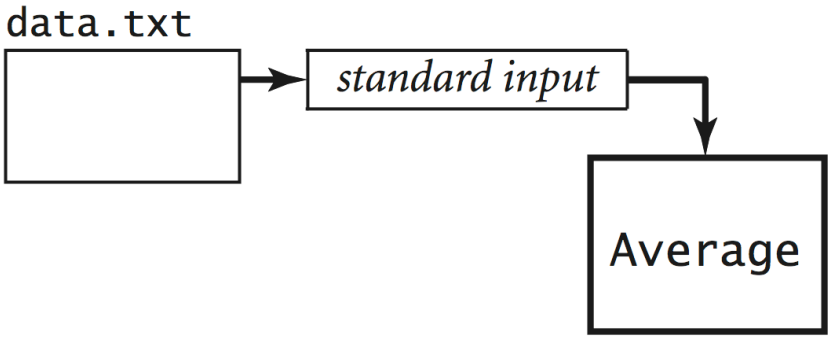



```
% java RandomSeq 1000 > data.txt
```

redirect standard output

Redirecting Standard Input

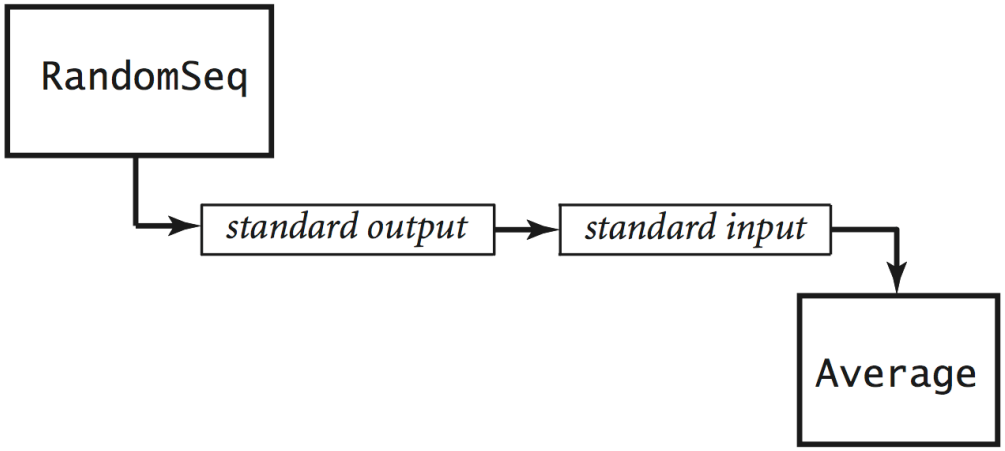
Redirecting standard input. Use OS directive to read standard input from a file (instead of terminal window).




```
% more < data.txt
0.5475375782884312
0.4971087292684019
0.2312380804175381 3
...

                              standard input

% java Average < data.txt
0.4947655567740991
```

Connecting Programs

Piping. Use OS directive to make the standard output of one program become the standard input of another.




```
% java RandomSeq 1000000 | java Average
0.499797047301602 8


% java RandomSeq 1000000 | java Average
0.500207187564484 2
```

Key point. Program does not limit amount of data.

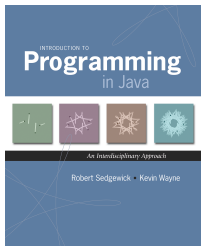
Standard Drawing

Standard Drawing

Standard drawing. `StdDraw` library has methods to produce graphical output.

```
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 filledCircle(double x, double y, double r)
   void square(double x, double y, double r)
   void filledSquare(double x, double y, double r)
   void polygon(double[] x, double[] y)
   void filledPolygon(double[] x, double[] y)
   void setXscale(double x0, double x1)   reset x range
   void setYscale(double y0, double y1)   reset y range
   void setPenRadius(double r)
   void setPenColor(Color c)
   void setFont(Font f)
   void setCanvasSize(int w, int h)
   void clear(Color c)                    clear the canvas; color it c
   void show(int dt)                      show all; pause dt millisecs
   void save(String filename)             save to .jpg or .png file
```

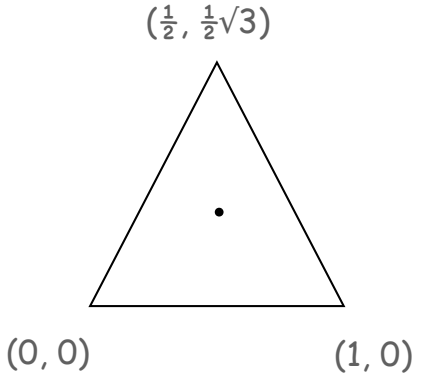
library developed for this course (and also broadly useful)

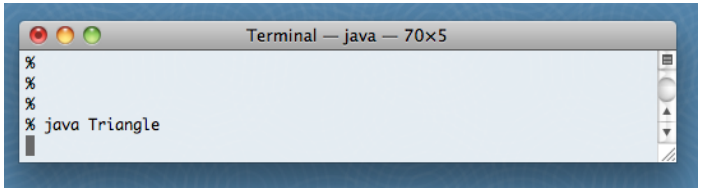


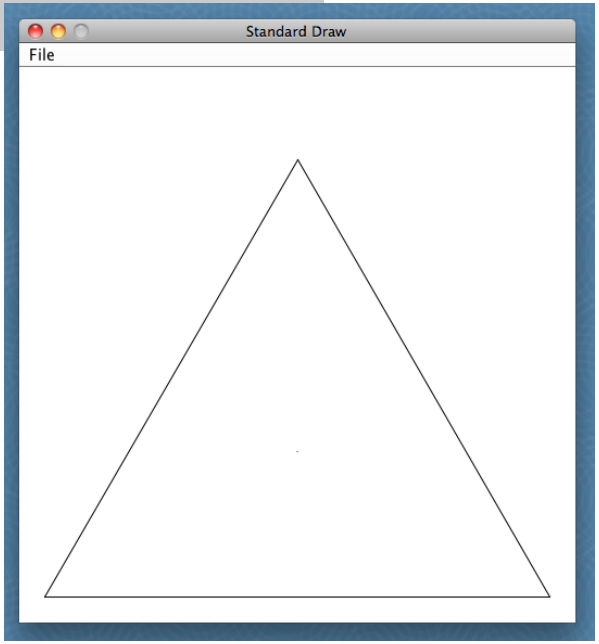
"Hello World" for Standard Draw

To use. Download `stdDraw.java` and put in working directory.

```
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);
      StdDraw.point(0.5, t/3.0);
   }
}
```







Data Visualization

Plot filter. Read in a sequence of (x, y) coordinates from standard input, and plot using standard drawing.

```
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);
      }
   }
}
```

rescale coordinate system

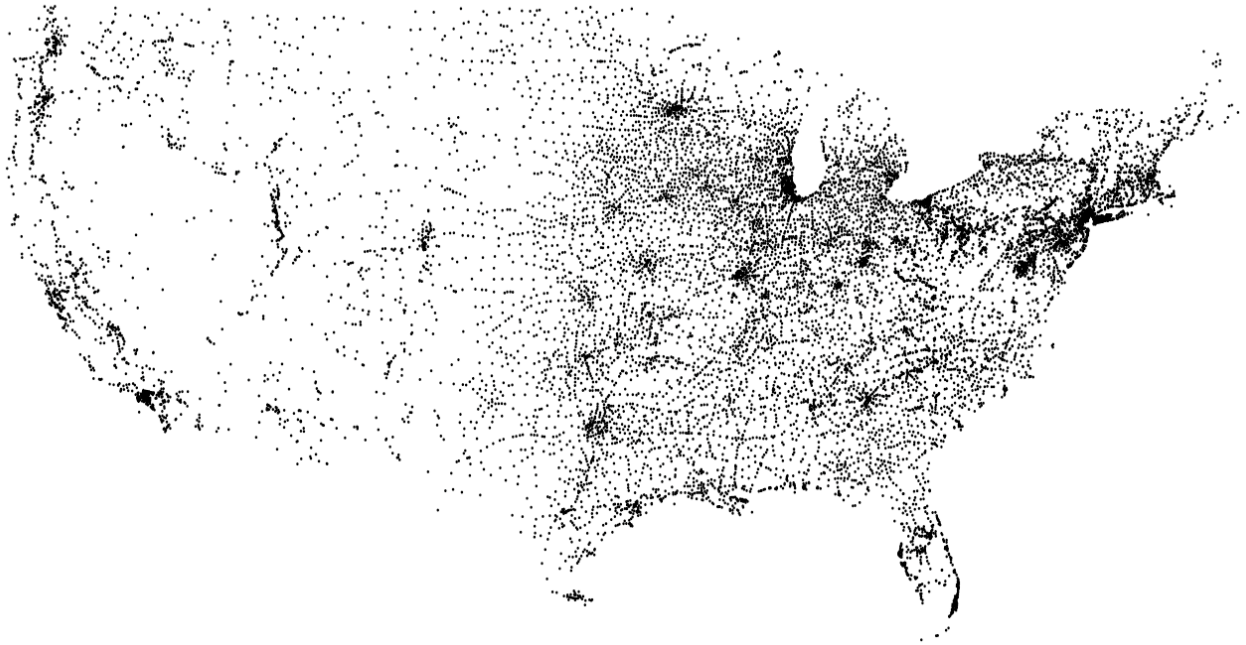
read in points, and plot them

Data Visualization

```
% 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
```

bounding box

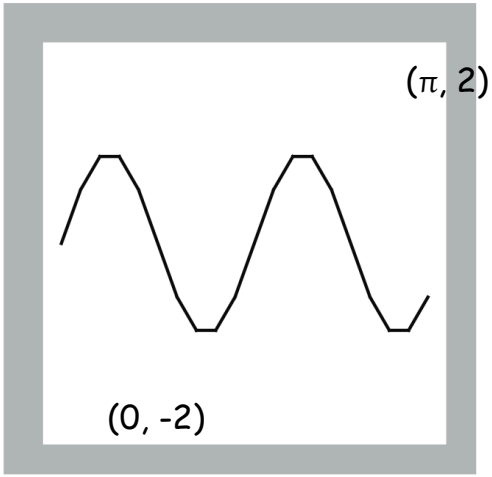
coordinates of 13,509 US cities



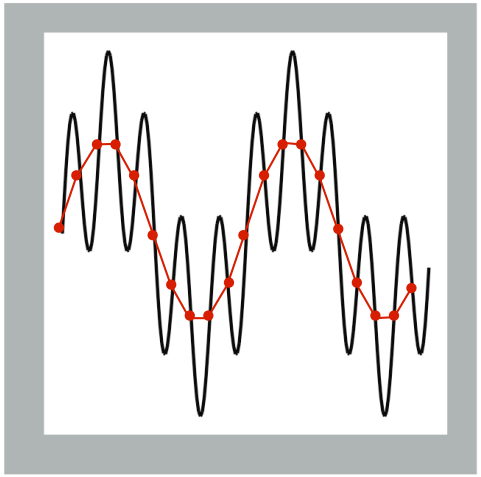
Plotting a Function with `StdDraw`

```
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]);
```

$N = 20$



$N = 200$



Lesson 1: Plotting is simple.

Lesson 2: If you don't plot enough points, you might miss something!

$$y = \sin 4x + \sin 20x, \; x \in [0, \pi]$$

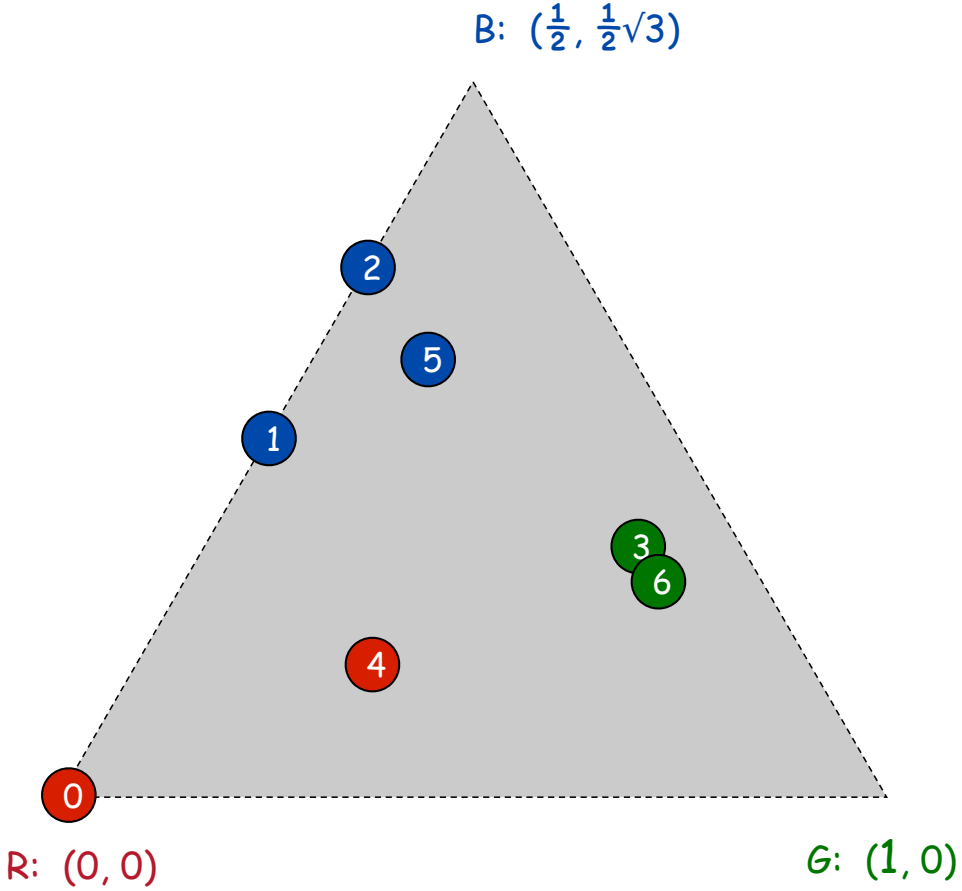
Chaos Game

Chaos game. Play on equilateral triangle, with vertices R, G, B.

- Start at R.
- Repeat the following N times:
  - pick a random vertex
  - move halfway between current point and vertex
  - draw a point in color of vertex

Q. What picture emerges?

B B G R B G ...



Example: Chaos Game

```
public class Chaos
{
   public static void main(String[] args)
   {
      int T = Integer.parseInt(args[0]);
      double[] cx = { 0.000, 1.000, 0.500 };
      double[] cy = { 0.000, 0.000, 0.866 };


      double x = 0.0, y = 0.0;
      for (int t = 0; t < T; t++)
      {
         int r = (int) (Math.random() * 3);
         x = (x + cx[r]) / 2.0;
         y = (y + cy[r]) / 2.0;
         StdDraw.point(x, y);
      }
   }
}
```

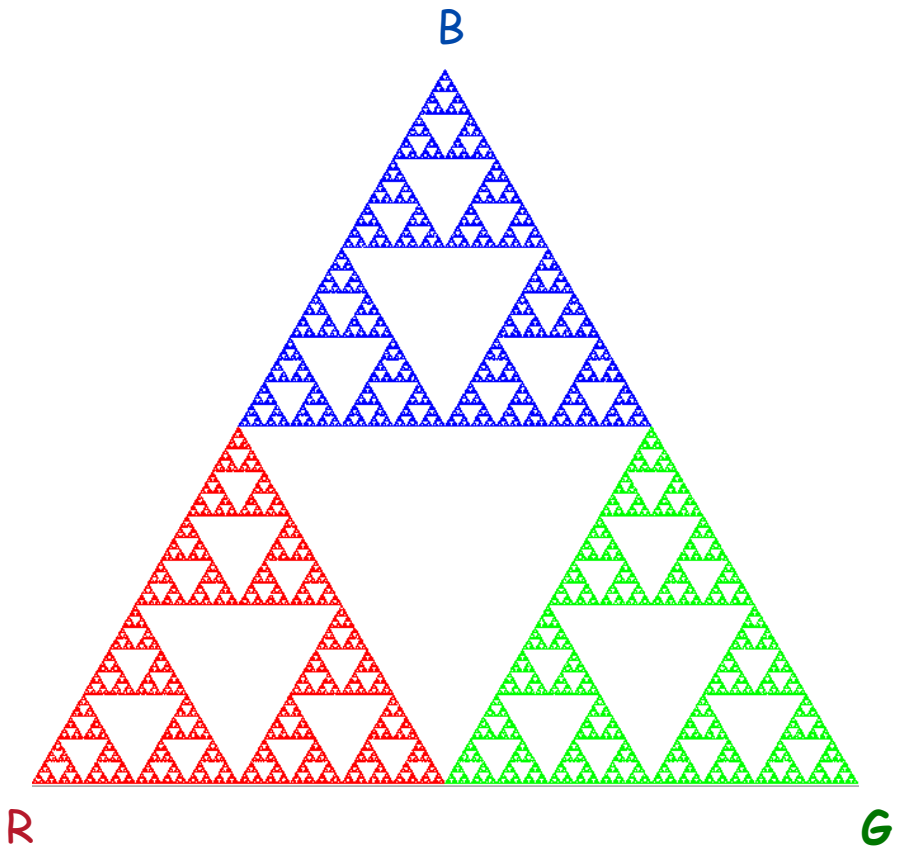
$\frac{1}{2}\sqrt{3}$
(best to avoid hardwired constants like this)

result: 0, 1 , or 2

Chaos Game

Easy modification. Color point according to random vertex chosen using `StdDraw.setPenColor(StdDraw.RED)` to change the pen color.

```
% java Chaos 10000
```



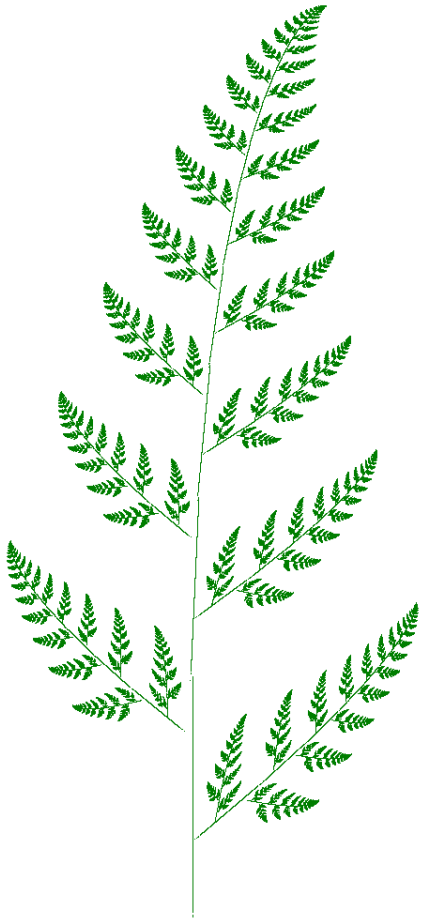
Sierpinski triangle

Commercial Break

Barnsley Fern

Barnsley fern. Play chaos game with different rules.

probability	new x	new y
2%	.50	.27y
15%	-.14x + .26y + .57	.25x + .22y - .04
13%	.17x - .21y + .41	.22x + .18y + .09
70%	.78x + .03y + .11	-.03x + .74y + .27



Q. What does computation tell us about nature?

Q. What does nature tell us about computation?

20th century sciences. Formulas.

21st century sciences. Algorithms?

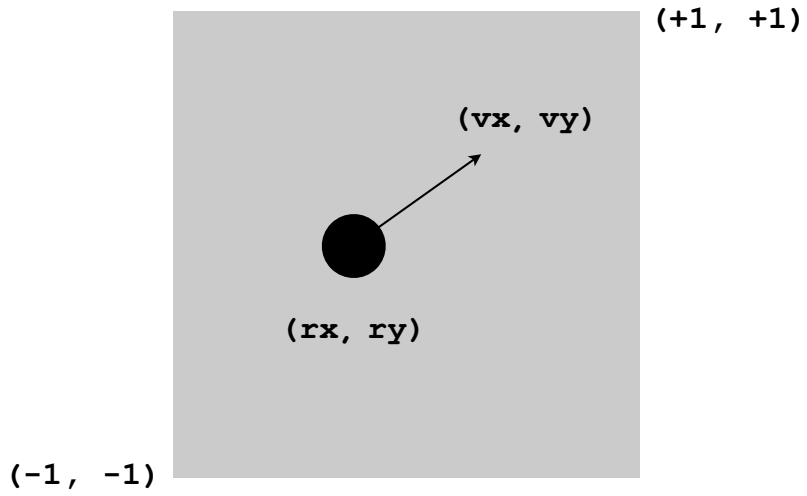
Animation

Animation loop. Repeat the following:

- Clear the screen.
- Move the object.
- Draw the object.
- Display and pause for a short while.

Ex. Bouncing ball.

- Ball has position `(rx, ry)` and constant velocity `(vx, vy)`.
- Detect collision with wall and reverse velocity.



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)
      {
         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.GRAY);
         StdDraw.filledSquare(0.0, 0.0. 1.0);
         StdDraw.setPenColor(StdDraw.BLACK);
         StdDraw.filledCircle(rx, ry, radius);
         StdDraw.show(20);
      }
   }
}
```

bounce

update position

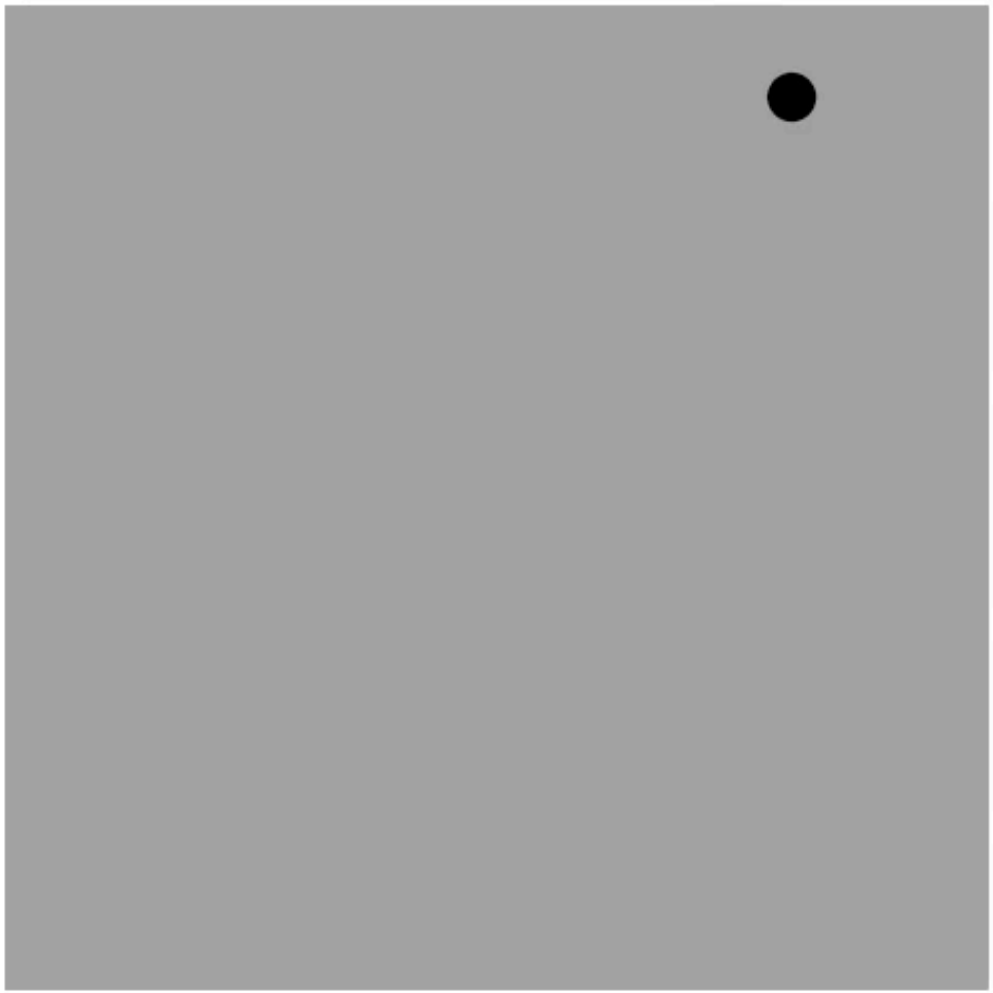
clear background

draw the ball

turn on animation mode:
display and pause for 50ms

Bouncing Ball Demo

% **java BouncingBall**



Special Effects

Images. Put `.gif`, `.png`, or `.jpg` file in the working directory and use `StdDraw.picture()` to draw it.

Sound effects. Put `.wav`, `.mid`, or `.au` file in the working directory and use `StdAudio.play()` to play it.

stay tuned for more on `StdAudio`

Ex. Modify `BouncingBall` to display image and play sound upon collision.

- Replace `StdDraw.filledCircle()` with:

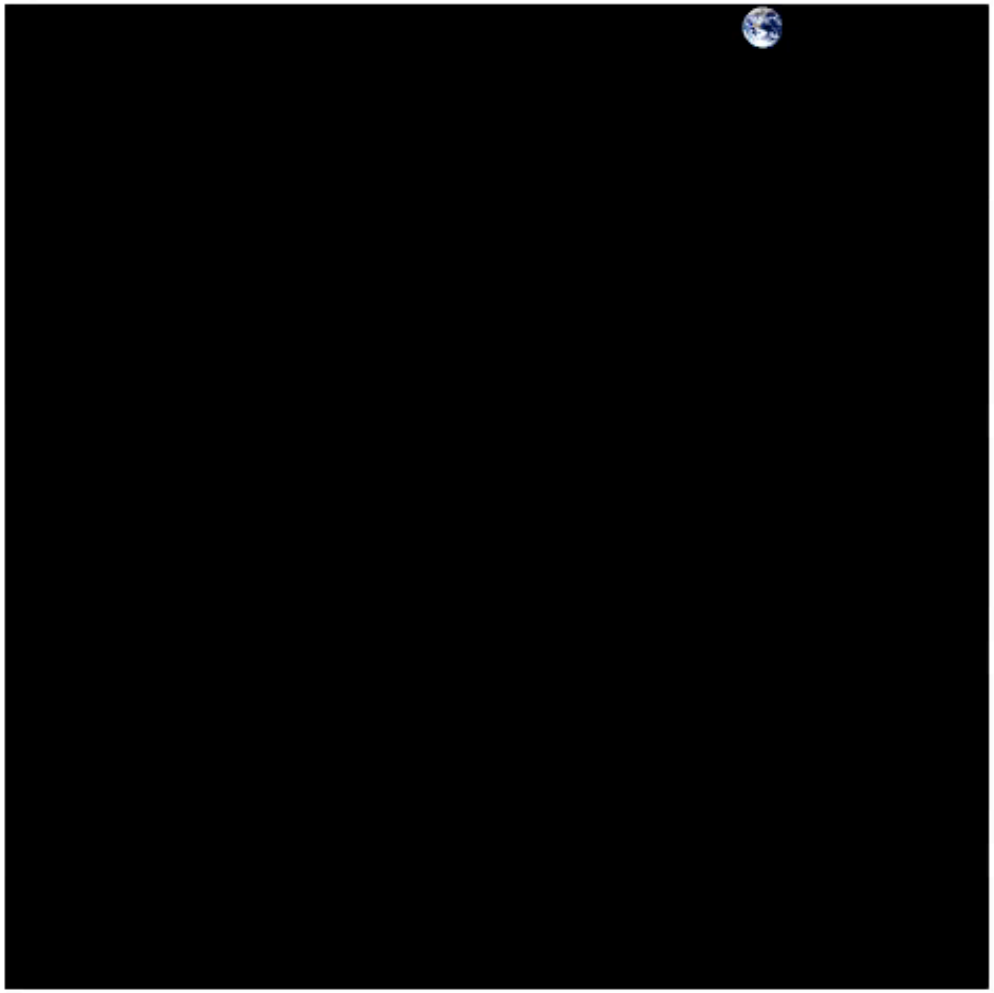
```
StdDraw.picture(rx, ry, "earth.gif");
```

- Add following code upon collision with wall:

```
StdAudio.play("boing.wav");
```

Deluxe Bouncing Ball Demo

% **java DeluxeBouncingBall**



Bouncing Ball Challenge

Q. What happens if you call `StdDraw.filledSquare()` before instead of inside loop?

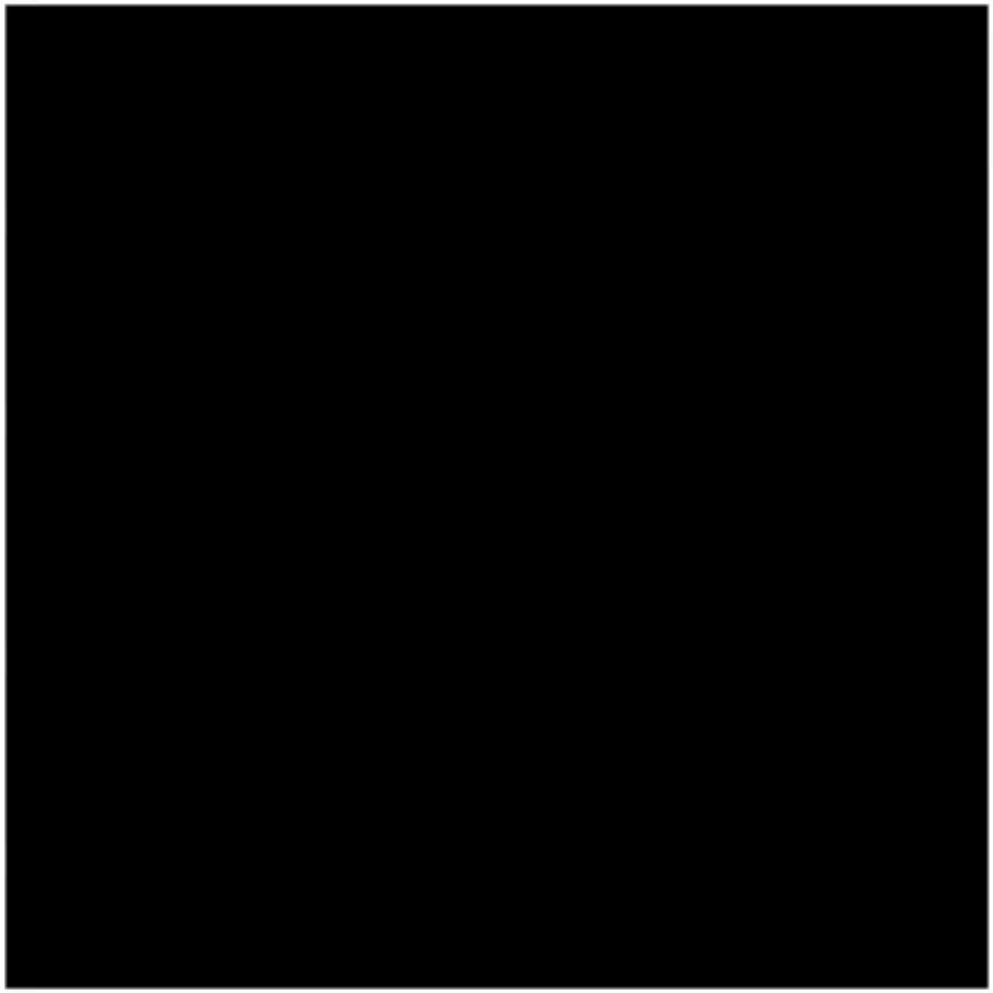
```
public class BouncingBall
{
   public static void main(String[] args)
   {
      double rx = .480, ry = .860;
      double vx = .015, vy = .023;
      double radius = .05;

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

      while(true)
      {
         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.GRAY);
         StdDraw.filledSquare(0.0, 0.0. 1.0);
         StdDraw.setPenColor(StdDraw.BLACK);
         StdDraw.filledCircle(rx, ry, radius);
         StdDraw.show(20);
      }
   }
}
```

```
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.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.GRAY);
         StdDraw.setPenColor(StdDraw.BLACK);
         StdDraw.filledCircle(rx, ry, radius);
         StdDraw.show(20);
      }
   }
}
```

Bouncing Ball Challenge

Q. What happens if you call `StdDraw.filledSquare()` before instead of inside loop?

```
% java DeluxeBouncingBall
```



Standard Audio

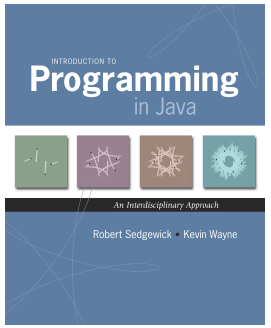
Digital Audio in Java

Standard audio. Library for playing digital audio.

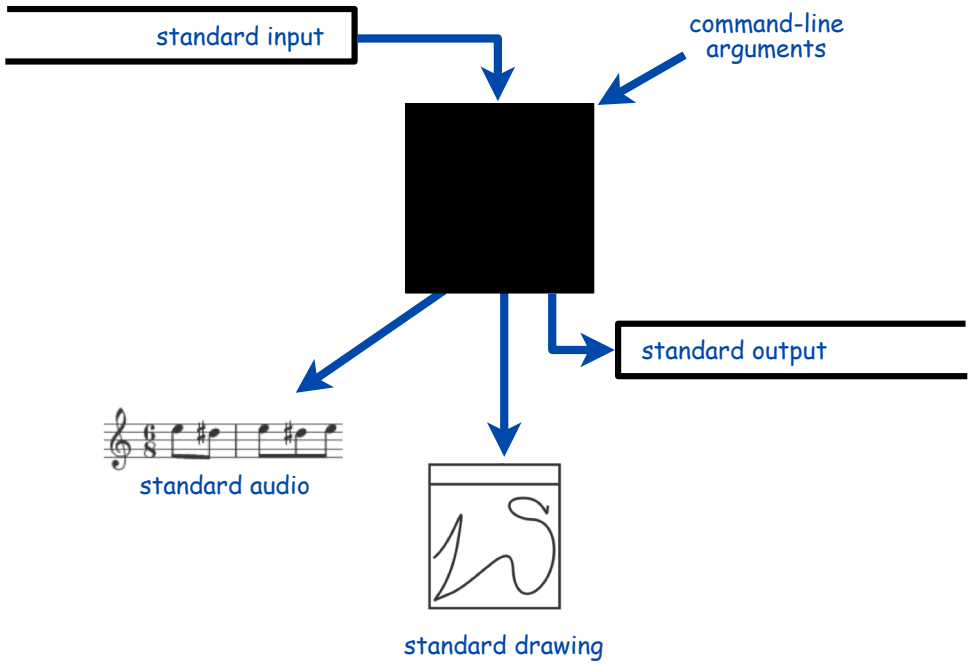
public class StdAudio	
void play(String file)	*play the given .wav file*
void play(double[] a)	*play the given sound wave*
void play(double x)	*play sample for 1/44100 second*
void save(String file, double[] a)	*save to a .wav file*
double[] read(String file)	*read from a .wav file*

library developed for this course (also broadly useful)

Stay tuned. Example client in next lecture.



Input/Output Summary



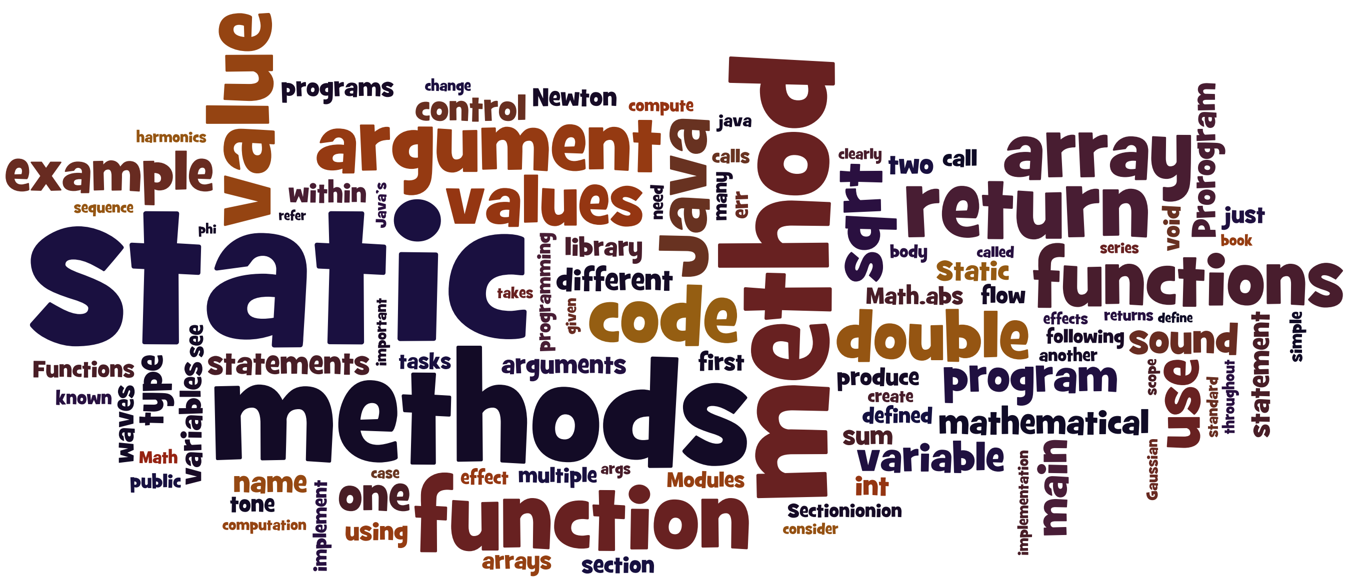
Command-line arguments. Parameters to control your program.

Standard input. Data for your program to process.

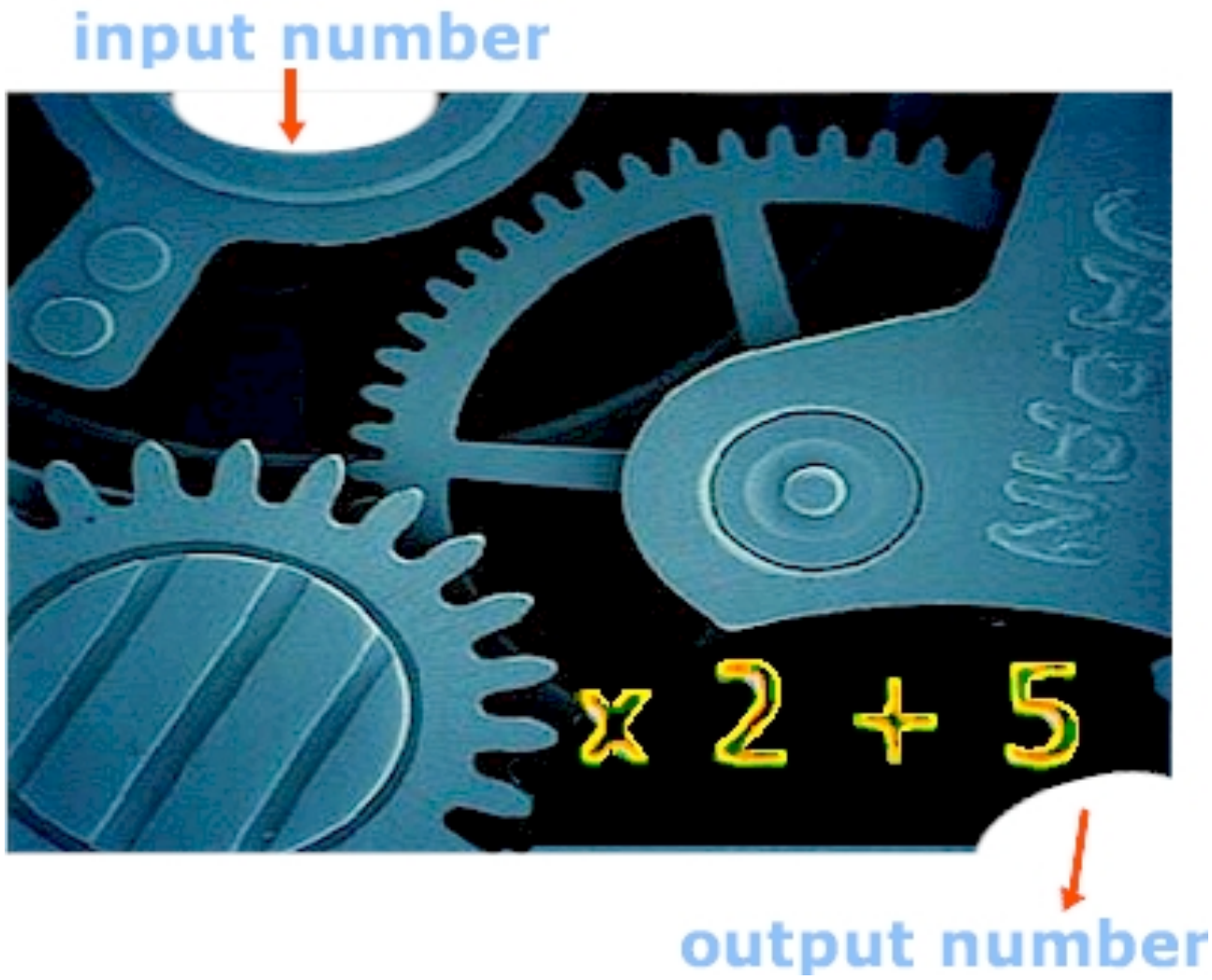
Standard output. Results of your program, or data for another program.

Standard drawing. Graphical output.

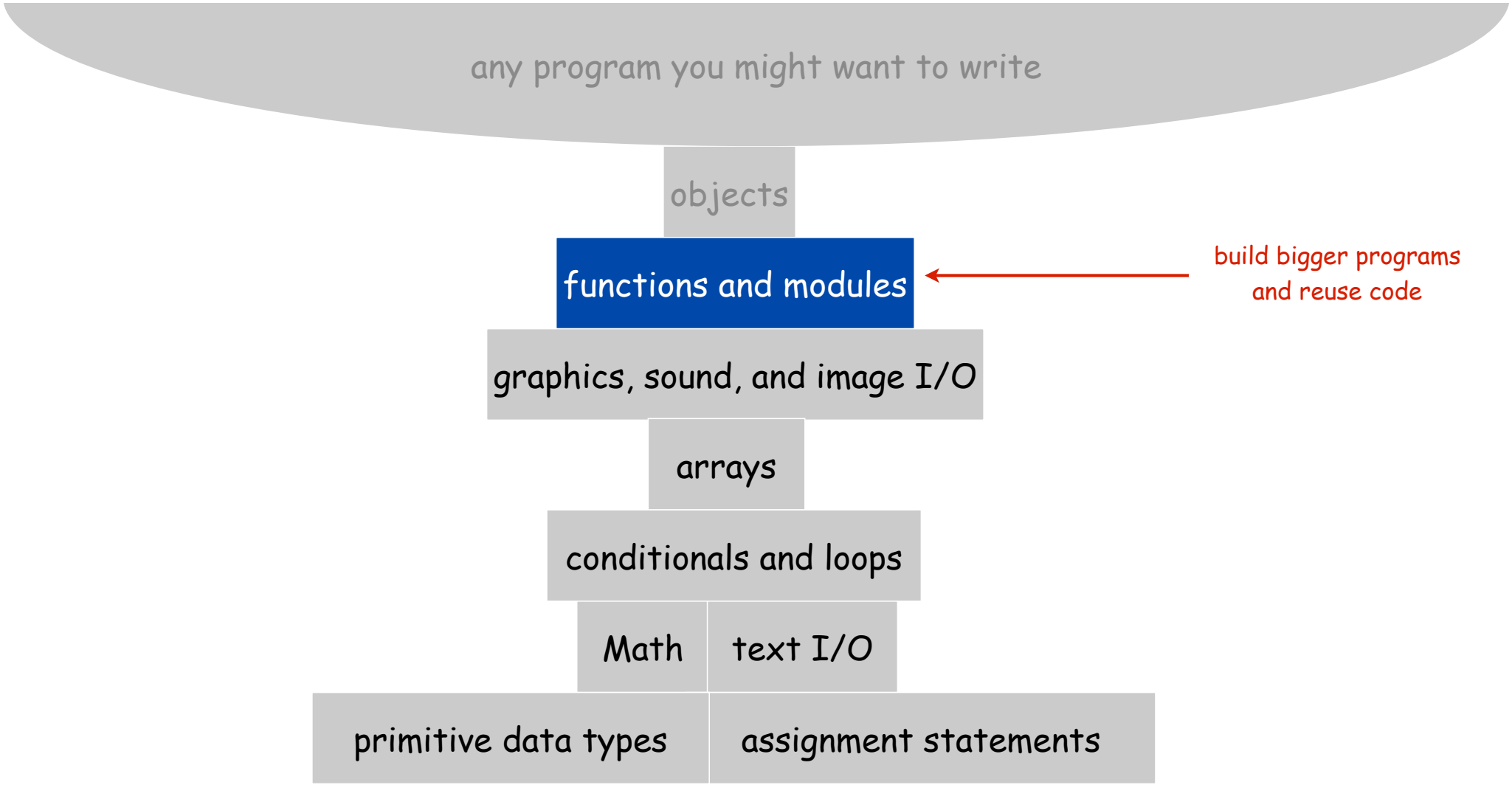
Standard audio. Sound output.


static
methods
method
function
functions
argument
values
value
return
array
code
double
example
Java
sqrt
program
programs
control
Newton
change
compute
harmonics
sequence
within
statements
arguments
different
library
variable
variables
mathematical
use
main
sound
Functions
type
waves
known
Math
public
name
tone
one
computation
implement
using
arrays
section
multiple
args
effect
case
Modules
int
Sectionionion
consider
produce
defined
create
sum
Math.abs
flow
Static
first
tasks
see
important
takes
given
programming
java
calls
many
need
err
clearly
two
call
body
called
series
book
Prorogram
void
just
effects
returns
define
following
another
statement
simple
scope
standard
throughout
Gaussian
implementation
phi
refer
Java's

2.1 Functions



A Foundation for Programming



Functions (Static Methods)

Java function.

- Takes zero or more input arguments.
- Returns zero or one output value.
- May cause side effects (e.g., output to standard draw).

more general than mathematical functions

Applications.

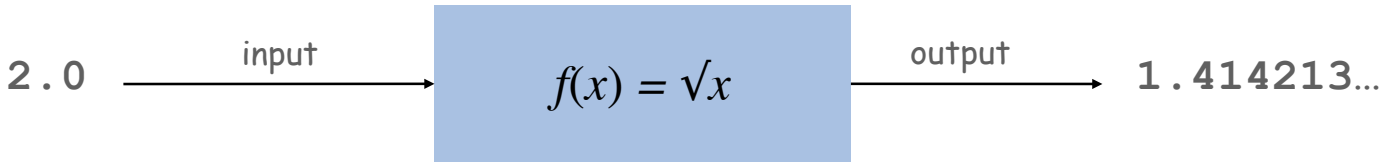
- Scientists use mathematical functions to calculate formulas.
- Programmers use functions to build modular programs.
- You use functions for both.

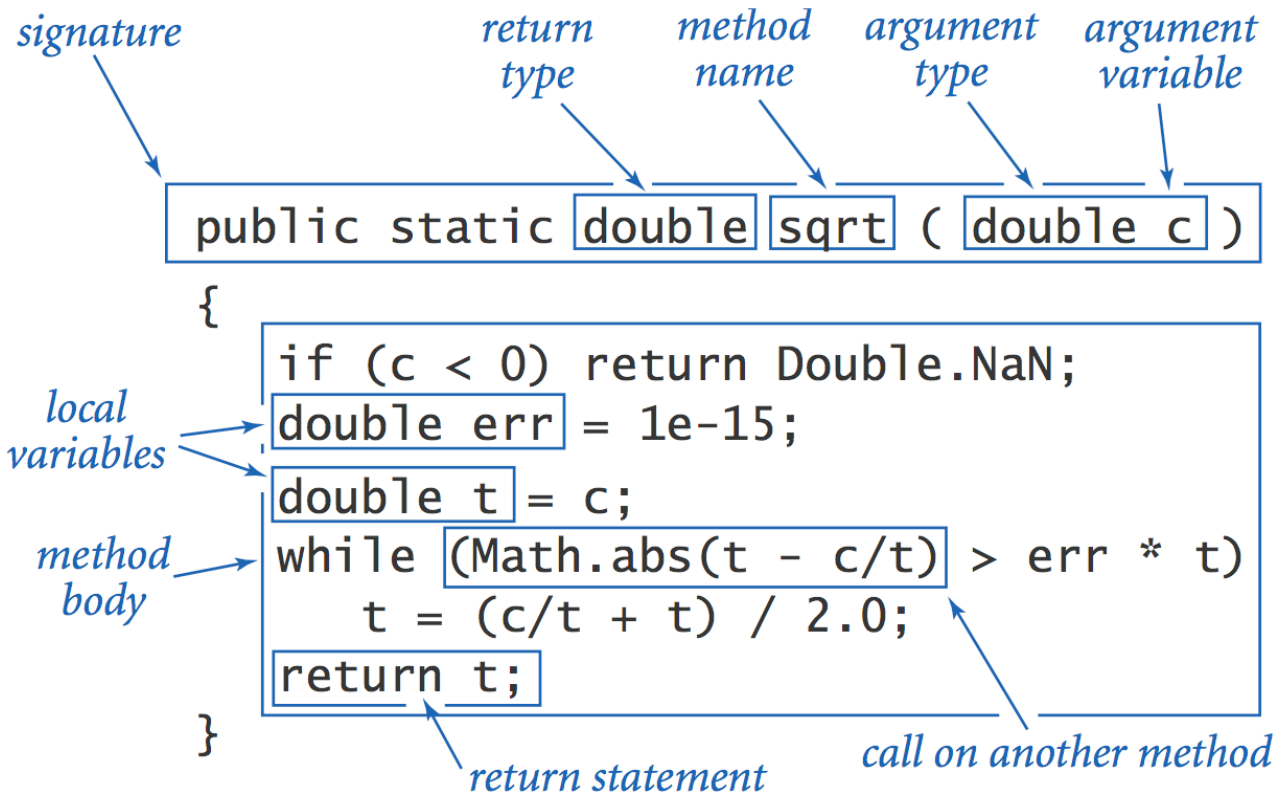
Examples.

- Built-in functions: `Math.random()`, `Math.abs()`, `Integer.parseInt()`.
- Our I/O libraries: `StdIn.readInt()`, `StdDraw.line()`, `StdAudio.play()`.
- User-defined functions: `main()`.

Anatomy of a Java Function

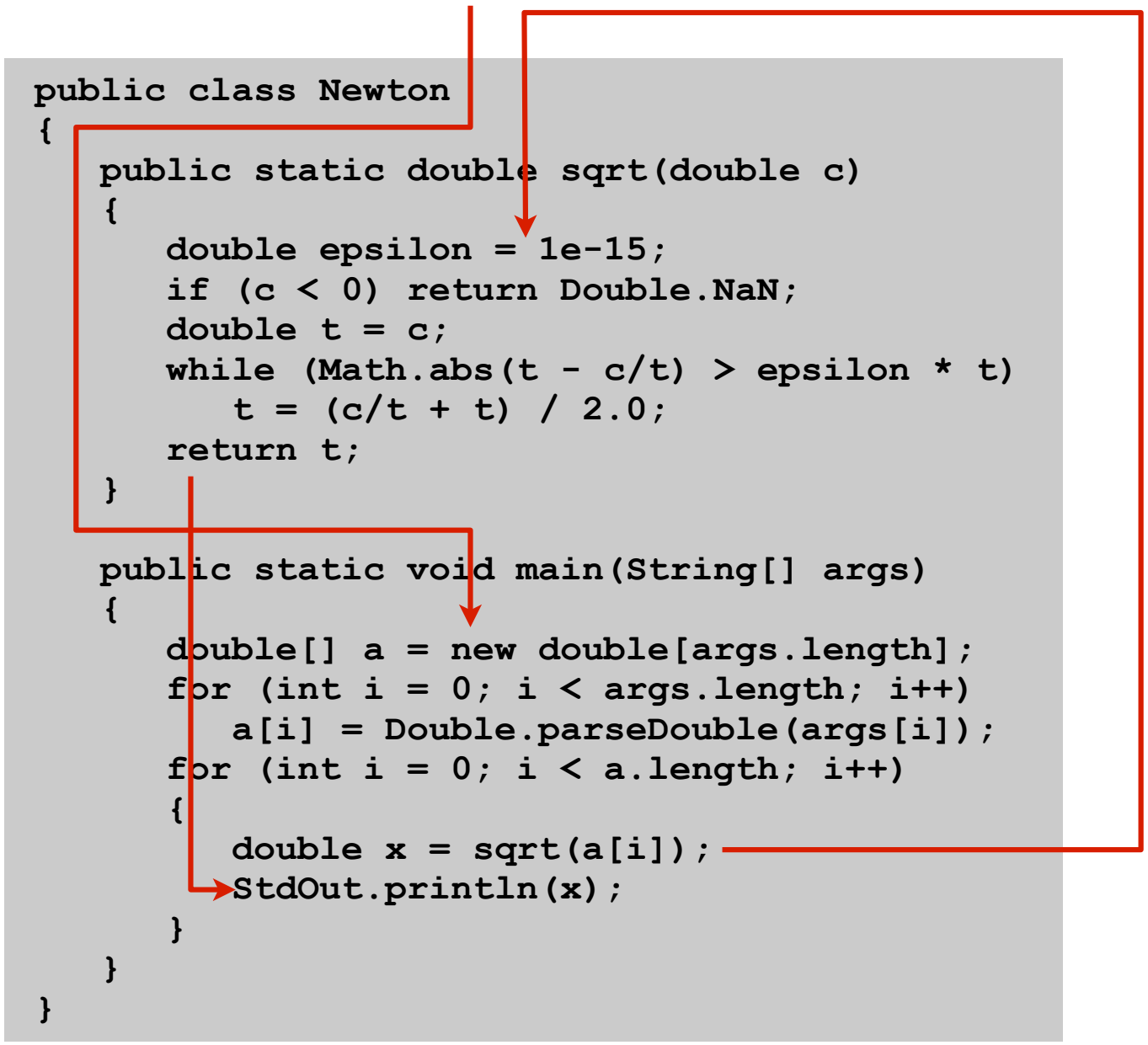
Java functions. Easy to write your own.





Flow of Control

Key point. Functions provide a new way to control the flow of execution.



```
public class Newton
{
   public static double sqrt(double c)
   {
      double epsilon = 1e-15;
      if (c < 0) return Double.NaN;
      double t = c;
      while (Math.abs(t - c/t) > epsilon * t)
         t = (c/t + t) / 2.0;
      return t;
   }

   public static void main(String[] args)
   {
      double[] a = new double[args.length];
      for (int i = 0; i < args.length; i++)
         a[i] = Double.parseDouble(args[i]);
      for (int i = 0; i < a.length; i++)
      {
         double x = sqrt(a[i]);
         StdOut.println(x);
      }
   }
}
```

Flow of Control

Key point. Functions provide a new way to control the flow of execution.

Summary of what happens when a function is called:

- Control transfers to the function code.
- Argument variables are assigned the values given in the call.
- Function code is executed.
- Return value is assigned in place of the function name in the calling code.
- Control transfers back to the calling code.

Note. This method (standard in Java) is known as "pass by value".

other languages may use different methods

Scope

Scope (of a name). The code that can refer to that name.

Def. A variable's scope is code following the declaration in its block.

```
public class Newton
{
   public static double sqrt(double c)
   {
      double epsilon = 1e-15;
      if (c < 0) return Double.NaN;
      double t = c;
      while (Math.abs(t - c/t) > epsilon * t)
         t = (c/t + t) / 2.0;
      return t;
   }

   public static void main(String[] args)
   {
      double[] a = new double[args.length];
      for (int i = 0; i < args.length; i++)
         a[i] = Double.parseDouble(args[i]);
      for (int i = 0; i < a.length; i++)
         System.out.println(sqrt(a[i]));
   }
}
```

scope of **c**

scope of **epsilon**

scope of **t**

two different variables with the same name **i** each with two lines of scope

scope of **a**

Best practice: declare variables so as to limit their scope.

Function Call Trace

```
public class Newton
{
   public static double sqrt(double c)
   {
      double epsilon = 1e-15;
      if (c < 0) return Double.NaN;
      double t = c;
      while (Math.abs(t - c/t) > epsilon * t)
         t = (c/t + t) / 2.0;
      return t;
   }

   public static void main(String[] args)
   {
      double[] a = new double[args.length];
      for (int i = 0; i < args.length; i++)
         a[i] = Double.parseDouble(args[i]);
      for (int i = 0; i < a.length; i++)
         System.out.println(sqrt(a[i]));
   }
}
```

TEQ on Functions 1.1

What happens when you compile and run the following code?

```
public class Cubes1
{
   public static int cube(int i)
   {
      int j = i * i * i;
      return j;
   }
   public static void main(String[] args)
   {
      int N = Integer.parseInt(args[0]);
      for (int i = 1; i <= N; i++)
         StdOut.println(i + " " + cube(i));
   }
}
```

TEQ on Functions 1.2

What happens when you compile and run the following code?

```
public class Cubes2
{
   public static int cube(int i)
   {
      int i = i * i * i;
      return i;
   }
   public static void main(String[] args)
   {
      int N = Integer.parseInt(args[0]);
      for (int i = 1; i <= N; i++)
         StdOut.println(i + " " + cube(i));
   }
}
```

TEQ on Functions 1.3

What happens when you compile and run the following code?

```
public class Cubes3
{
   public static int cube(int i)
   {
      i = i * i * i;
   }
   public static void main(String[] args)
   {
      int N = Integer.parseInt(args[0]);
      for (int i = 1; i <= N; i++)
         StdOut.println(i + " " + cube(i));
   }
}
```

TEQ on Functions 1.4

What happens when you compile and run the following code?

```
public class Cubes4
{
   public static int cube(int i)
   {
      i = i * i * i;
      return i;
   }
   public static void main(String[] args)
   {
      int N = Integer.parseInt(args[0]);
      for (int i = 1; i <= N; i++)
         StdOut.println(i + " " + cube(i));
   }
}
```

TEQ on Functions 1.5

What happens when you compile and run the following code?

```
public class Cubes5
{
   public static int cube(int i)
   {
      return i * i * i;
   }
   public static void main(String[] args)
   {
      int N = Integer.parseInt(args[0]);
      for (int i = 1; i <= N; i++)
         StdOut.println(i + " " + cube(i));
   }
}
```

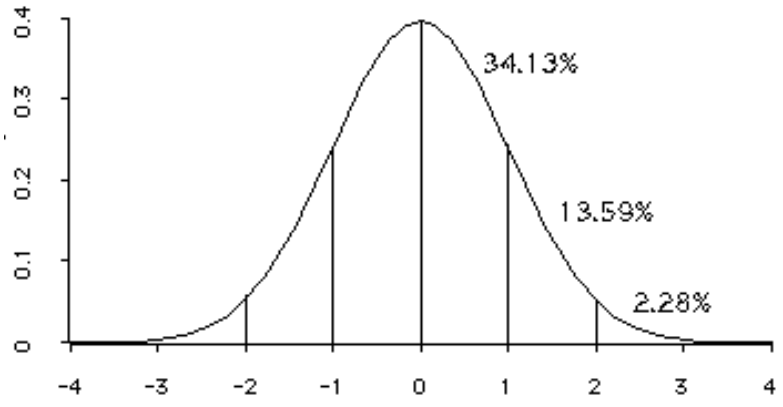
Example: Gaussian Distribution

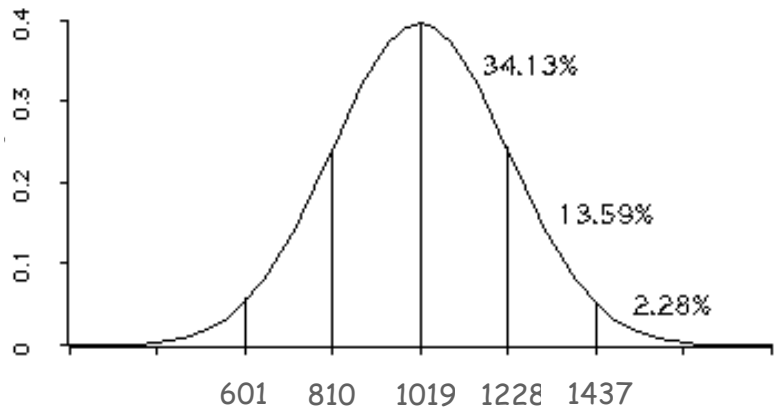
Gaussian Distribution

Standard Gaussian distribution.

- "Bell curve."
- Basis of most statistical analysis in social and physical sciences.

Ex. 2000 SAT scores follow a Gaussian distribution with mean μ = 1019, stddev σ = 209.





$$\phi(x) = \frac{1}{\sqrt{2\pi}}\, e^{-x^2/2}$$

$$\phi(x, \mu, \sigma) = \frac{1}{\sigma\sqrt{2\pi}}\, e^{-(x-\mu)^2/2\sigma^2}$$

$$= \phi\left(\frac{x-\mu}{\sigma}\right) / \sigma$$

Java Function for $\phi(x)$

Mathematical functions. Use built-in functions when possible; build your own when not available.

```
public class Gaussian
{

   public static double phi(double x)
   {
      return Math.exp(-x*x / 2) / Math.sqrt(2 * Math.PI);
   }

   public static double phi(double x, double mu, double sigma)
   {
      return phi((x - mu) / sigma) / sigma;
   }
}
```

$$\phi(x) = \frac{1}{\sqrt{2\pi}} e^{-x^2/2}$$

$$\phi(x, \mu, \sigma) = \phi\left(\frac{x - \mu}{\sigma}\right) / \sigma$$

Overloading. Functions with different signatures are different.

Multiple arguments. Functions can take any number of arguments.

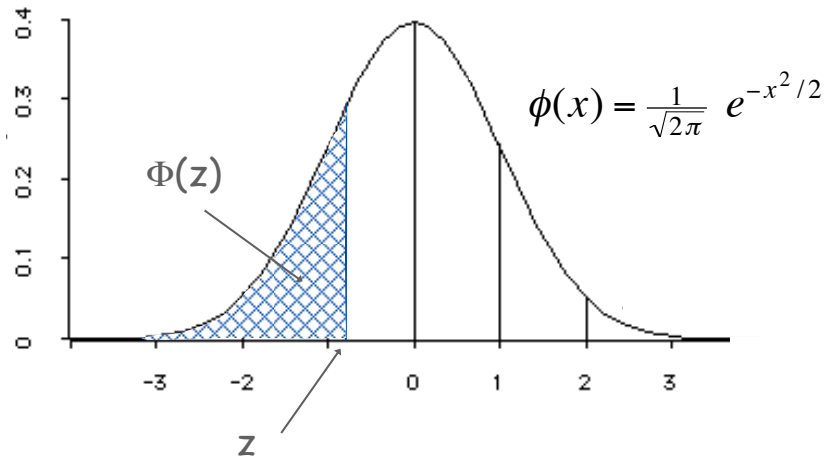
Calling other functions. Functions can call other functions.

library or user-defined

Gaussian Cumulative Distribution Function

Goal. Compute Gaussian cdf $\Phi(z)$.

Challenge. No "closed form" expression and not in Java library.



Taylor series

$$\Phi(z) = \int_{-\infty}^{z} \phi(x)dx$$

$$= \frac{1}{2} + \phi(z)\left(z + \frac{z^3}{3} + \frac{z^5}{3 \cdot 5} + \frac{z^7}{3 \cdot 5 \cdot 7} + \cdots\right)$$

Bottom line. 1,000 years of mathematical formulas at your fingertips.

Java function for $\Phi(z)$

```
public class Gaussian
{

   public static double phi(double x)
      // as before

   public static double Phi(double z)
   {
      if (z < -8.0) return 0.0;
      if (z >  8.0) return 1.0;
      double sum = 0.0, term = z;
      for (int i = 3; sum + term != sum; i += 2) {
         sum  = sum + term;
         term = term * z * z / i;
      }
      return 0.5 + sum * phi(z);
   }

   public static double Phi(double z, double mu, double sigma)

   {

      return Phi((z - mu) / sigma);

   }
}
```

accurate with absolute error less than $8 * 10^{-16}$

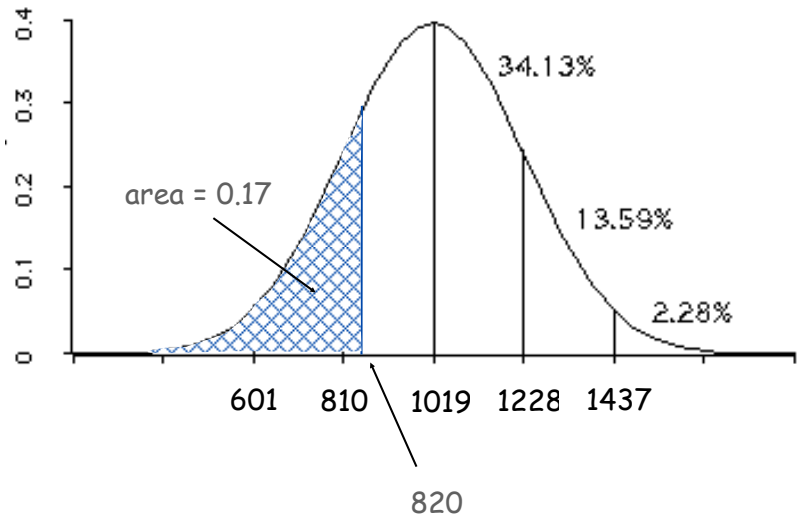
$$\Phi(z,\ \mu,\ \sigma) = \int_{-\infty}^{z} \phi(z, \mu, \sigma) = \Phi((z-\mu) / \sigma)$$

SAT Scores

Q. NCAA requires at least 820 for Division I athletes.

What fraction of test takers in 2000 do not qualify?

A. $\Phi(820, \mu, \sigma) \approx 0.17051$. [approximately 17%]




```
double fraction = Gaussian.Phi(820, 1019, 209);
```

Gaussian Distribution

Q. Why relevant in mathematics?

A. Central limit theorem: under very general conditions, average of a set of variables tends to the Gaussian distribution.

Q. Why relevant in the sciences?

A. Models a wide range of natural phenomena and random processes.

- Weights of humans, heights of trees in a forest.
- SAT scores, investment returns.

Caveat.

> Everybody believes in the exponential law of errors: the experimenters, because they think it can be proved by mathematics; and the mathematicians, because they believe it has been established by observation. - M. Lippman in a letter to H. Poincaré

Building Functions

Functions enable you to build a new layer of abstraction.

- Takes you beyond pre-packaged libraries.
- You build the tools you need: `Gaussian.phi()`, …

Process.

- Step 1: identify a useful feature.
- Step 2: implement it.
- Step 3: use it.

- Step 3': re-use it in any of your programs.

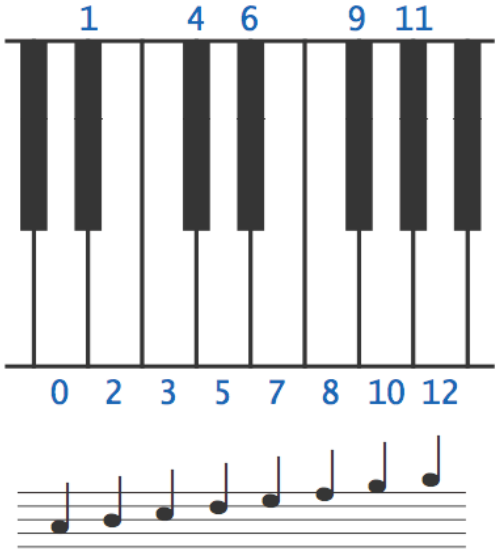
Digital Audio

Crash Course in Sound

Sound. Perception of the vibration of molecules in our eardrums.

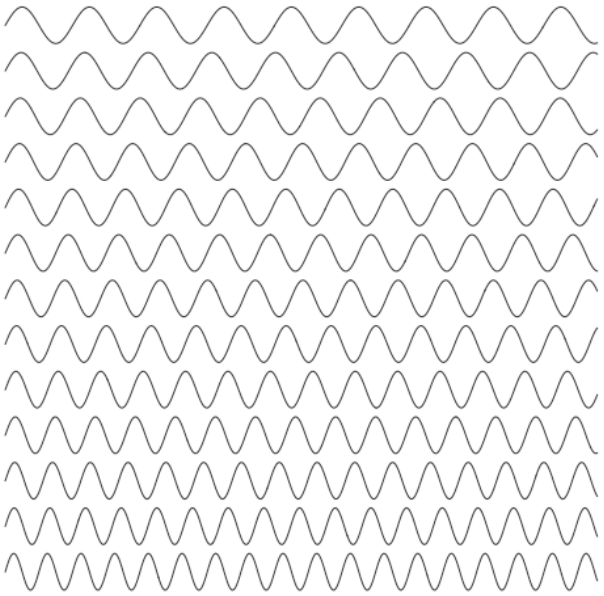
Concert A. Sine wave, scaled to oscillated at 440Hz.

Other notes. 12 notes on chromatic scale, divided logarithmically.



note	*i*	*frequency*
A	0	440.00
A♯ *or* B♭	1	466.16
B	2	493.88
C	3	523.25
C♯ *or* D♭	4	554.37
D	5	587.33
D♯ *or* E♭	6	622.25
E	7	659.26
F	8	698.46
F♯ *or* G♭	9	739.99
G	10	783.99
G♯ *or* A♭	11	830.61
A	12	880.00

$440 \times 2^{i/12}$



Notes, numbers, and waves

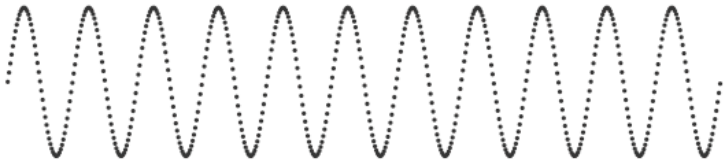
Digital Audio

Sampling. Represent curve by sampling it at regular intervals.

5,512 *samples/second, 137 samples*

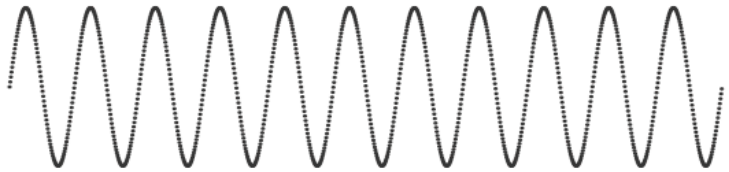
11,025 *samples/second, 275 samples*

22,050 *samples/second, 551 samples*



44,100 *samples/second, 1,102 samples*

audio CD



$$y(i) = \sin\left(\frac{2\pi \cdot i \cdot 440}{44{,}100}\right)$$

Musical Tone Function

Musical tone. Create a music tone of a given frequency and duration.

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

$$y(i) = \sin\left(\frac{2\pi \cdot i \cdot hz}{44{,}100}\right)$$

Remark. Can use arrays as function return value and/or argument.

Digital Audio in Java

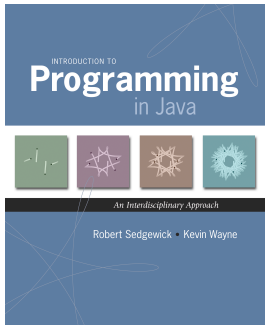
Standard audio. Library for playing digital audio.

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 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*

Concert A. Play concert A for `1.5` seconds using `StdAudio`.

library developed for this course (also broadly useful)

```
double[] a = tone(440, 1.5);
StdAudio.play(a);
```



Warmup: Musical Tone

Musical tone. Create a music tone of a given frequency and duration.

```
public class Tone
{
   public static void main(String[] args)
   {
      int sps = 44100;
      double hz       = Double.parseDouble(args[0]);
      double duration = Double.parseDouble(args[1]);
      int N = (int) (sps * duration);
      double[] a = new double[N+1];
      for (int i = 0; i <= N; i++)
         a[i] = Math.sin(2 * Math.PI * i * hz / sps);
      StdAudio.play(a);
   }
}
```

$$y(i) = \sin\left(\frac{2\pi \cdot i \cdot hz}{44{,}100}\right)$$

```
% java Tone 440 1.5
[ concert A for 1.5 seconds]
```

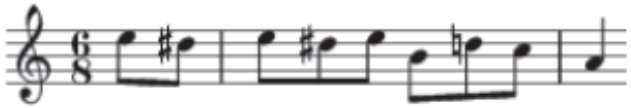
Play That Tune

Goal. Play pitches and durations from standard input on standard audio.

```
public class PlayThatTune
{
   public static void main(String[] args)
   {
      int sps = 44100;
      while (!StdIn.isEmpty())
      {
            int pitch = StdIn.readInt();
            double duration = StdIn.readDouble();
            double hz = 440 * Math.pow(2, pitch / 12.0);
            int N = (int) (sps * duration);
            double[] a = new double[N+1];
            for (int i = 0; i <= N; i++)
               a[i] = Math.sin(2 * Math.PI * i * hz / sps);
            StdAudio.play(a);
      }
   }
}
```

```
% more elise.txt
7 .125
6 .125
7 .125
6 .125
7 .125
2 .125
5 .125
3 .125
0 .25
...
```

```
% java PlayThatTune < elise.txt
```



TEQ on Functions 2

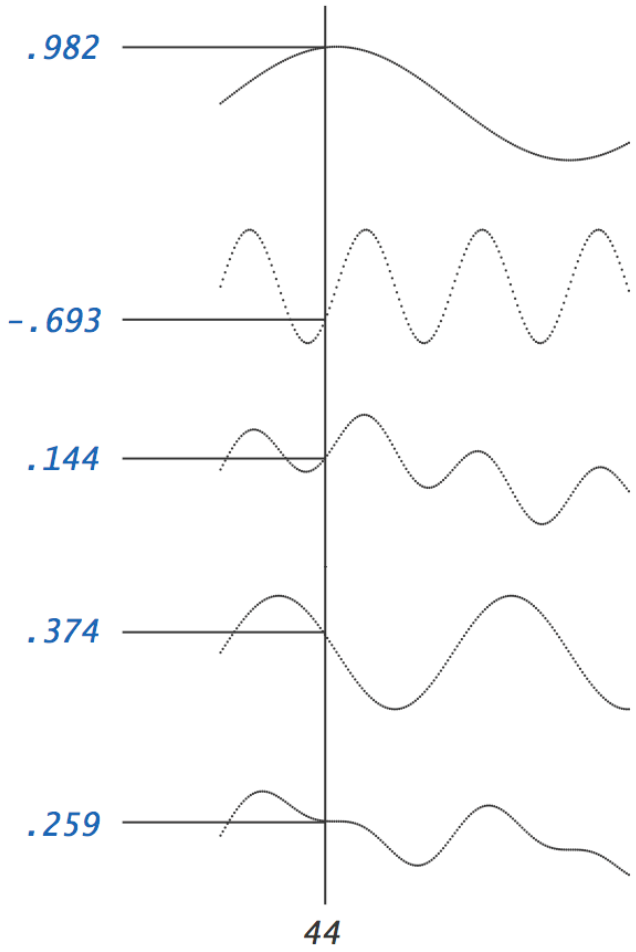
What sound does the following program produce?

```
public static double[] tone(double hz, double seconds)
{
   int SAMPLE_RATE = 44100;
   int N = (int) (seconds * SAMPLE_RATE);
   double[] a = new double[N+1];
   for (int i = 0; i <= N; i++)
      a[i] = Math.Random();
   return a;
}
```

Harmonics

Concert A with harmonics. Obtain richer sound by adding tones one octave above and below concert A.

880 Hz 220 Hz 440 Hz




```
lo = tone(220, .0041);
lo[44] = .982
```

```
hi = tone(880, .0041);
hi[44] = -.693
```

```
h = sum(hi, lo, .5, .5);
h[44] = .5*lo[44]+.5*hi[44];
      = .5*.982 - .5*.693 = .144
```

```
A = tone(440, .0041);
A[44] = .374
```

```
sum(A, h, .5, .5);
A[44] + h[44] = .5*.144 + .5*.374
              = .259
```

Harmonics

```
public class PlayThatTune
{
   // Return weighted sum of two arrays.
   public static double[] sum(double[] a, double[] b, double awt, double bwt) {
      double[] c = new double[a.length];
      for (int i = 0; i < a.length; i++)
         c[i] = a[i]*awt + b[i]*bwt;
      return c;
   }

   // Return a note of given pitch and duration.
   public static double[] note(int pitch, double duration) {
      double hz = 440.0 * Math.pow(2, pitch / 12.0);
      double[] a  = tone(1.0 * hz, duration);
      double[] hi = tone(2.0 * hz, duration);
      double[] lo = tone(0.5 * hz, duration);
      double[] h  = sum(hi, lo, .5, .5);
      return sum(a, h, .5, .5);
   }

   public static double[] tone(double hz, double t)
      // see previous slide

   public static void main(String[] args)
      // see next slide
}
```

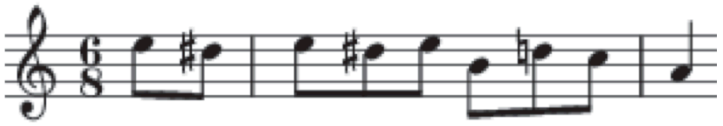
Harmonics

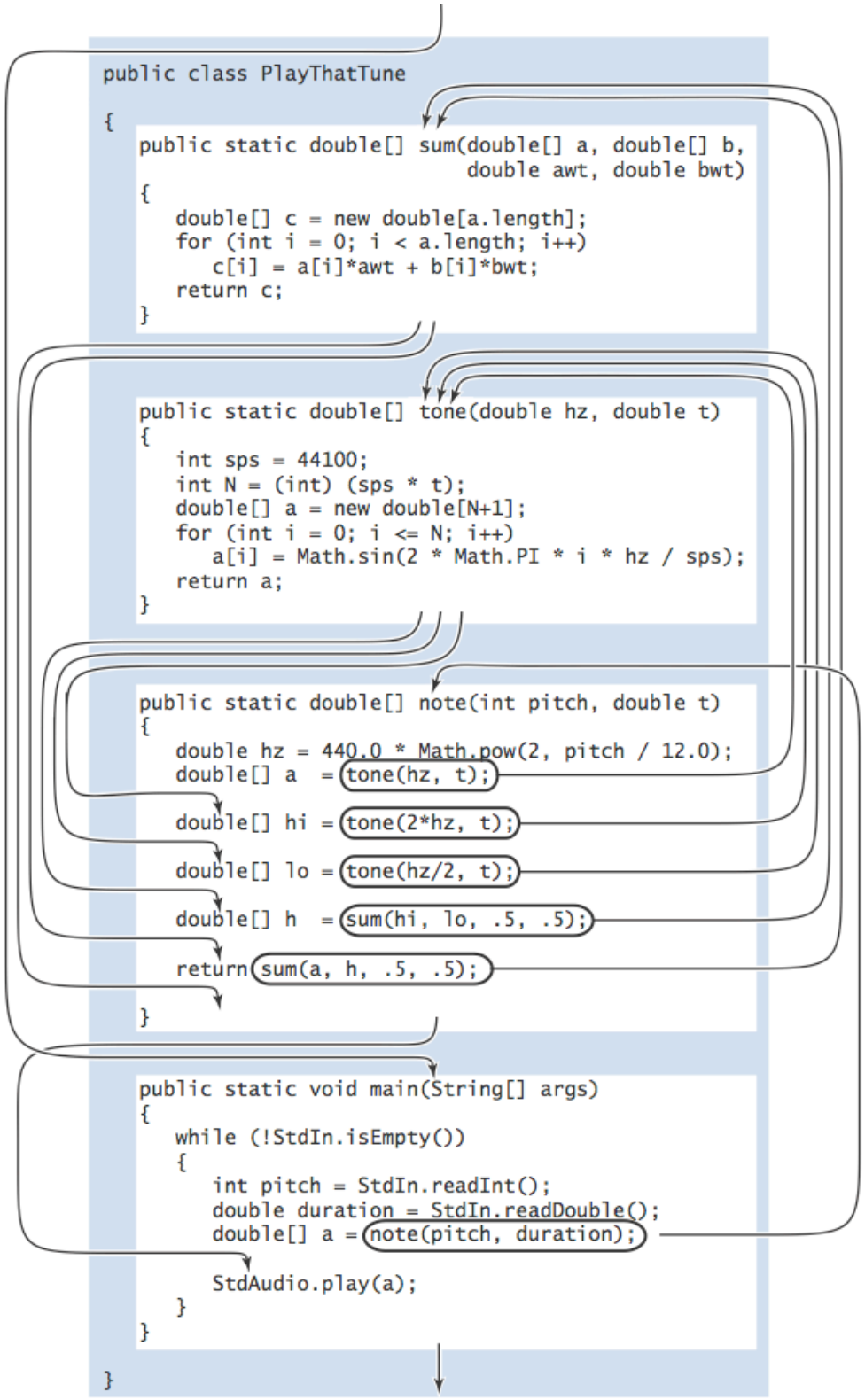
Play that tune. Read in pitches and durations from standard input, and play using standard audio.

```
public static void main(String[] args)
{
   while (!StdIn.isEmpty())
   {
      int pitch = StdIn.readInt();
      double duration = StdIn.readDouble();
      double[] a = note(pitch, duration);
      StdAudio.play(a);
   }
}
```

```
% more elise.txt
7 .125
6 .125
7 .125
6 .125
7 .125
2 .125
5 .125
3 .125
0 .25
```

```
% java PlayThatTune < elise.txt
```




public class PlayThatTune
{
public static double[] sum(double[] a, double[] b,
double awt, double bwt)
{
double[] c = new double[a.length];
for (int i = 0; i < a.length; i++)
c[i] = a[i]*awt + b[i]*bwt;
return c;
}
public static double[] tone(double hz, double t)
{
int sps = 44100;
int N = (int) (sps * t);
double[] a = new double[N+1];
for (int i = 0; i <= N; i++)
a[i] = Math.sin(2 * Math.PI * i * hz / sps);
return a;
}
public static double[] note(int pitch, double t)
{
double hz = 440.0 * Math.pow(2, pitch / 12.0);
double[] a = tone(hz, t);
double[] hi = tone(2*hz, t);
double[] lo = tone(hz/2, t);
double[] h = sum(hi, lo, .5, .5);
return sum(a, h, .5, .5);
}
public static void main(String[] args)
{
while (!StdIn.isEmpty())
{
int pitch = StdIn.readInt();
double duration = StdIn.readDouble();
double[] a = note(pitch, duration);
StdAudio.play(a);
}
}
}

2.2 Libraries and Clients

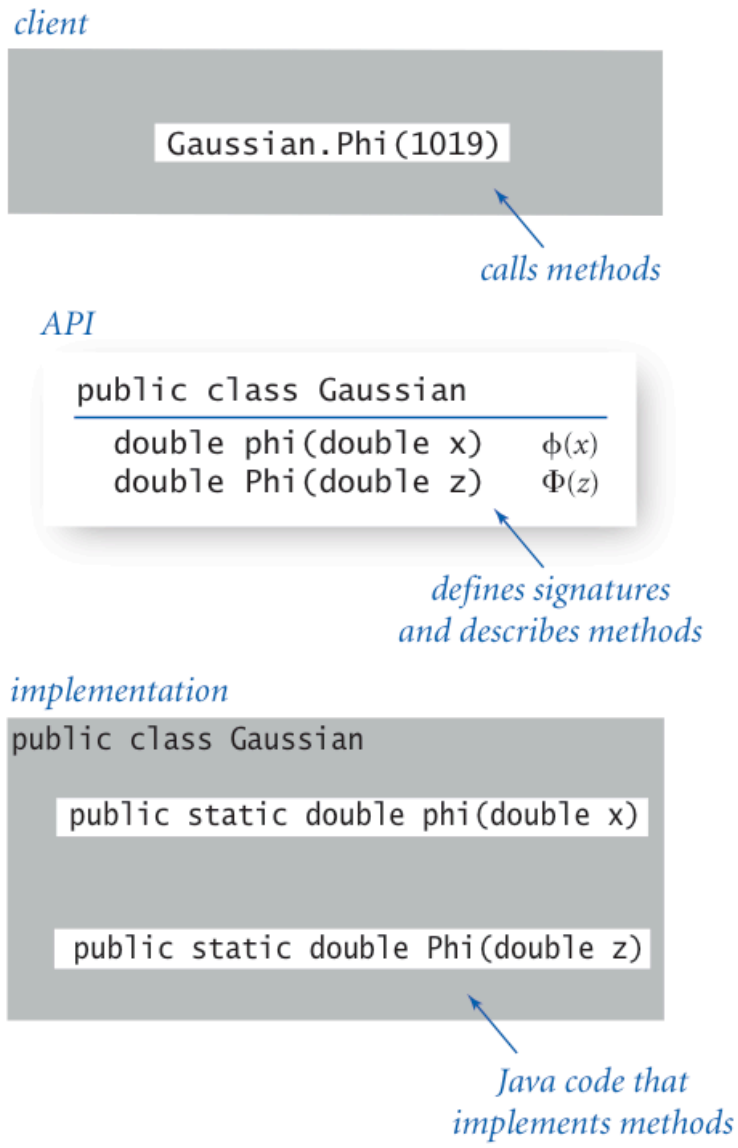
Libraries

Library. A module whose methods are primarily intended for use by many other programs.

Client. Program that calls a library.

API. Contract between client and implementation.

Implementation. Program that implements the methods in an API.



Standard Random

Standard random. Our library to generate pseudo-random numbers.

public class StdRandom	
int uniform(int N)	*integer between* 0 *and* N-1
double uniform(double lo, double hi)	*real between* lo *and* hi
boolean bernoulli(double p)	true *with probability* p
double gaussian()	*normal, mean* 0, *standard deviation* 1
double gaussian(double m, double s)	*normal, mean* m, *standard deviation* s
int discrete(double[] a)	i *with probability* a[i]
void shuffle(double[] a)	*randomly shuffle the array* a[]

```
int getRandomNumber()
{
   return 4;  // chosen by fair dice roll.
              // guaranteed to be random.
}
```

Standard Random

```
public class StdRandom
{
   // between a and b
   public static double uniform(double a, double b)
   {
      return a + Math.random() * (b-a);
   }

   // between 0 and N-1
   public static int uniform(int N)
   {
      return (int) (Math.random() * N);
   }

   // true with probability p
   public static boolean bernoulli(double p)
   {
      return Math.random() < p;
   }

   // gaussian with mean = 0, stddev = 1
   public static double gaussian()
      /* see Exercise 1.2.27 */

   // gaussian with given mean and stddev
   public static double gaussian(double mean, double stddev)
   {
      return mean + (stddev * gaussian());
   }

   ...
}
```

Unit Testing

Unit test. Include `main()` to test each library.

```
public class StdRandom
{
   ...
   public static void main(String[] args)
   {
      int N = Integer.parseInt(args[0]);
      for (int i = 0; i < N; i++) {
         StdOut.printf(" %2d " , uniform(100));
         StdOut.printf("%8.5f ", uniform(10.0, 99.0));
         StdOut.printf("%5b "  , bernoulli(.5));
         StdOut.printf("%7.5f ", gaussian(9.0, .2));
         StdOut.println();
      }
   }
}
```

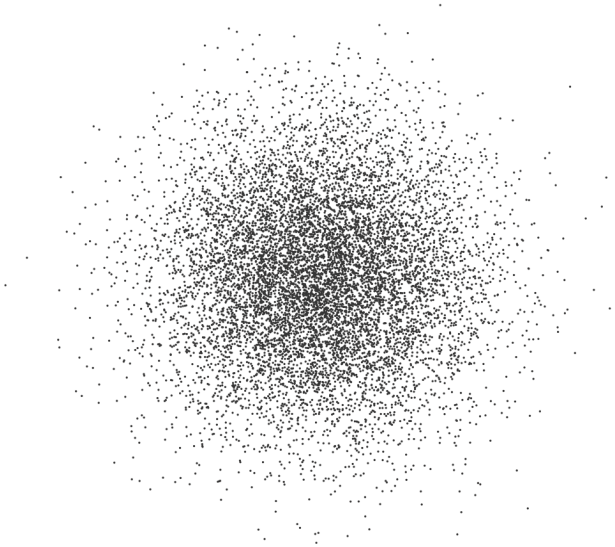
```
% java StdRandom 5
 61 21.76541  true 9.30910
 57 43.64327 false 9.42369
 31 30.86201  true 9.06366
 92 39.59314  true 9.00896
 36 28.27256 false 8.66800
```

Using a Library

```
public class RandomPoints
{
   public static void main(String args[])
   {
      int N = Integer.parseInt(args[0]);
      for (int i = 0; i < N; i++)
      {
         double x = StdRandom.gaussian(0.5, 0.2);
         double y = StdRandom.gaussian(0.5, 0.2);
         StdDraw.point(x, y);
      }
   }
}
```

use library name to invoke method

```
% javac RandomPoints.java
% java RandomPoints 10000
```



Standard Statistics

Ex. Library to compute statistics on an array of real numbers.

public class StdStats	
double max(double[] a)	*largest value*
double min(double[] a)	*smallest value*
double mean(double[] a)	*average*
double var(double[] a)	*sample variance*
double stddev(double[] a)	*sample standard deviation*
double median(double[] a)	*median*
void plotPoints(double[] a)	*plot points at* (i, a[i])
void plotLines(double[] a)	*plot lines connecting points at* (i, a[i])
void plotBars(double[] a)	*plot bars to points at* (i, a[i])

$$\mu = \frac{a_0 + a_1 + \cdots + a_{n-1}}{n}, \qquad \sigma^2 = \frac{(a_0 - \mu)^2 + (a_1 - \mu)^2 + \cdots + (a_{n-1} - \mu)^2}{n - 1}$$

mean *sample variance*

Standard Statistics

Ex. Library to compute statistics on an array of real numbers.

```
public class StdStats
{
   public static double max(double[] a)
   {
      double max = Double.NEGATIVE_INFINITY;
      for (int i = 0; i < a.length; i++)
         if (a[i] > max) max = a[i];
      return max;
   }

   public static double mean(double[] a)
   {
      double sum = 0.0;
      for (int i = 0; i < a.length; i++)
         sum = sum + a[i];
      return sum / a.length;
   }

   public static double stddev(double[] a)
     // see text

}
```

Modular Programming

Modular Programming

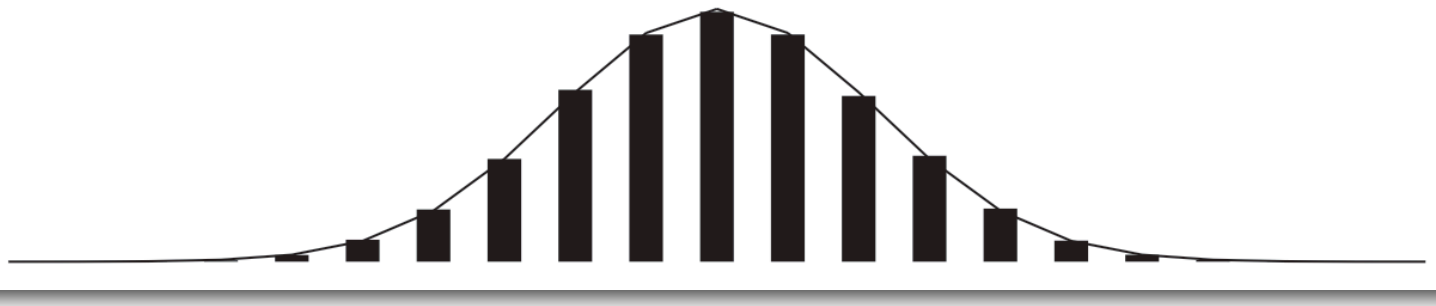
Modular programming.

- Divide program into self-contained pieces.
- Test each piece individually.
- Combine pieces to make program.

Ex. Flip N coins. How many heads?

- Read arguments from user.
- Flip one fair coin.
- Flip N fair coins and count number of heads.
- Repeat simulation, counting number of times each outcome occurs.
- Plot histogram of empirical results.
- Compare with theoretical predictions.

```
% java Bernoulli 20 100000
```



Bernoulli Trials

```
public class Bernoulli
{
   public static int binomial(int N)
   {
      int heads = 0;
      for (int j = 0; j < N; j++)
         if (StdRandom.bernoulli(0.5)) heads++;
      return heads;
   }

   public static void main(String[] args)
   {
      int N = Integer.parseInt(args[0]);
      int T = Integer.parseInt(args[1]);

      int[] freq = new int[N+1];
      for (int i = 0; i < T; i++)
         freq[binomial(N)]++;

      double[] normalized = new double[N+1];
      for (int i = 0; i <= N; i++)
         normalized[i] = (double) freq[i] / T;
      StdStats.plotBars(normalized);

      double mean = N / 2.0, stddev = Math.sqrt(N) / 2.0;
      double[] phi  = new double[N+1];
      for (int i = 0; i <= N; i++)
         phi[i] = Gaussian.phi(i, mean, stddev);
      StdStats.plotLines(phi);
  }
}
```

flip N fair coins; return # heads

perform T trials of N coin flips each

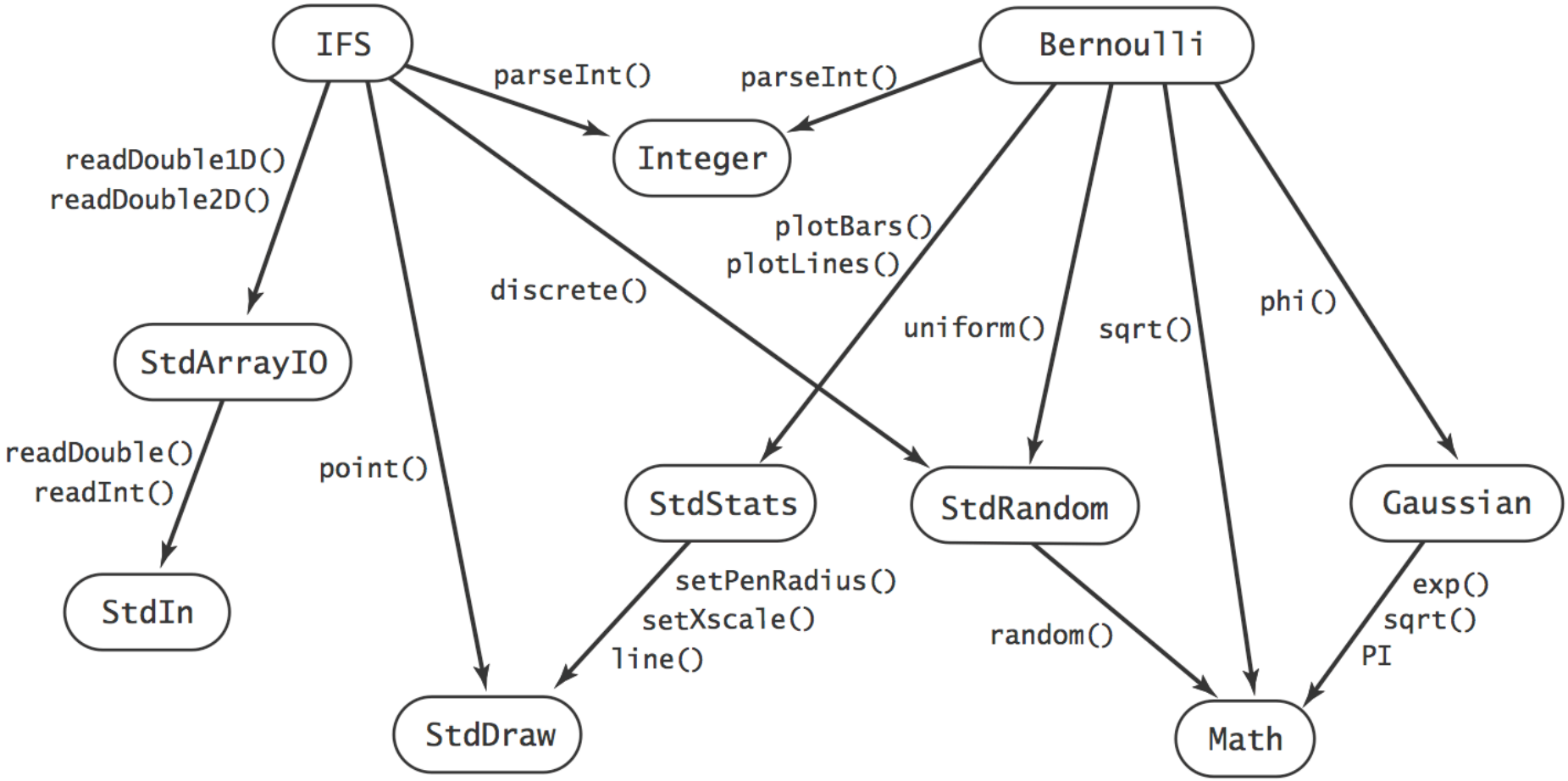
compute frequency of occurrence of each count of heads

plot histogram of frequencies

theoretical prediction

Dependency Graph

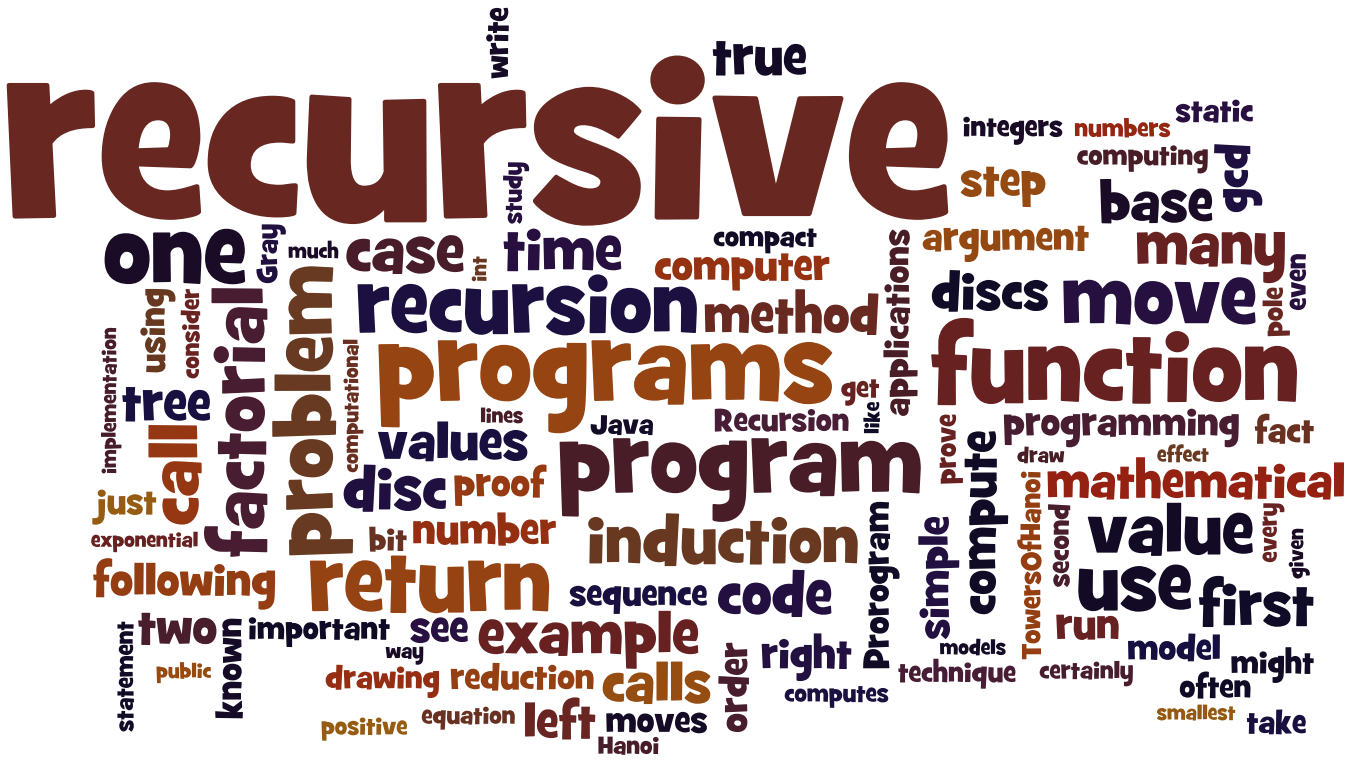
Modular programming. Build relatively complicated program by combining several small, independent, modules.



Libraries

Why use libraries?

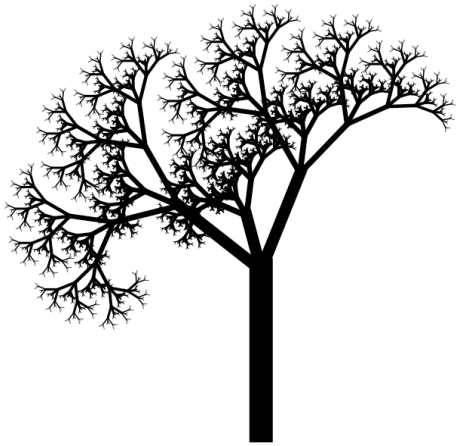
- Makes code easier to understand.
- Makes code easier to debug.
- Makes code easier to maintain and improve.
- Makes code easier to reuse.


recursive
true
write
integers
numbers
static
computing
gcd
step
base
study
one
Gray
much
case
int
time
compact
computer
argument
many
factorial
problem
recursion
method
discs
move
pole
even
using
consider
implementation
programs
function
tree
get
applications
lines
Java
Recursion
programming
fact
call
values
program
like
prove
draw
effect
computational
disc
proof
compute
mathematical
just
bit
number
induction
simple
second
TowersOfHanoi
every
value
exponential
following
return
sequence
code
Prorogram
use
first
given
two
important
see
example
run
known
way
right
models
model
might
statement
public
drawing
reduction
calls
technique
certainly
often
order
computes
positive
equation
left
moves
smallest
take
Hanoi

2.3 Recursion

Overview

What is recursion? When one function calls itself directly or indirectly.



Why learn recursion?

- New mode of thinking.
- Powerful programming paradigm.

Many computations are naturally self-referential.

- Binary search, mergesort, FFT, GCD.
- Linked data structures.
- A folder contains files and other folders.

Closely related to mathematical induction.

M. C. Escher, 1956

Greatest Common Divisor

Gcd. Find largest integer that evenly divides into p and q.

Ex. gcd(4032, 1272) = 24.

$$4032 = 2^6 \times 3^2 \times 7^1$$
$$1272 = 2^3 \times 3^1 \times 53^1$$
$$\text{gcd} = 2^3 \times 3^1 = 24$$

Applications.

- Simplify fractions: 1272/4032 = 53/168.
- RSA cryptosystem.

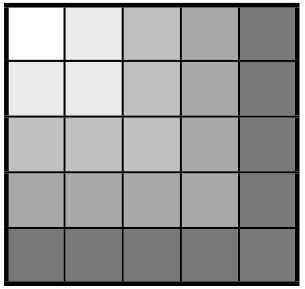
Mathematical Induction

Mathematical induction. Prove a statement involving an integer N by

- base case: Prove it for some specific N (usually 0 or 1).
- induction step: Assume it to be true for all positive integers less than N, use that fact to prove it for N.

Ex. Sum of the first N odd integers is N^2.

$$
\begin{aligned}
1 &= 1 \\
1 + 3 &= 4 \\
1 + 3 + 5 &= 9 \\
1 + 3 + 5 + 7 &= 16 \\
1 + 3 + 5 + 7 + 9 &= 25 \\
\ldots &\ \ldots
\end{aligned}
$$



Base case: True for N = 1.

Induction step:

- Let T(N) be the sum of the first N odd integers: 1 + 3 + 5 + ... + (2N - 1).
- Assume that $T(N-1) = (N-1)^2$.
- $T(N) = T(N-1) + (2N - 1)$

  $= (N-1)^2 + (2N - 1)$

  $= N^2 - 2N + 1 + (2N - 1)$

  $= N^2$

Recursive Program

Recursive Program. Implement a function having integer arguments by

- base case: Implementing it for some specific values of the arguments.
- reduction step: Assume the function works for smaller values of its arguments and use it to implement it for the given values.

Ex. gcd(p, q).

Base case: gcd(p, 0) = p.

Reduction step: gcd(p, q) = gcd(p, p-q) if p-q > 0

= gcd(p, p-2q) if p-2q > 0

...

= gcd(p, p % q)

Greatest Common Divisor

GCD. Find largest integer that evenly divides into p and q.

Euclid's algorithm. [Euclid 300 BCE]

$$\gcd(p, q) = \begin{cases} p & \text{if } q = 0 \\ \gcd(q, p \ \% \ q) & \text{otherwise} \end{cases}$$

← base case

← reduction step, converges to base case

```
gcd(4032, 1272)   = gcd(1272, 216)
                  = gcd(216, 192)
                  = gcd(192, 24)
                  = gcd(24, 0)
                  = 24.
```

4032 = 3 × 1272 + 216

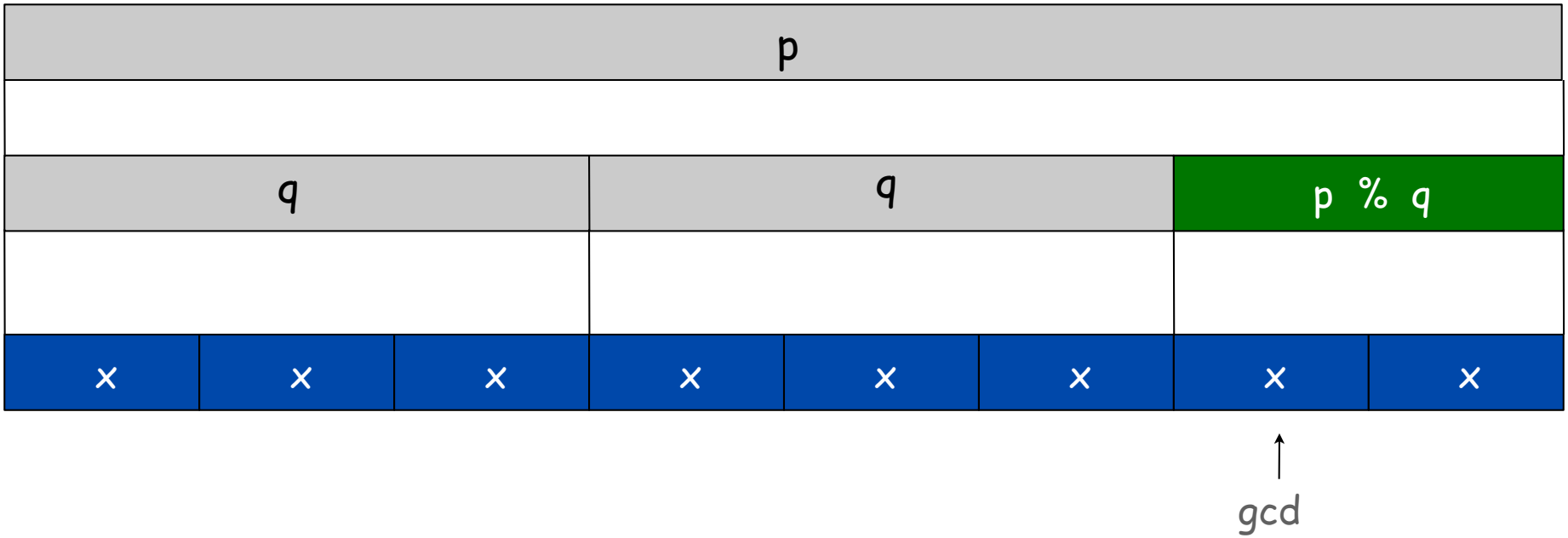
Euclid's Algorithm

GCD. Find largest integer d that evenly divides into p and q.

$$\gcd(p, q) = \begin{cases} p & \text{if } q = 0 \\ \gcd(q, p \ \% \ q) & \text{otherwise} \end{cases}$$

← base case

← reduction step, converges to base case



p = 8x
q = 3x

gcd(p, q) = gcd(3x, 2x) = x

Euclid's Algorithm

GCD. Find largest integer d that evenly divides into p and q.

$$\gcd(p, q) = \begin{cases} p & \text{if } q = 0 \\ \gcd(q, p \ \% \ q) & \text{otherwise} \end{cases}$$

← base case

← reduction step, converges to base case

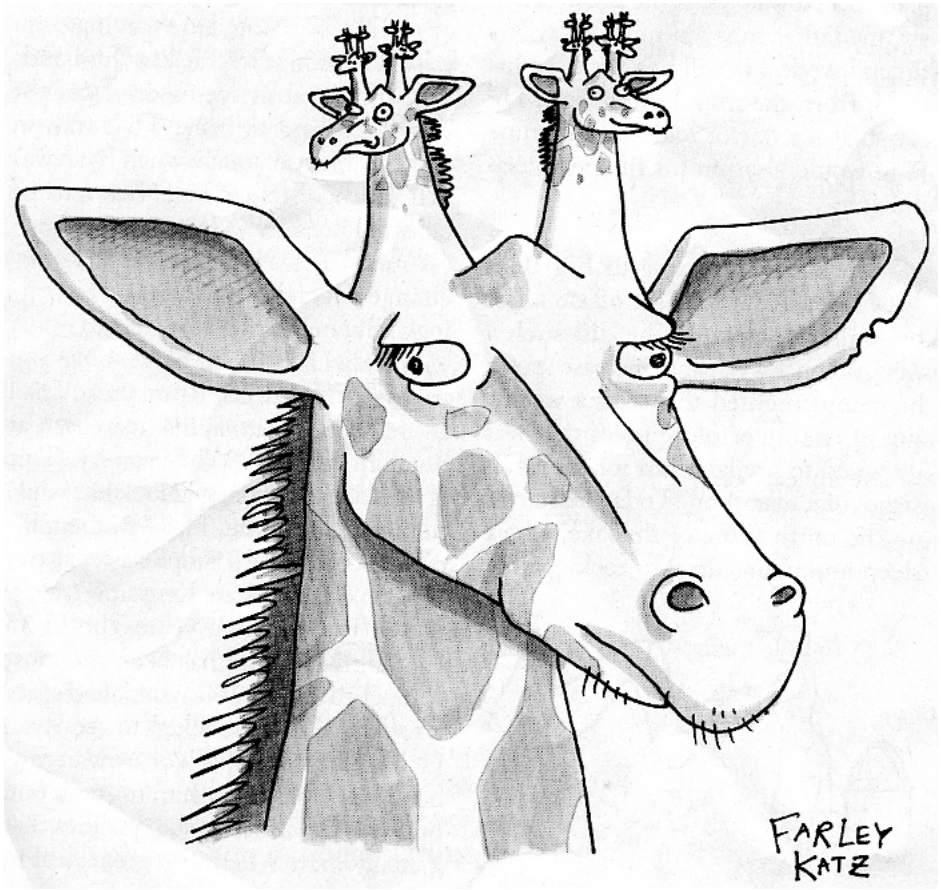
Recursive program

```
public static int gcd(int p, int q)
{
   if (q == 0) return p;
   else return gcd(q, p % q);
}
```

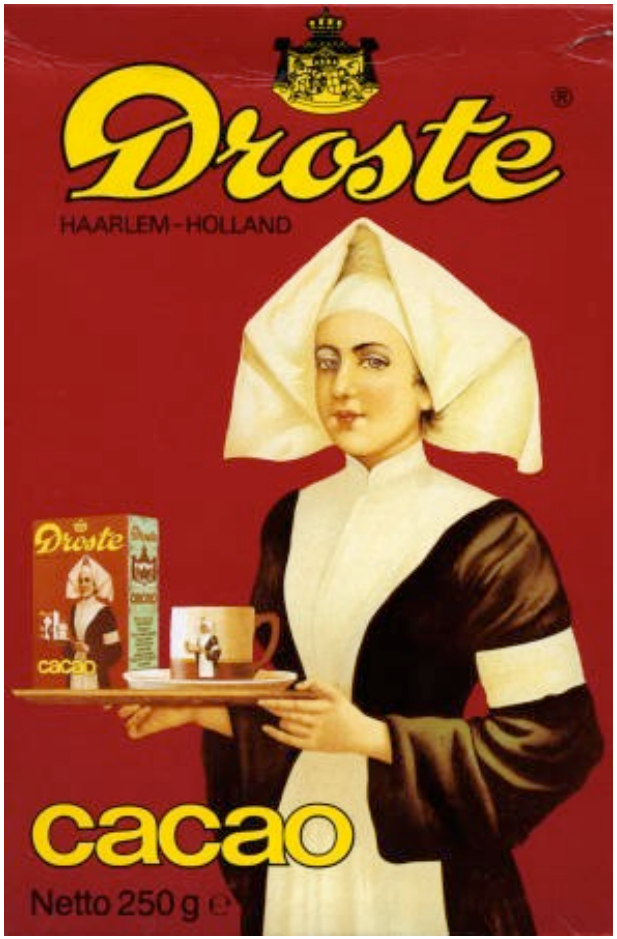
← base case

← reduction step

Recursive Graphics



New Yorker Magazine, August 11, 2008



Design Life Now at the Cooper-Hewitt National Design Museum includes this toy from the New York company Kidrobot.

Fruits of Design, Certified Organic

It's Triennial time at the Cooper-Hewitt National Design Museum. This means that the former Andrew Carnegie mansion is up to its neck in mostly American design from the last three years. Like its predecessors, "Design Life Now," the museum's third National Design Triennial, is a crazed affair that illuminates a volatile, contradictory, ever-expanding field but fails to call it to order.

ROBERTA SMITH
ART REVIEW

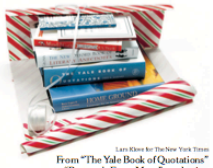
The exhibition has been organized by the Cooper-Hewitt curators Barbara Bloemink, Ellen Lupton and Matilda McQuaid and a guest, Brooke Hodge, a curator at the Museum of Contemporary Art, Los Angeles.

Once again the Triennial answers the question "What's design?" with the evasive catchall "What's not?" Covering so many bases so equivocally, it never gets around to tackling the weightier questions of "What is good design?" or, more to the point, "What is design good for?" It refuses to take sides on the issue of whether design should aim for social or environmental benefit or serve a relatively decorative purpose. Still, the show's benefits are many, even if you have to work for them.

The displays here range from genius to schlock, delightful to dispiriting. They cover life-extending innovations, completely frivolous reiterations of received ideas (far too many of which trace to Surrealism) and more varieties of recycling than you can easily count. Fashion, building materials, furniture, toys, theatrical sets, jewelry and textiles, medical and military hardware, all qualify as design according to this exhibition.

The main point comes across loud and clear: design permeates every aspect of contemporary life. Everything that exists is designed, whether natural or cultural. And while all of nature's designs are intelligent, whether you go by Darwin or the Bible, the human kind are much

Continued on Page 51



Lars Klove for The New York Times

From "The Yale Book of Quotations" to "Postcards From Mars," a selection of the best holiday books.

The Gifts to Open Again and Again

I've made my list, and I'm checking it twice. It's a list of the qualities that make the ideal holiday book, and after carefully considering the books of Christmas past, I have come up with some guidelines. A gift book should either be no surprise or a big surprise: the one you always wanted or the one you never knew you wanted. It should either be expensive and large, or cheap and small. It should be high-minded or totally frivolous. And no matter what, it should not require sustained attention, which is impossible during the yuletide season. My gift selections, chosen entirely at random but with exquisite taste, satisfy at least two of these requirements.

WILLIAM GRIMES
BOOKS

Let's open the big presents first. The season's whopper, in every way, is "New York 2100," the fifth installment in Robert A. M. Stern's architectural history of New York. The series starts in 1880, when 10 stories qualified as a skyscraper, and has now caught up to the new millennium. Taken together, the volumes make an enormous, endlessly fascinating family scrapbook for New Yorkers, who can coo over baby pictures of the Flatiron Building and leaf forward, through many hundreds of pages and thousands of photographs, to the big, grown-up New York of the Lipstick Building, countless Trump projects and the new Tweed Courthouse.

At 1,520 pages and 10 pounds 12 ounces, "New York

Continued on Page 46

Serkan Özkaya

Black, White and Read All Over Over

By RANDY KENNEDY

In one of Jorge Luis Borges's best-known short stories, "Pierre Menard, Author of the Quixote," a 20th-century French writer sets out to compose a verbatim copy of Cervantes's 17th-century masterpiece simply because he thinks he can, originality perhaps not being all it's cracked up to be.

He manages two chapters word for word, a spontaneous duplicate that Borges's narrator finds to be "infinitely richer" than the original because it contains all manner of new meanings and inflections, wrenched as it is from its proper time and context.

When a young Turkish artist named Serkan Özkaya set out recently to practice his skills as a copyist — a scrivener, as he says — his goals were a little less ambitious than channeling Cervantes. He simply wanted to draw and see printed a faithful copy of all the type and pictures planned for a broadsheet page of this newspaper: this very page you are reading right now, which shows his version of the page you are reading right now, which shows his version of his version of the page you are reading right now, which . . .

Do not be alarmed: There has been no break in the space-time-newsprint continuum.

Serkan Özkaya's drawing of the page you are reading right now, showing his drawing of the page you are reading right now, showing . . .

Mr. Özkaya, a 33-year-old artist who lives and works in Istanbul, did not propose this exercise in handmade surrealism because of a particular love of calligraphy or newspapers or, for that matter, even drawing, which he admits he is not very good at. The

Continued on Page 51

Divine and Devotee Meet Across Hinges

WASHINGTON — For toothache, dial St. Apollonia ASAP. She'll bring relief in a flash. Keep St. Matthew, ex-banker, in mind in April; he'll help get your taxes in shape. Everyone knows that a prayer to St. Roch, protector from plague, is as good as a flu shot, and that lightning will never strike when St. Barbara's on the job.

HOLLAND COTTER
ART REVIEW

Most important, for dire and intangible problems — moral confusion, inconsolable grief, sickness of soul — there's the Virgin. Day and night she's on the toll-free hot line offering gentle attention and prudent advice.

To European Christians half a millennium ago, the Virgin and a raft of familiar saints were the exalted personnel in a kind of celestial welfare system, available to all believers. And one quick way to access its benefits was through devotional painting of the kind found in "Prayers and Portraits: Unfolding the Netherlandish Diptych" at the National Gallery of Art.

Probably nothing in Western art comes closer to formal perfection than these pictures, produced by the likes of Jan van Eyck, Rogier van der Weyden and Hugo van der Goes across an area that now encompasses the Netherlands, Belgium, Luxembourg and parts of France. These painters were pictorial magicians, creating visual worlds, cosmically abstract and microscopically realistic, of peerless breadth.

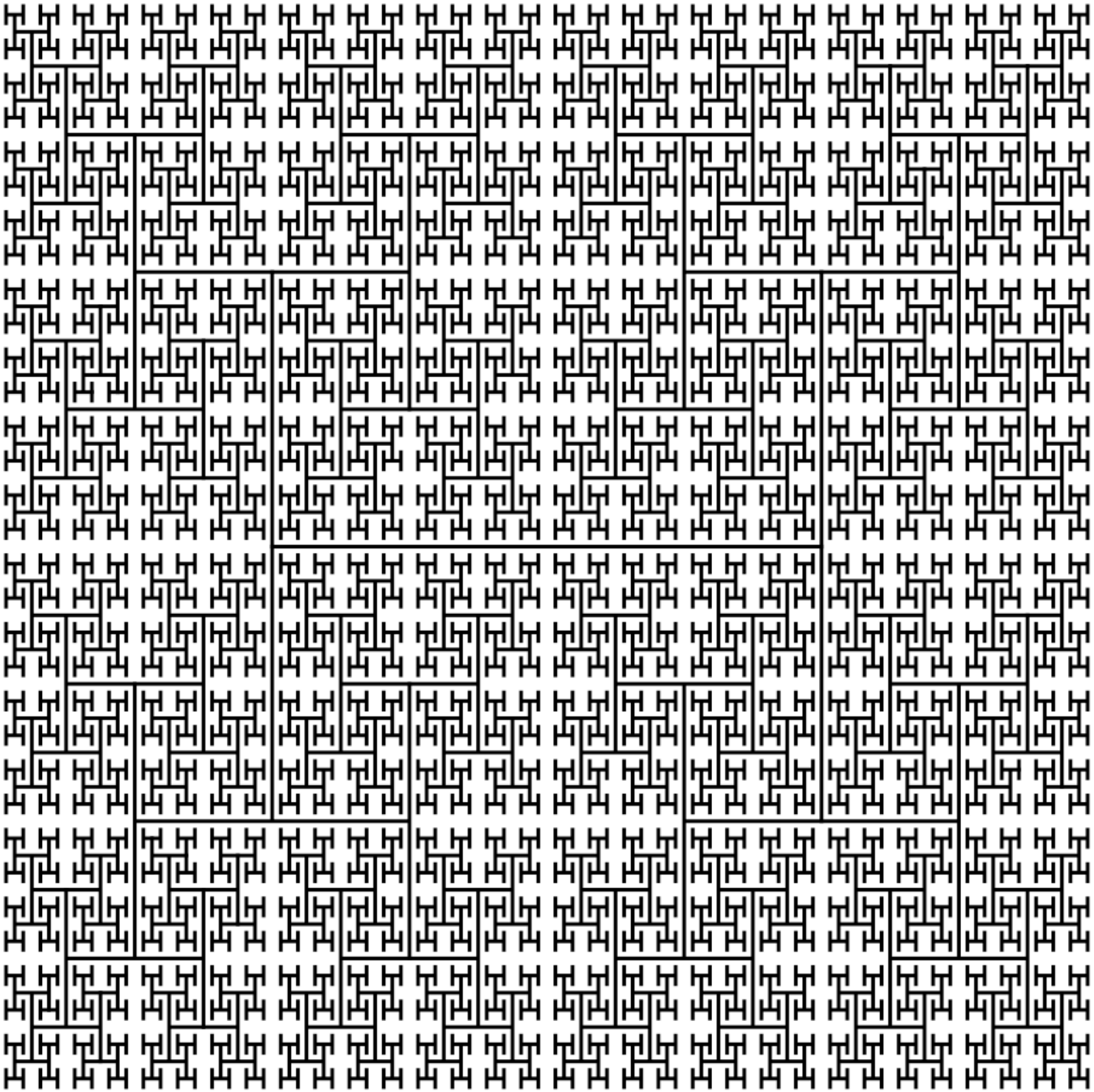
You see all of this in one glance at the 40 double-panel paintings, or diptychs, here. Then you learn gradually as you move through the show how diptych paintings have been unmade and remade, broken up and reconfigured, over the centuries, with the result that few survive in their intended form.

"Prayers and Portraits" is an attempt to restore that form, at least to a few of them. It brings art historians and art

Continued on Page 44

Left, Gemäldegalerie, Berlin; National Gallery of Art, Washington

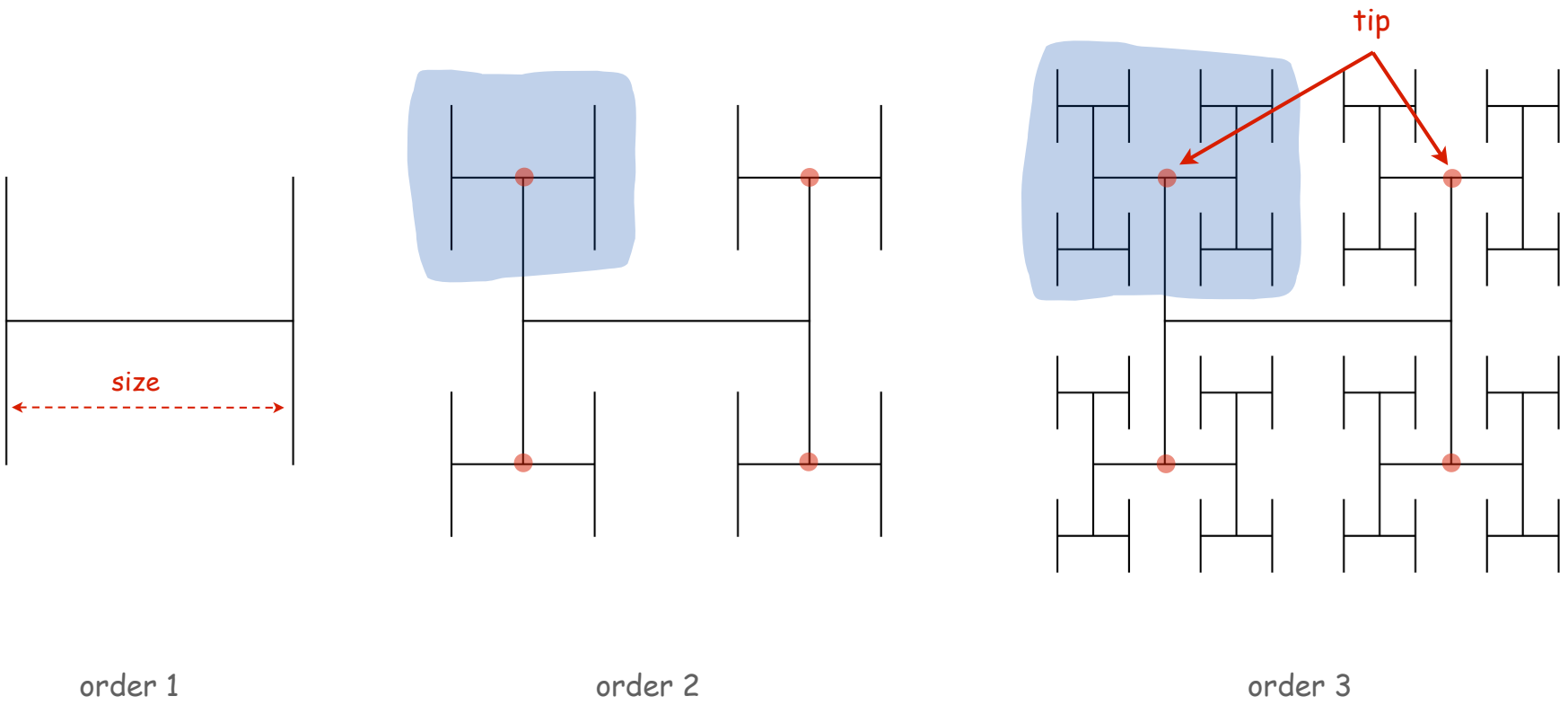
Prayers and Portraits: *Unfolding the Netherlandish Diptych.* Two panels of an early 16th-century diptych by Michel Sittow, left, are reunited in an exhibition at the National Gallery of Art in Washington through Feb. 4.



Htree

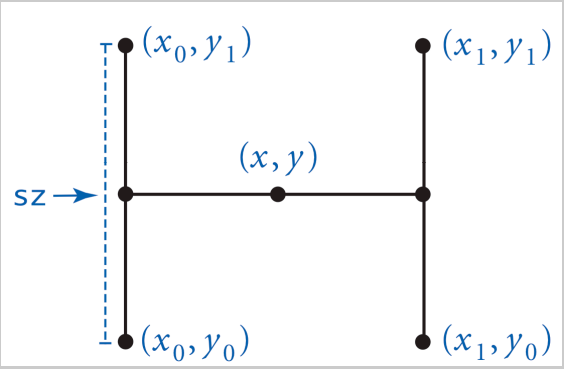
H-tree of order n.

- Draw an H.
- Recursively draw 4 H-trees of order n-1, and half the size, one connected to each tip.



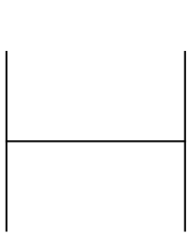
Htree in Java

```
public class Htree
{
   public static void draw(int n, double sz, double x, double y)
   {
      if (n == 0) return;
      double x0 = x - sz/2, x1 = x + sz/2;
      double y0 = y - sz/2, y1 = y + sz/2;

      StdDraw.line(x0,  y, x1,  y);
      StdDraw.line(x0, y0, x0, y1);   ← draw the H, centered on (x, y)
      StdDraw.line(x1, y0, x1, y1);

      draw(n-1, sz/2, x0, y0);
      draw(n-1, sz/2, x0, y1);
      draw(n-1, sz/2, x1, y0);        ← recursively draw 4 half-size Hs
      draw(n-1, sz/2, x1, y1);
   }

   public static void main(String[] args)
   {
      int n = Integer.parseInt(args[0]);
      draw(n, .5, .5, .5);
   }
}
```

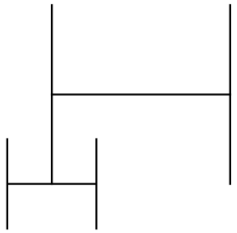


Animated H-tree

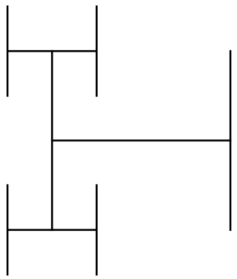
Animated H-tree. Pause for 1 second after drawing each H.



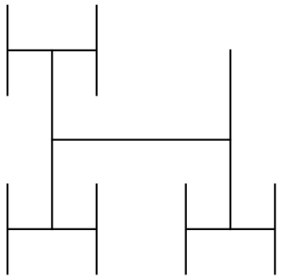
20%



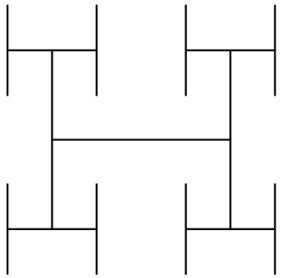
40%



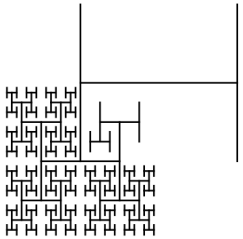
60%

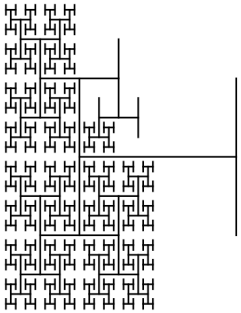


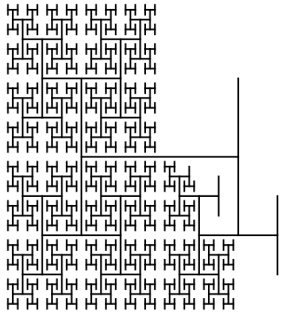
80%

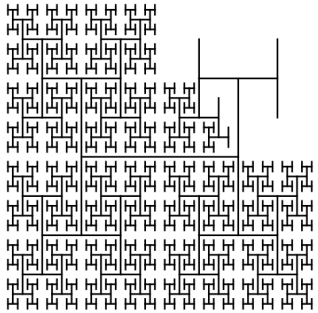


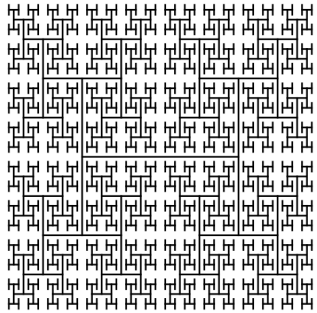
100%











Divide-and-Conquer

Divide-and-conquer paradigm.

- Break up problem into smaller subproblems of same structure.
- Solve subproblems recursively using same method.
- Combine results to produce solution to original problem.

> Divide et impera. Veni, vidi, vici. - Julius Caesar

Many important problems succumb to divide-and-conquer.

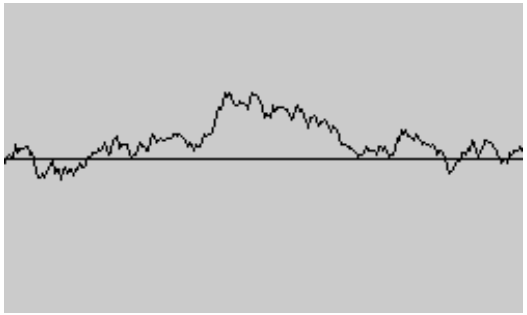
- FFT for signal processing.
- Parsers for programming languages.
- Multigrid methods for solving PDEs.
- Quicksort and mergesort for sorting.
- Hilbert curve for domain decomposition.
- Quad-tree for efficient N-body simulation.
- Midpoint displacement method for fractional Brownian motion.

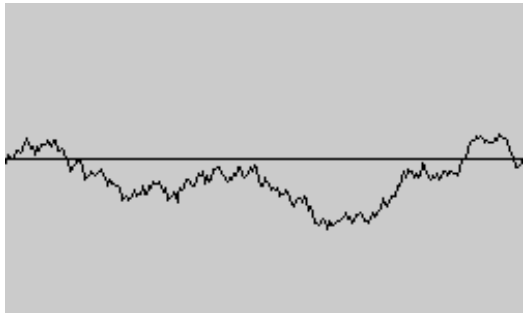
Application: Fractional Brownian Motion

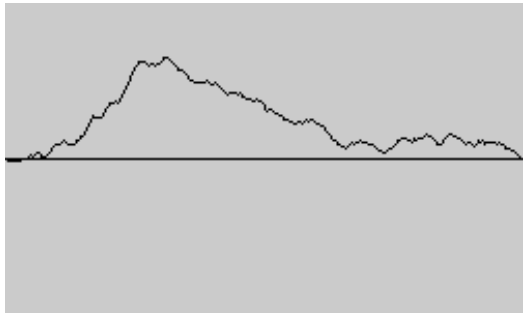
Fractional Brownian Motion

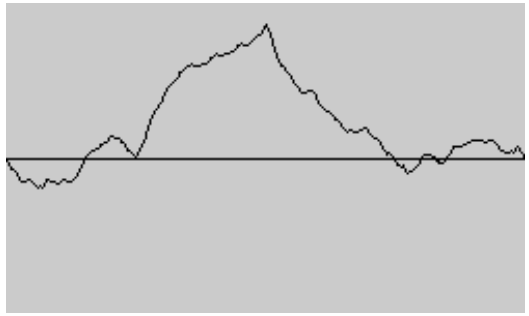
Physical process which models many natural and artificial phenomenon.

- Price of stocks.
- Dispersion of ink flowing in water.
- Rugged shapes of mountains and clouds.
- Fractal landscapes and textures for computer graphics.





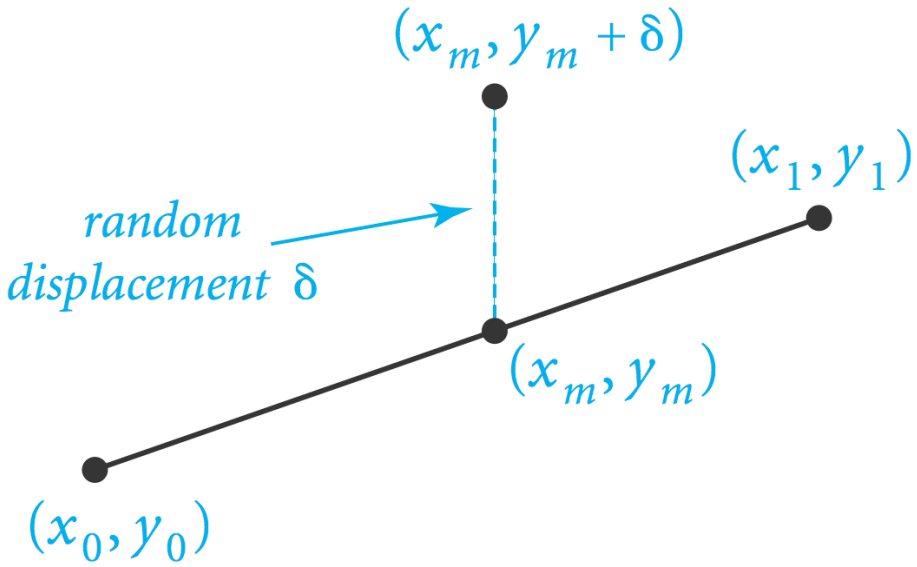




Simulating Brownian Motion

Midpoint displacement method.

- Maintain an interval with endpoints (x_0, y_0) and (x_1, y_1).
- Divide the interval in half.
- Choose δ at random from Gaussian distribution.
- Set $x_m = (x_0 + x_1)/2$ and $y_m = (y_0 + y_1)/2 + \delta$.
- Recur on the left and right intervals.



Simulating Brownian Motion: Java Implementation

Midpoint displacement method.

- Maintain an interval with endpoints (x_0, y_0) and (x_1, y_1).
- Choose δ at random from Gaussian distribution.
- Divide the interval in half: Set $x_m = (x_0 + x_1)/2$ and $y_m = (y_0 + y_1)/2 + \delta$.
- Recur on the left and right intervals.

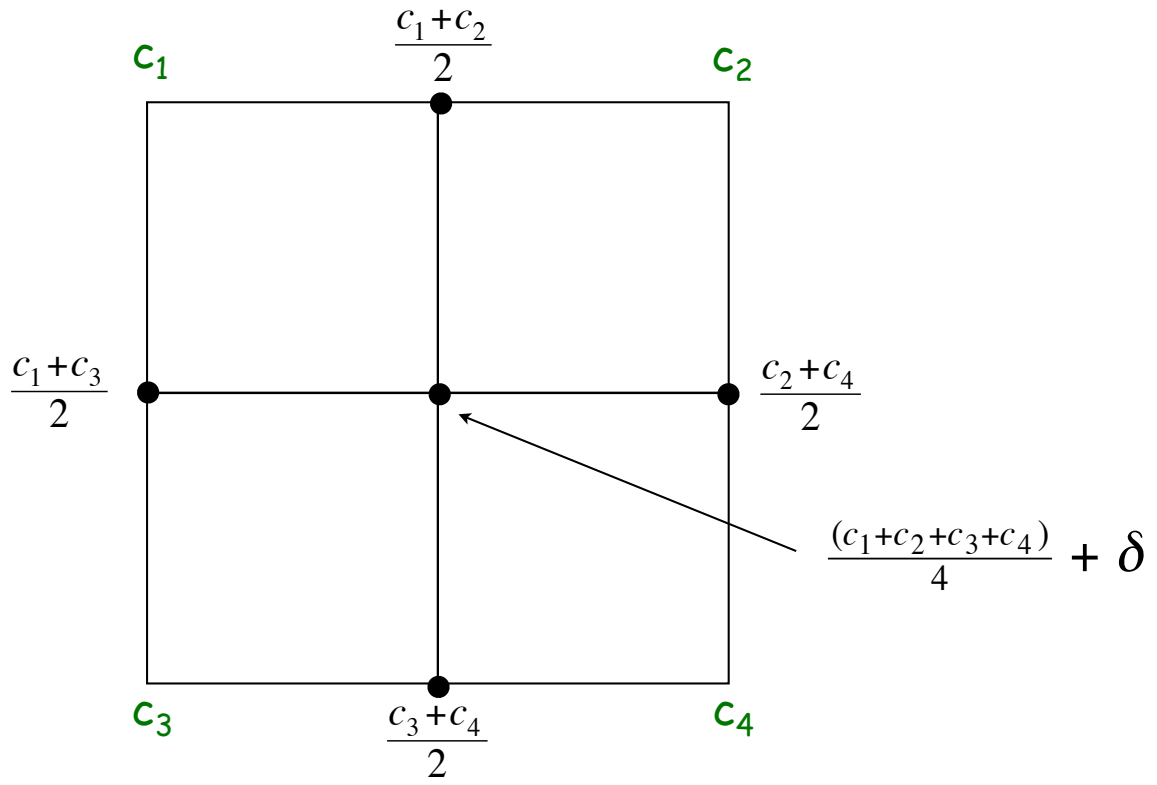
```
public static void curve(double x0, double y0,
                         double x1, double y1, double var)
{
   if (x1 - x0 < 0.01)
   {
      StdDraw.line(x0, y0, x1, y1);
      return;
   }
   double xm = (x0 + x1) / 2;
   double ym = (y0 + y1) / 2;
   ym += StdRandom.gaussian(0, Math.sqrt(var));
   curve(x0, y0, xm, ym, var/2);
   curve(xm, ym, x1, y1, var/2);
}
```

variance halves at each level;
change factor to get different shapes

Plasma Cloud

Plasma cloud centered at (x, y) of size s.

- Each corner labeled with some grayscale value.
- Divide square into four quadrants.
- The grayscale of each new corner is the average of others.
  - center: average of the four corners + random displacement
  - others: average of two original corners
- Recur on the four quadrants.

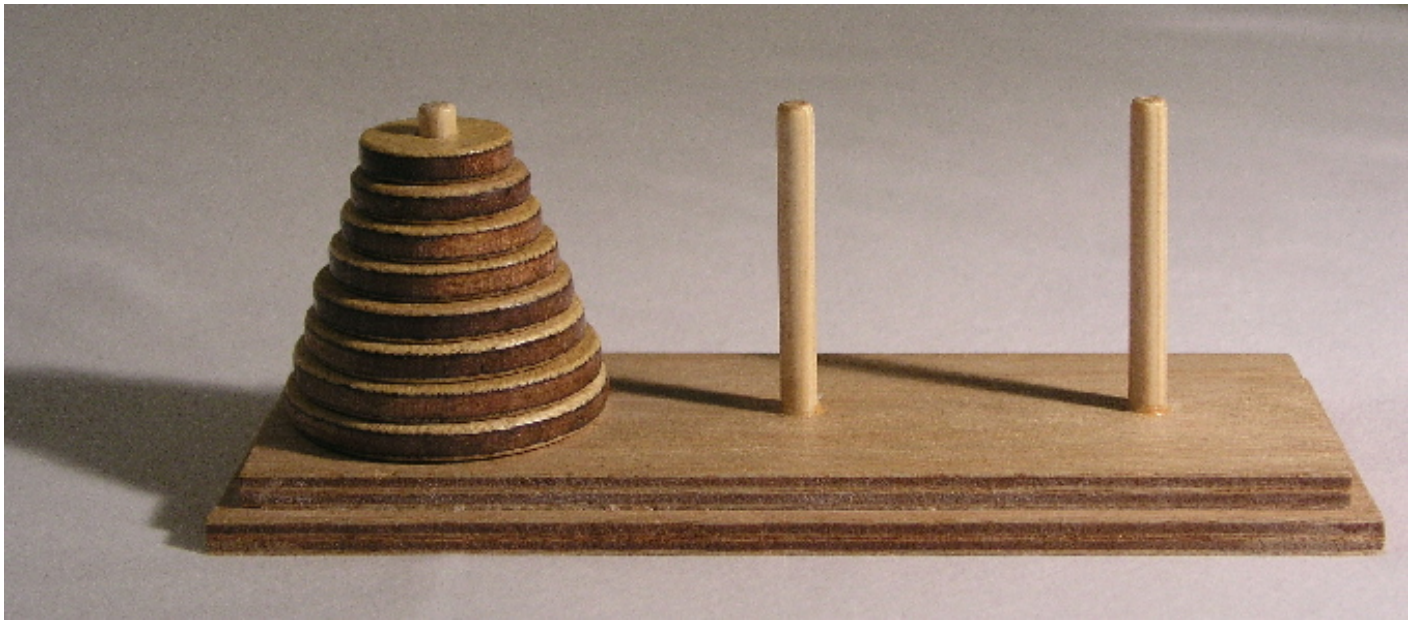


Plasma Cloud

Brownian Landscape

Reference: http://www.geocities.com/aaron_torpy/gallery.htm

Towers of Hanoi

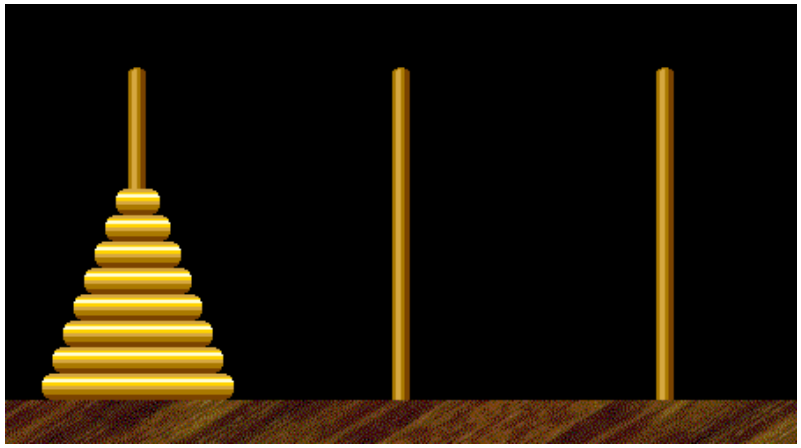


http://en.wikipedia.org/wiki/Image:Hanoiklein.jpg

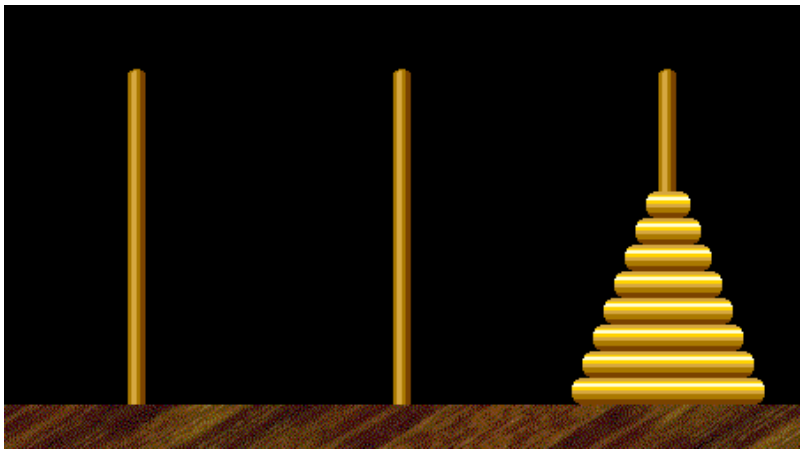
Towers of Hanoi

Move all the discs from the leftmost peg to the rightmost one.

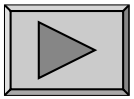
- Only one disc may be moved at a time.
- A disc can be placed either on empty peg or on top of a larger disc.



start



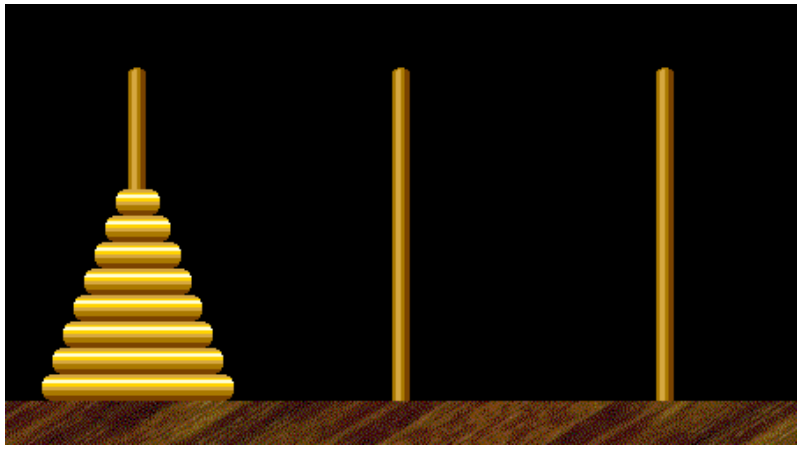
finish



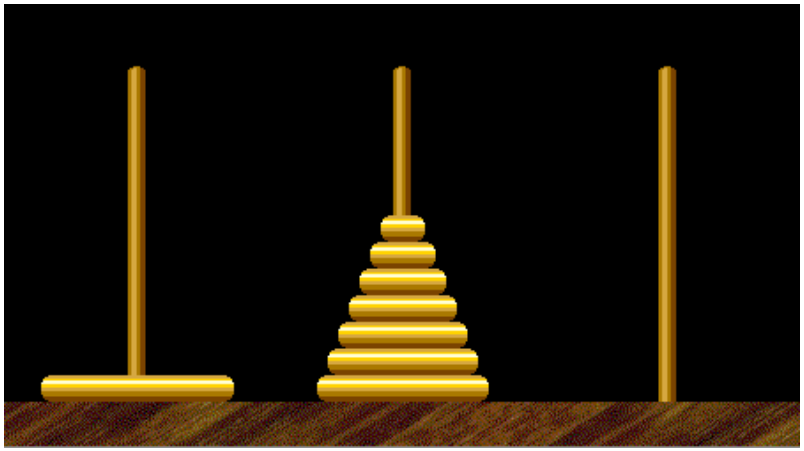
Towers of Hanoi demo

Edouard Lucas (1883)

Towers of Hanoi: Recursive Solution

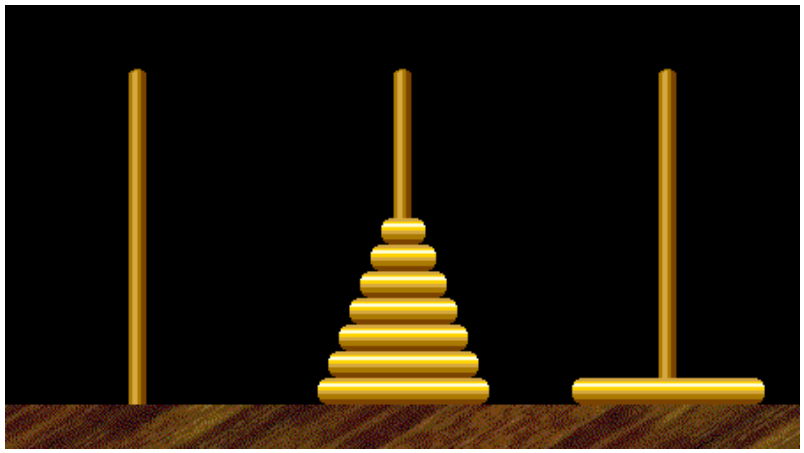


Move n-1 smallest discs right.

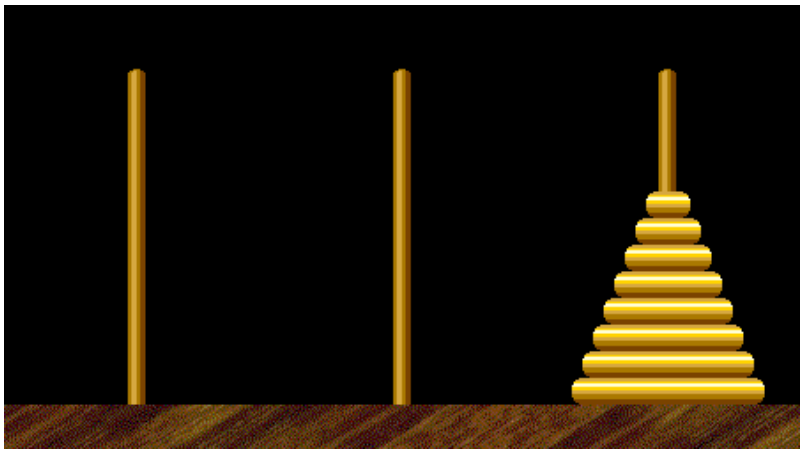


Move largest disc left.

cyclic wrap-around



Move n-1 smallest discs right.



Towers of Hanoi Legend

Q. Is world going to end (according to legend)?

- 64 golden discs on 3 diamond pegs.
- World ends when certain group of monks accomplish task.

Q. Will computer algorithms help?

Towers of Hanoi: Recursive Solution

```
public class TowersOfHanoi
{
   public static void moves(int n, boolean left)
   {
      if (n == 0) return;
      moves(n-1, !left);
      if (left) System.out.println(n + " left");
      else      System.out.println(n + " right");
      moves(n-1, !left);
   }

   public static void main(String[] args)
   {
      int N = Integer.parseInt(args[0]);
      moves(N, true);
   }

}
```

`moves(n, true)` : move discs 1 to n one pole to the left

`moves(n, false)`: move discs 1 to n one pole to the right

smallest disc

Towers of Hanoi: Recursive Solution

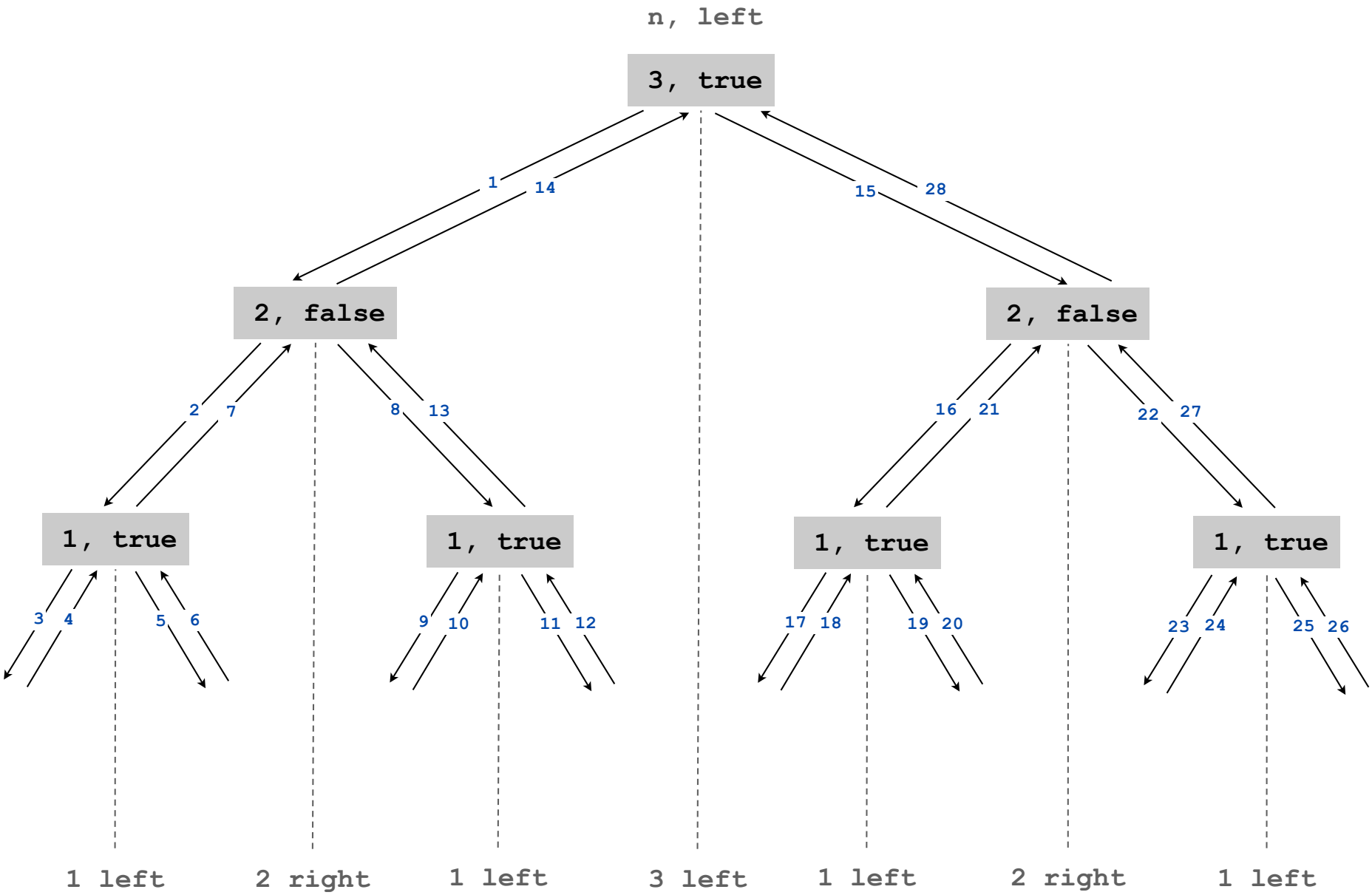
```
% java TowersOfHanoi 3
1 left
2 right
1 left
3 left
1 left
2 right
1 left
```

```
% java TowersOfHanoi 4
1 right
2 left
1 right
3 right
1 right
2 left
1 right
4 left
1 right
2 left
1 right
3 right
1 right
2 left
1 right
```

every other move is smallest disc

subdivisions of ruler

Towers of Hanoi: Recursion Tree



Towers of Hanoi: Properties of Solution

Remarkable properties of recursive solution.

- Takes $2^n - 1$ moves to solve n disc problem.
- Sequence of discs is same as subdivisions of ruler.
- Every other move involves smallest disc.

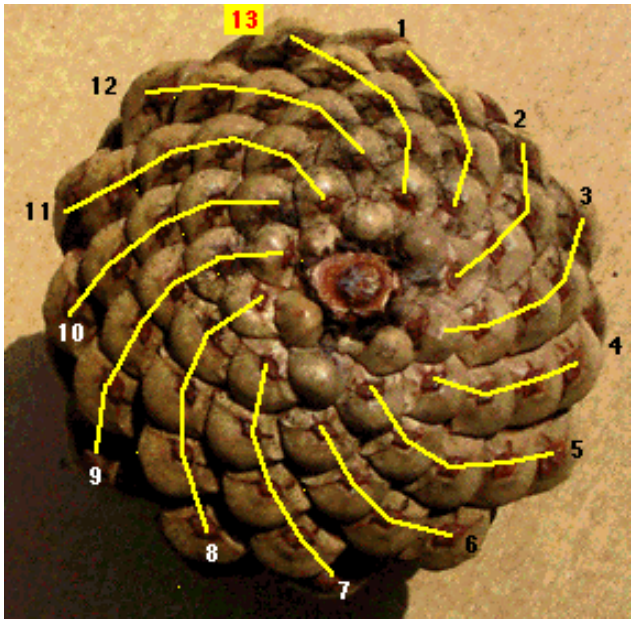
Recursive algorithm yields non-recursive solution!

- Alternate between two moves:
  - move smallest disc to right if n is even
  - make only legal move not involving smallest disc

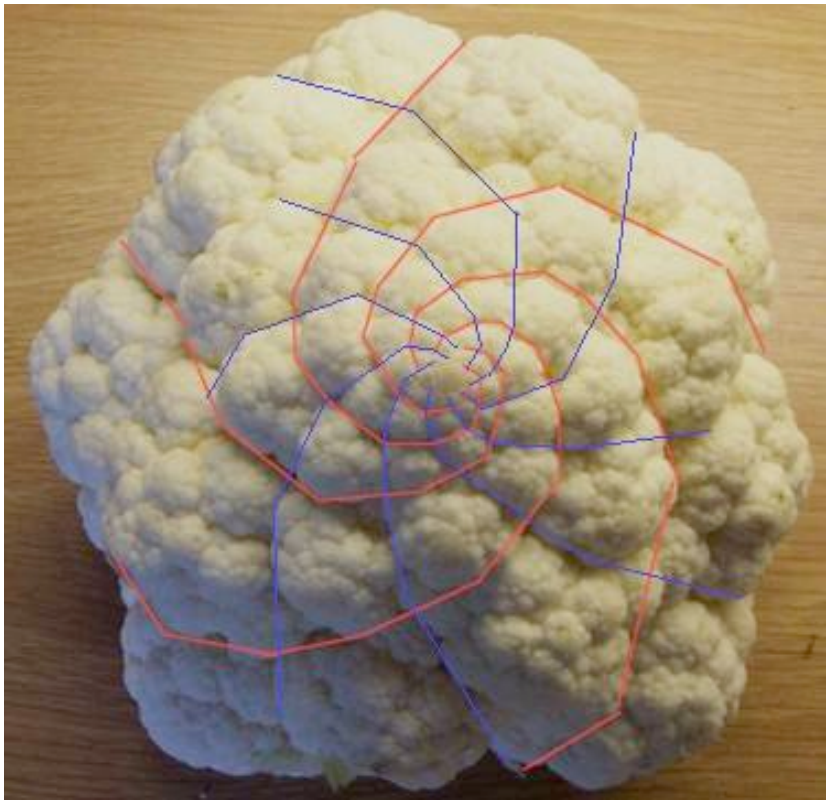
Recursive algorithm may reveal fate of world.

- Takes 585 billion years for n = 64 (at rate of 1 disc per second).
- Reassuring fact: any solution takes at least this long!

Fibonacci Numbers



pinecone

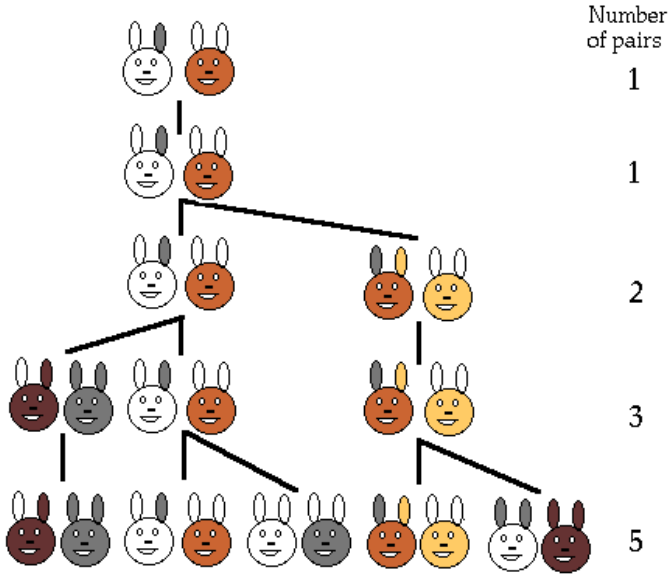


cauliflower

Fibonacci Numbers

Fibonacci numbers. 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, …

$$F_n = \begin{cases} 0 & \text{if } n = 0 \\ 1 & \text{if } n = 1 \\ F_{n-1} + F_{n-2} & \text{otherwise} \end{cases}$$



Fibonacci rabbits

L. P. Fibonacci
(1170 - 1250)

A Possible Pitfall With Recursion

Fibonacci numbers. 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, ...

$$F_n = \begin{cases} 0 & \text{if } n = 0 \\ 1 & \text{if } n = 1 \\ F_{n-1} + F_{n-2} & \text{otherwise} \end{cases}$$

FYI (classical math):

$$F(n) = \frac{\phi^n - (1-\phi)^n}{\sqrt{5}} = \left\lfloor \phi^n / \sqrt{5} \right\rfloor$$

ϕ = golden ratio $\approx$ 1.618

Ex: F(50) ≈ 1.2 × 10^{10}

A natural for recursion?

```
public static long F(int n)
{
   if (n == 0) return 0;
   if (n == 1) return 1;
   return F(n-1) + F(n-2);
}
```

TEQ on Recursion 1.1 (difficult but important)

Is this an efficient way to compute F(50)?

```
public static long F(int n)
{
   if (n == 0) return 0;
   if (n == 1) return 1;
   return F(n-1) + F(n-2);
}
```

TEQ on Recursion 1.2 (easy and also important)

Is this an efficient way to compute F(50)?

```
long[] F = new long[51];
F[0] = 0; F[1] = 1;
if (n == 1) return 1;
for (int i = 2; i <= 50; i++)
   F[i] = F[i-1] + F[i-2];
```

Summary

How to write simple recursive programs?

- Base case, reduction step.
- Trace the execution of a recursive program.
- Use pictures.

Why learn recursion?

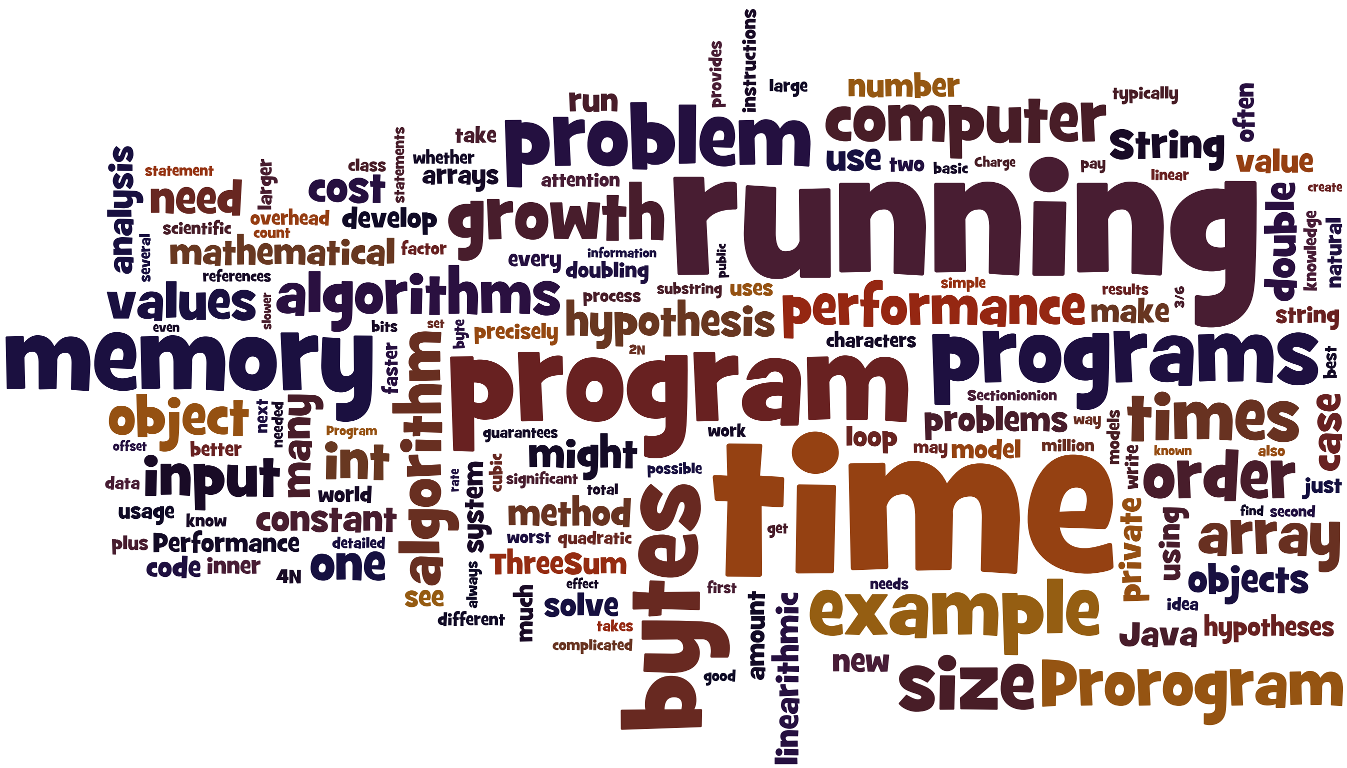
- New mode of thinking.
- Powerful programming tool.

Towers of Hanoi by W. A. Schloss.

Divide-and-conquer. Elegant solution to many important problems.

Exponential time.

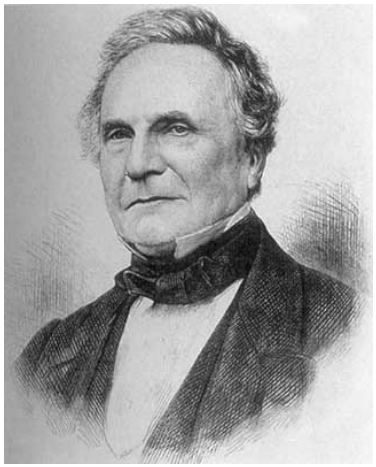
- Easy to specify recursive program that takes exponential time.
- Don't do it unless you plan to (and are working on a small problem).


running
time
program
programs
memory
problem
computer
growth
bytes
algorithms
algorithm
example
size
Prorogram
performance
hypothesis
times
order
array
objects
input
values
object
need
mathematical
analysis
cost
might
int
constant
one
method
ThreeSum
String
number
value
double
case
string
problems
model
loop
make
linearithmic
Java
hypotheses
new
solve
many
Performance
code
inner
develop
run
use
two
basic
arrays
take
whether
attention
doubling
every
precisely
characters
private
using
just
see
much
different
system
amount
typically
often
instructions
provides
large
statement
statements
class
larger
overhead
count
scientific
references
several
factor
information
public
substring
uses
simple
results
knowledge
natural
create
linear
pay
Charge
process
bits
faster
slower
even
set
byte
2N
Sectionionion
way
models
write
known
also
million
may
work
possible
guarantees
significant
cubic
rate
total
worst
quadratic
next
needed
better
offset
data
usage
know
world
plus
detailed
4N
always
effect
takes
complicated
first
get
good
needs
idea
find
second
best
3/6
Program

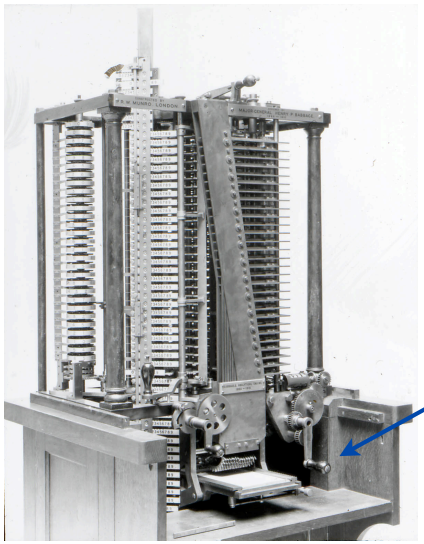
4.1 Performance

Running Time

"As soon as an Analytic Engine exists, it will necessarily guide the future course of the science. Whenever any result is sought by its aid, the question will arise - By what course of calculation can these results be arrived at by the machine in the shortest time?" – *Charles Babbage*



Charles Babbage (1864)



Analytic Engine

how many times do you have to turn the crank?

The Challenge

compile

debug on test cases

solve problems in practice

Q. Will my program be able to solve a large practical problem?

Key insight. [Knuth 1970s]

Use the scientific method to understand performance.

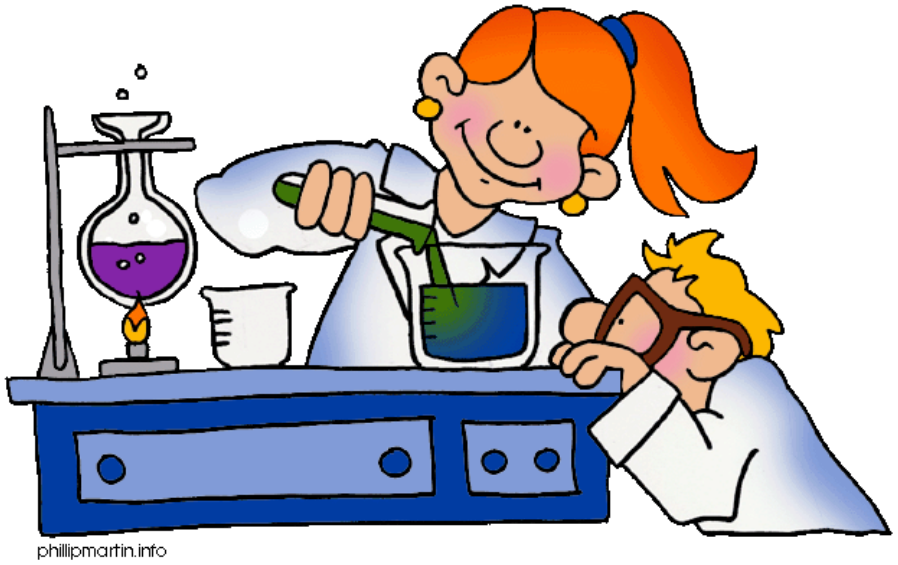
Scientific Method

Scientific method.

- Observe some feature of the natural world.
- Hypothesize a model that is consistent with the observations.
- Predict events using the hypothesis.
- Verify the predictions by making further observations.
- Validate by repeating until the hypothesis and observations agree.

Principles.

- Experiments must be reproducible;
- Hypotheses must be falsifiable.



Reasons to Analyze Algorithms

Predict performance.

- Will my program finish?
- When will my program finish?

Compare algorithms.

- Will this change make my program faster?
- How can I make my program faster?

Basis for inventing new ways to solve problems.

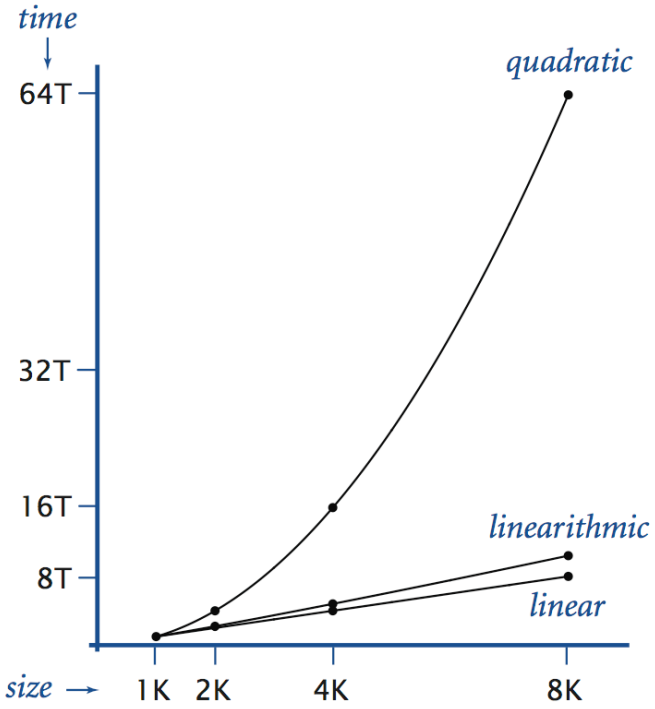
- Enables new technology.
- Enables new research.

Algorithmic Successes

Discrete Fourier transform.

- Break down waveform of N samples into periodic components.
- Applications: DVD, JPEG, MRI, astrophysics, ….
- Brute force: N^2 steps.
- FFT algorithm: N log N steps, enables new technology.

Freidrich Gauss
1805

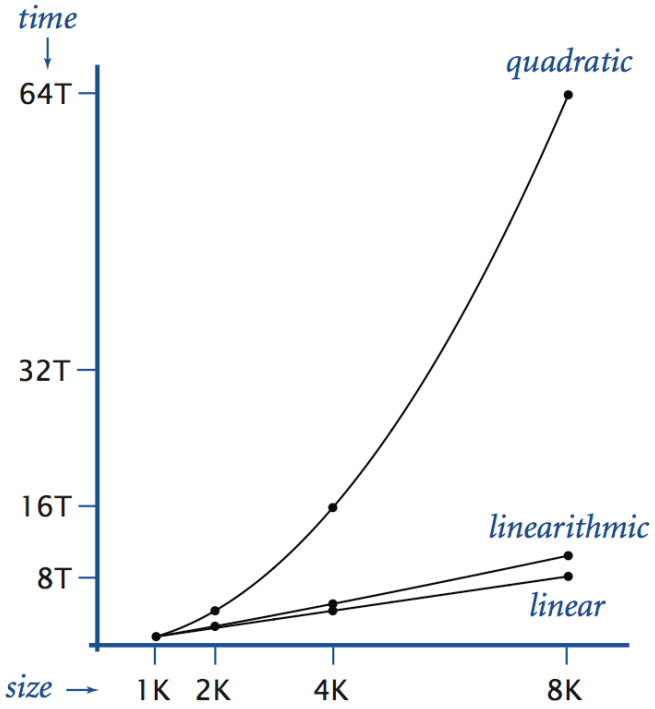


Algorithmic Successes

N-body Simulation.

- Simulate gravitational interactions among N bodies.
- Brute force: N^2 steps.
- Barnes-Hut: N log N steps, enables new research.

Andrew Appel
PU '81



Example: Three-Sum Problem

Three-sum problem. Given N integers, find triples that sum to 0.

Application. Deeply related to problems in computational geometry.

```
% more 8ints.txt
30 -30 -20 -10 40 0 10 5

% java ThreeSum < 8ints.txt
 4
 30 -30   0
 30 -20 -10
-30 -10  40
-10   0  10
```

Q. How would you write a program to solve the problem?

Three-Sum

```
public class ThreeSum
{
   // Return number of distinct triples (i, j, k)
   //      such that (a[i] + a[j] + a[k] == 0)
   public static int count(int[] a) {
      int N = a.length;
      int cnt = 0;
      for (int i = 0; i < N; i++)
         for (int j = i+1; j < N; j++)
            for (int k = j+1; k < N; k++)
               if (a[i] + a[j] + a[k] == 0) cnt++;
      return cnt;
   }

   public static void main(String[] args) {
      int[] a = StdArrayIO.readInt1D();
      StdOut.println(count(a));
   }
}
```

all possible triples i < j < k

Empirical Analysis

Empirical Analysis

Empirical analysis. Run the program for various input sizes.

N	time (1970) [1]	time (2010) [2]
500	62	0.03
1,000	531	0.26
2,000	4322	2.16
4,000	34377	17.18
8,000	265438	137.76

1. Time in seconds on Jan 18, 2010 running Linux on Sun-Fire-X4100 with 16GB RAM
2. Time in seconds in 1970 running MVT on IBM 360/50 with 256 KB RAM (estimate)

Stopwatch

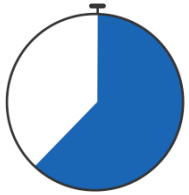
Q. How to time a program?

A. A stopwatch.

```
% java ThreeSum < 1Kints.txt
```

tick tick tick

```
0
% java ThreeSum < 2Kints.txt
```



tick tick tick tick tick tick
tick tick tick tick tick tick
tick tick tick tick tick tick
tick tick tick tick tick tick

```
2
391930676 -763182495 371251819
-326747290 802431422 -475684132
```

Stopwatch

Q. How to time a program?

A. A `Stopwatch` object.

public class Stopwatch	
Stopwatch()	*create a new stopwatch and start it running*
double elapsedTime()	*return the elapsed time since creation, in seconds*

```
public class Stopwatch
{
   private final long start;

   public Stopwatch()
   {
      start = System.currentTimeMillis();
   }

   public double elapsedTime()
   {
      return (System.currentTimeMillis() - start) / 1000.0;
   }
}
```

Stopwatch

Q. How to time a program?

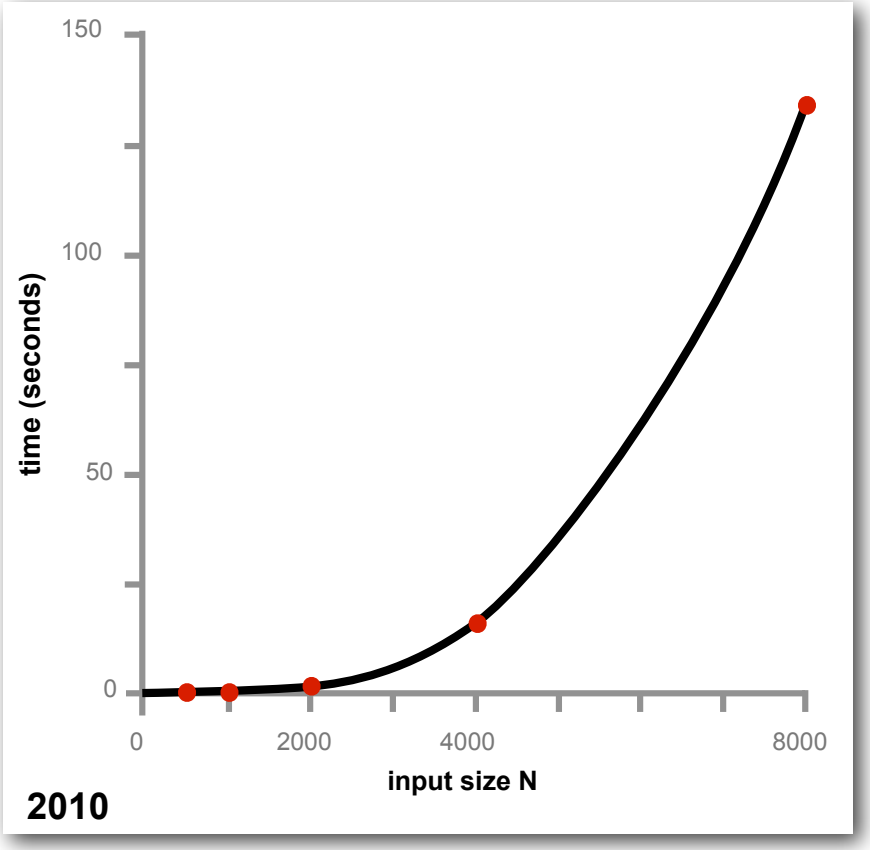
A. A `Stopwatch` object.

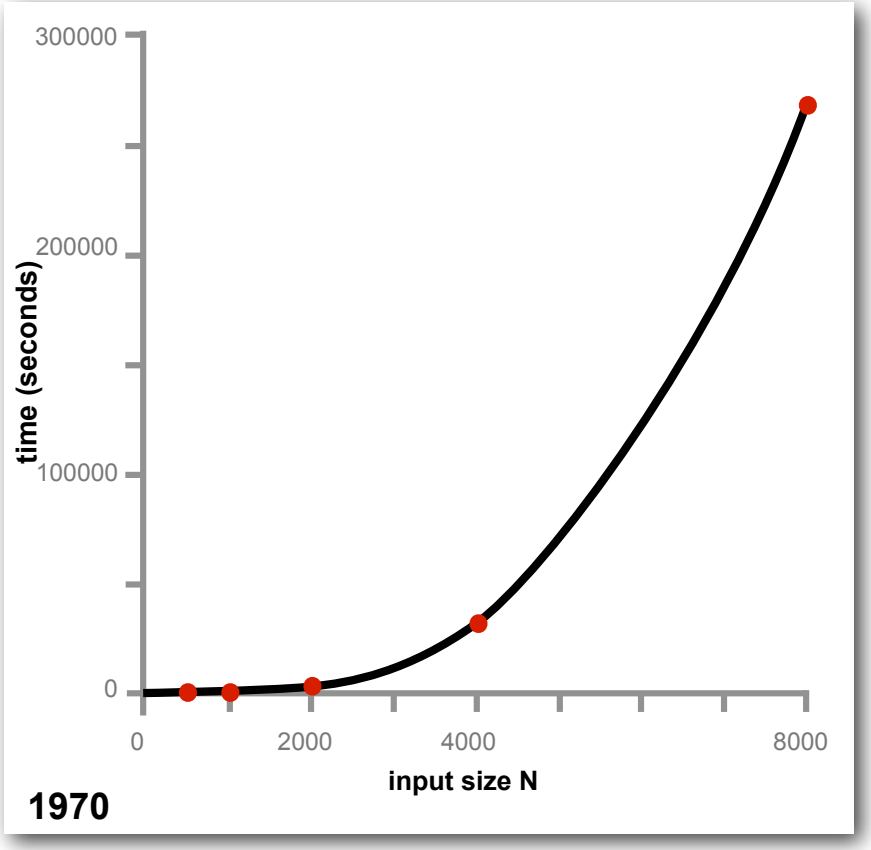
public class Stopwatch	
Stopwatch()	*create a new stopwatch and start it running*
double elapsedTime()	*return the elapsed time since creation, in seconds*

```
public static void main(String[] args)
{
   int[] a = StdArrayIO.readInt1D();
   Stopwatch timer = new Stopwatch();
   StdOut.println(count(a));
   StdOut.println(timer.elapsedTime());
}
```

Data Analysis

Data analysis. Plot running time vs. input size N.



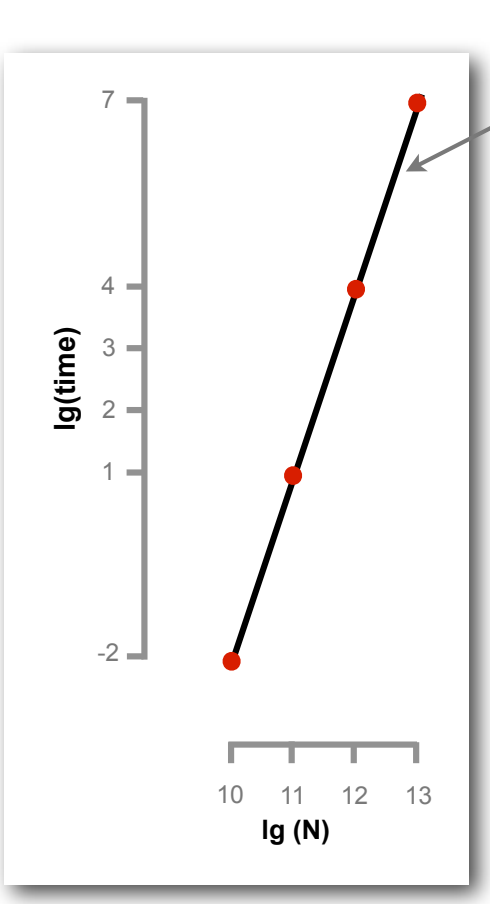


Q. How does running time grow as a function of input size N ?

Hypothesis: Running times on different computers differ by a constant factor

Data Analysis

Data analysis. Plot running time vs. input size N on a log-log scale



N	T(N)	lg(N)	lg(T(N))
1,000	0.26	10	-1.9
2,000	2.16	11	1.1
4,000	17.18	12	4.1
8,000	137.76	13	7.1

$$\lg (T(N)) = 3 \lg N + a$$

$$T(N) = a N^3$$

Hypothesis: Running time grows as the cube of the input size: $a N^3$

machine-dependent
constant factor

Prediction and verification

Hypothesis. Running time is about $a\,N^3$ for input of size N.

Q. How to estimate a ?

A. Solve for it!

$$137.76 = a \times 8000^3$$
$$\Rightarrow \quad a = 2.7 \times 10^{-10}$$

N	T(N)
1,000	0.26
2,000	2.16
4,000	17.18
8,000	137.76

Refined hypothesis. Running time is about $2.7 \times 10^{-10} \times N^3$ seconds.

Prediction. 1,100 seconds for N = 16,000.

Observation.

N	time (seconds)
16000	1110.73

validates hypothesis!

Doubling hypothesis

Doubling hypothesis. Quick two-step method for prediction.

Hypothesis: T(2N)/T(N) approaches a constant.

Step 1: Run program, doubling input size, to find the constant

Step 2: Extrapolate to predict next entries

N	T(N)	ratio
500	0.03	-
1,000	0.26	7.88
2,000	2.16	8.43
4,000	17.18	7.96
8,000	137.76	7.96
16,000	1102	8
32,000	8816	8
...	...	
512,000	36112957	

seems to converge to 8

137.76*8

1102*8

$8816*8^4$

Consistent with power law hypothesis

$a(2N)^b / aN^b = 2^b$

(exponent is lg of ratio)

Admits more functions

Ex. $T(N) = N \lg N$

$a(2N \lg 2N) / aN \lg N = 2 + 1/(\lg N) \rightarrow 2$

TEQ on Performance 1

Let F(N) be the running time of program **`Mystery`** for input N.

```
public static Mystery
{
   ...
   int N = Integer.parseInt(args[0]);
   ...
}
```

Observation:

N	T(N)	ratio
1,000	4	
2,000	15	4
4,000	60	4
8,000	240	4

Q. Predict the running time for N = 128,000

TEQ on Performance 2

Let F(N) be the running time of program **`Mystery`** for input N.

```
public static Mystery
{
   ...
   int N = Integer.parseInt(args[0]);
   ...
}
```

Observation:

N	T(N)	ratio
1,000	4	
2,000	15	4
4,000	60	4
8,000	240	4

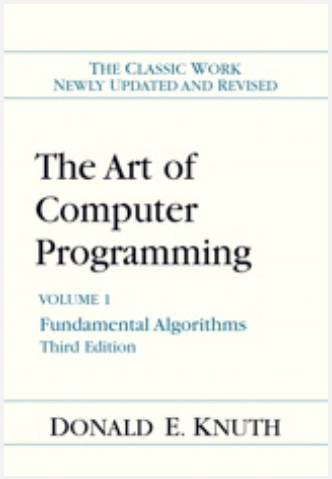
Q. Order of growth of the running time?

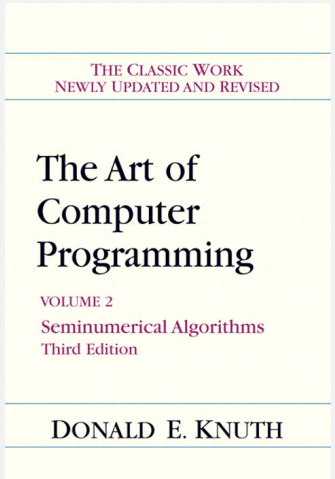
Mathematical Analysis

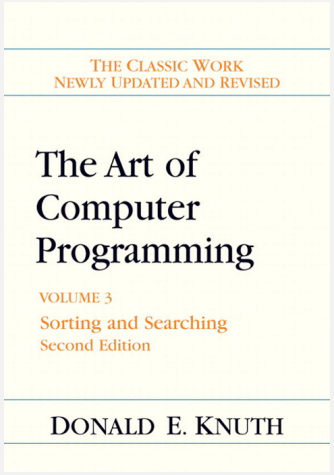
Mathematical models for running time

Total running time: sum of cost × frequency for all operations.

- Need to analyze program to determine set of operations.
- Cost depends on machine, compiler.
- Frequency depends on algorithm, input data.

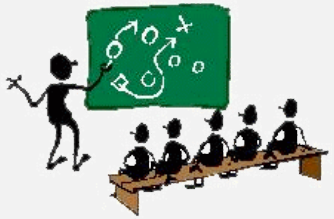






Donald Knuth
1974 Turing Award

In principle, accurate mathematical models are available.



Example: 1-sum

Q. How many instructions as a function of N?

```
int count = 0;
for (int i = 0; i < N; i++)
   if (a[i] == 0) count++;
```

operation	frequency
variable declaration	2
assignment statement	2
less than compare	$N + 1$
equal to compare	N
array access	N
increment	$\leq 2N$

between N (no zeros) and 2N (all zeros)

Example: 2-sum

Q. How many instructions as a function of N?

```
int count = 0;
for (int i = 0; i < N; i++)
   for (int j = i+1; j < N; j++)
      if (a[i] + a[j] == 0) count++;
```

operation	frequency
variable declaration	$N + 2$
assignment statement	$N + 2$
less than compare	$1/2\ (N + 1)\ (N + 2)$
equal to compare	$1/2\ N\ (N - 1)$
array access	$N\ (N - 1)$
increment	$\leq\ N^2$

$$0 + 1 + 2 + \ldots + (N-1) = \frac{1}{2} N (N-1)$$
$$= \binom{N}{2}$$

tedious to count exactly

Tilde notation

- Estimate running time (or memory) as a function of input size N.
- Ignore lower order terms.
  - when N is large, terms are negligible
  - when N is small, we don't care

Ex 1. $6N^3 + 20N + 16 \sim 6N^3$

Ex 2. $6N^3 + 100N^{4/3} + 56 \sim 6N^3$

Ex 3. $6N^3 + 17N^2 \lg N + 7N \sim 6N^3$

discard lower-order terms
(e.g., N = 1000: 6 billion vs. 169 million)

Technical definition. $f(N) \sim g(N)$ means $\lim_{N \to \infty} \frac{f(N)}{g(N)} = 1$

Example: 2-sum

Q. How long will it take as a function of N?

```
int count = 0;
for (int i = 0; i < N; i++)
   for (int j = i+1; j < N; j++)
      if (a[i] + a[j] == 0) count++;
```

"inner loop"

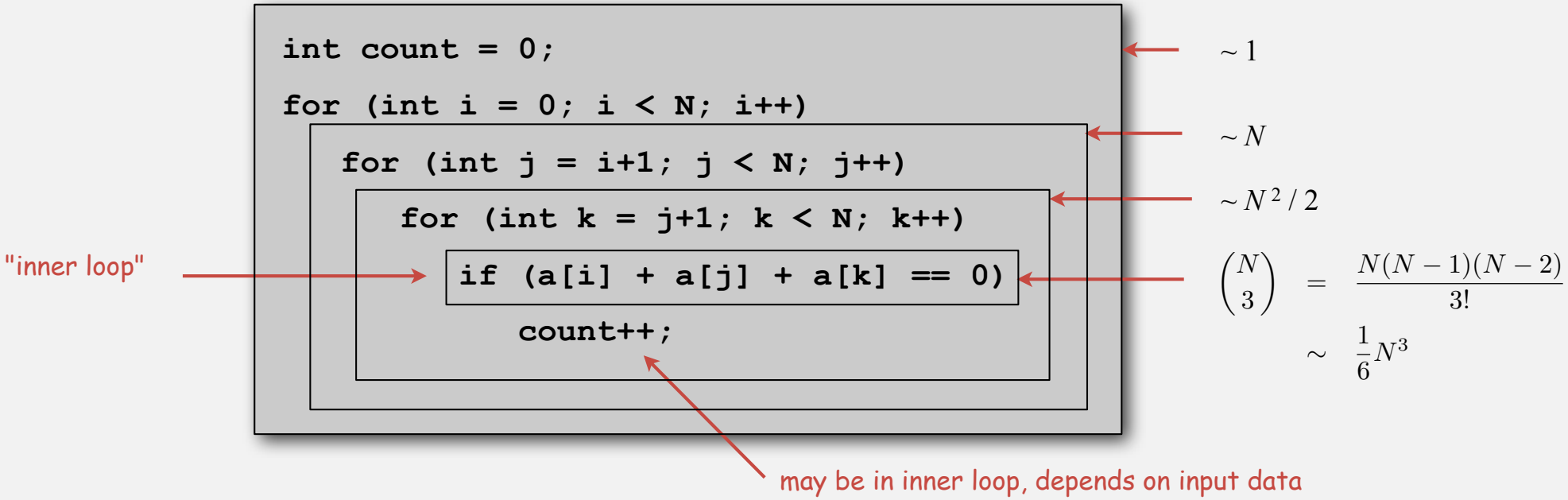
operation	frequency	time per op	total time
variable declaration	~ N	c_1	~ $c_1 N$
assignment statement	~ N	c_2	~ $c_2 N$
less than comparison	~ $1/2\ N^2$	c_3	~ $c_3 N^2$
equal to comparison	~ $1/2\ N^2$		
array access	~ N^2	c_4	~ $c_4 N^2$
increment	$\leq N^2$	c_5	$\leq c_5 N^2$
total			~ $c N^2$

depends on input data

depends on machine

Example: 3-sum

Q. How many instructions as a function of N?



Remark. Focus on instructions in inner loop; ignore everything else!

Mathematical models for running time

In principle, accurate mathematical models are available.

In practice,

- Formulas can be complicated.
- Advanced mathematics might be required.
- Exact models best left for experts.

costs (depend on machine, compiler)

$$T_N = c_1 A + c_2 B + c_3 C + c_4 D + c_5 E$$

A = variable declarations
B = assignment statements
C = compare
D = array access
E = increment

frequencies
(depend on algorithm, input)

Bottom line. We use approximate models in this course: $T_N \sim c\ N^3$.

Constants in Power Law

Power law. Running time of a typical program is $\sim a N^b$.

not quite, there may be lg(N) or similar factors

Exponent b depends on: algorithm.

Constant a depends on:

- algorithm
- input data
- hardware (CPU, memory, cache, ...)
- software (compiler, interpreter, garbage collector,...)
- system (network, other applications,...

system independent effects

system dependent effects

Our approach.

- Empirical analysis (doubling hypothesis to determine b, solve for a)
- Mathematical analysis (approximate models based on frequency counts)
- Scientific method (validate models through extrapolation)

Analysis: Empirical vs. Mathematical

Empirical analysis.

- Use doubling hypothesis to solve for a and b in power-law model $\sim a N^b$.
- Easy to perform experiments.
- Model useful for predicting, but not for explaining.

Mathematical analysis.

- Analyze algorithm to develop a model of running time as a function of N [gives a power-law or similar model where doubling hypothesis is valid].
- May require advanced mathematics.
- Model useful for predicting and explaining.

not quite, need empirical study to find a nowadays

Scientific method.

- Mathematical model is independent of a particular machine or compiler; can apply to machines not yet built.
- Empirical analysis is necessary to validate mathematical models.

Order of Growth Classifications

Observation. A small subset of mathematical functions suffice to describe running time of many fundamental algorithms.

```
while (N > 1) {
   N = N / 2;
   ...
}
```

lg N

lg N = $\log_2$ N

```
for (int i = 0; i < N; i++)
   ...
```

N

```
for (int i = 0; i < N; i++)
   for (int j = 0; j < N; j++)
      ...
```

N^2

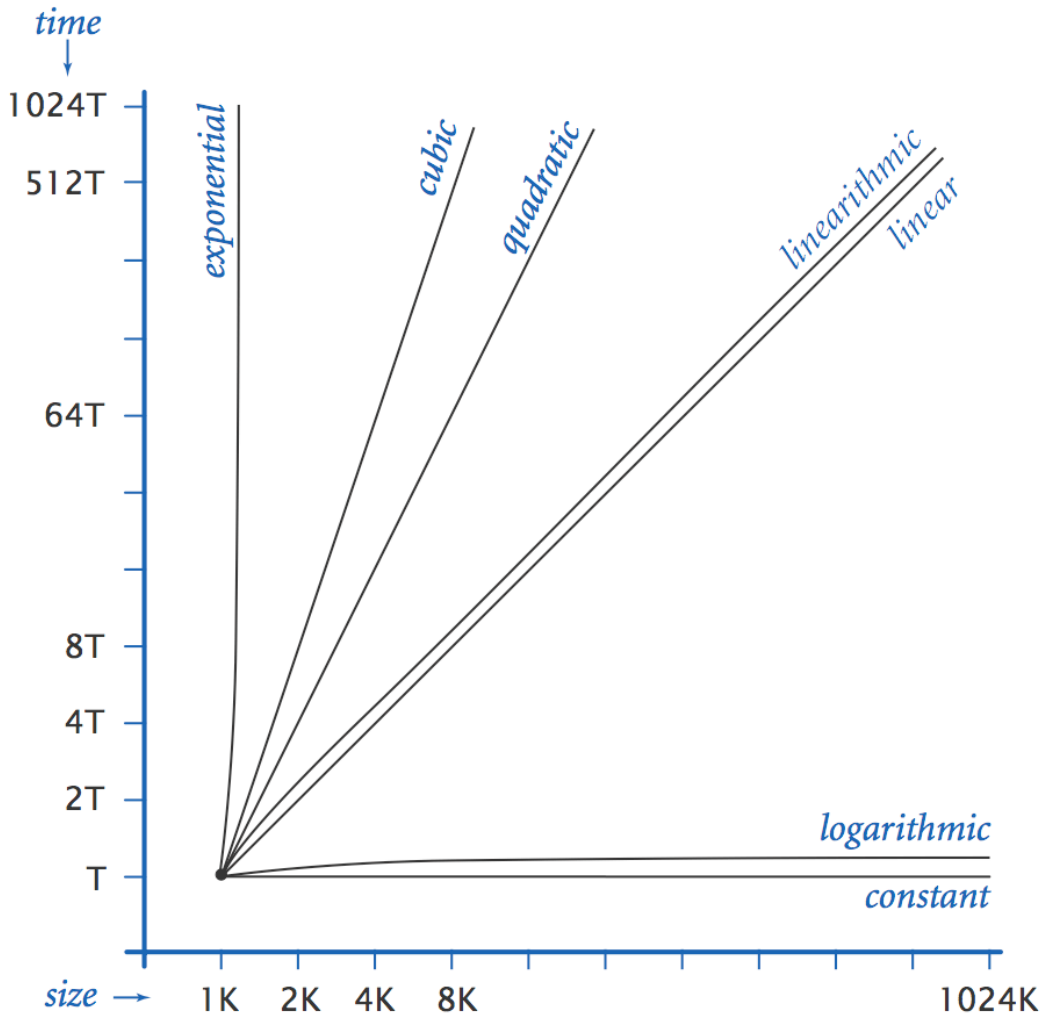
```
public static void g(int N) {
   if (N == 0) return;
   g(N/2);
   g(N/2);
   for (int i = 0; i < N; i++)
      ...
}
```

N lg N

```
public static void f(int N) {
   if (N == 0) return;
   f(N-1);
   f(N-1);
   ...
}
```

2^N

Order of Growth Classifications



Orders of growth (log-log plot)

order of growth: description	order of growth: function	factor for doubling hypothesis
constant	1	1
logarithmic	$\log N$	1
linear	N	2
linearithmic	$N \log N$	2
quadratic	N^2	4
cubic	N^3	8
exponential	2^N	2^N

Commonly encountered growth functions

Order of Growth: Consequences

order of growth	*predicted running time if problem size is increased by a factor of 100*
linear	a few minutes
linearithmic	a few minutes
quadratic	several hours
cubic	a few weeks
exponential	forever

Effect of increasing problem size for a program that runs for a few seconds

order of growth	*predicted factor of problem size increase if computer speed is increased by a factor of 10*
linear	10
linearithmic	10
quadratic	3-4
cubic	2-3
exponential	1

Effect of increasing computer speed on problem size that can be solved in a fixed amount of time

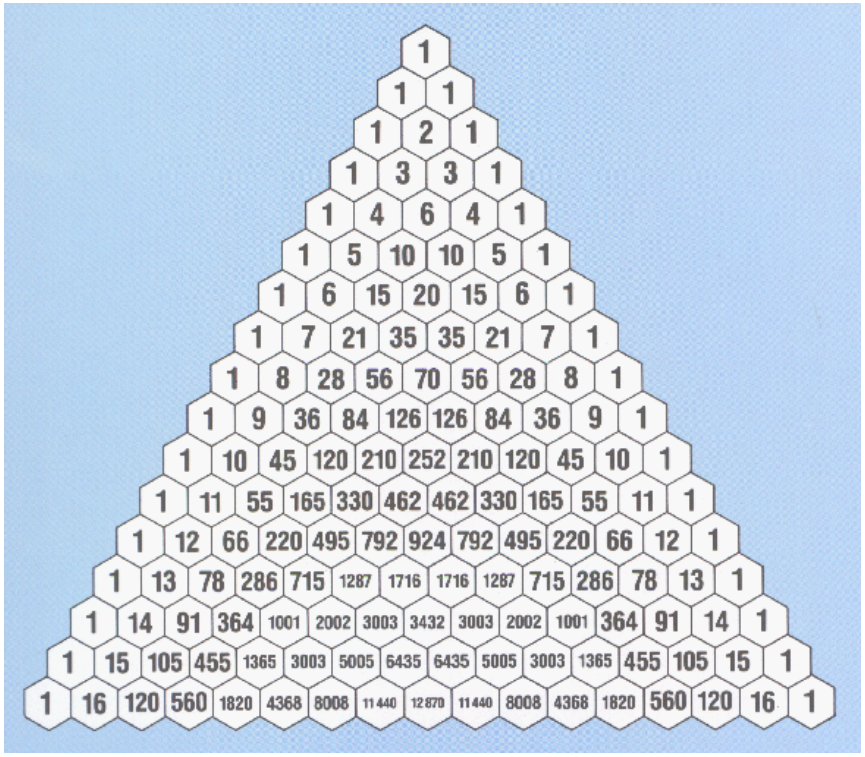
Dynamic Programming

Binomial Coefficients

Binomial coefficient. $\binom{n}{k}$ = number of ways to choose k of n elements.

Pascal's identity.

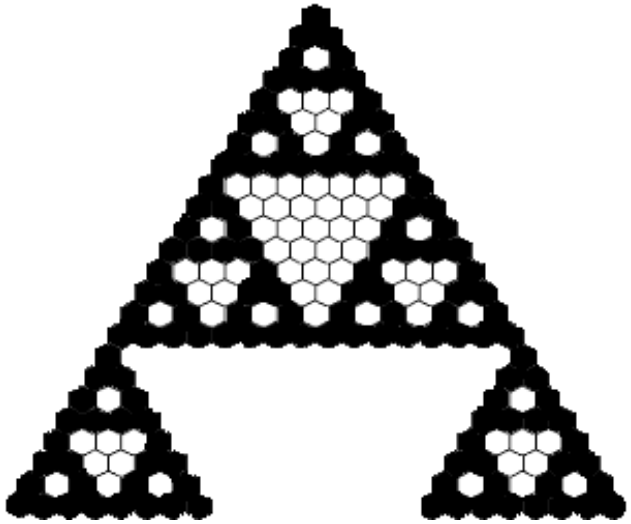
$$\binom{n}{k} = \underbrace{\binom{n-1}{k-1}}_{\text{contains first element}} + \underbrace{\binom{n-1}{k}}_{\text{excludes first element}}$$



Binomial Coefficients: Sierpinski Triangle

Binomial coefficient. $\binom{n}{k}$ = number of ways to choose k of n elements.

Sierpinski triangle. Color black the odd integers in Pascal's triangle.

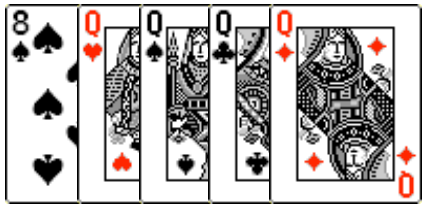


Binomial Coefficients: Poker Odds

Binomial coefficient. $\binom{n}{k}$ = number of ways to choose k of n elements.

Probability of "quads" in Texas hold 'em:

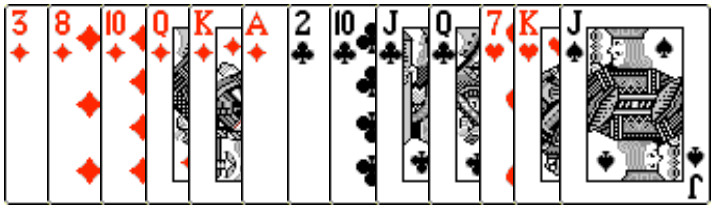
$$\frac{\binom{13}{1} \times \binom{48}{3}}{\binom{52}{7}} = \frac{224{,}848}{133{,}784{,}560} \quad (about \;\; 594 : 1)$$



Probability of 6-4-2-1 split in bridge:

$$\frac{\binom{4}{1} \times \binom{13}{6} \times \binom{3}{1} \times \binom{13}{4} \times \binom{2}{1} \times \binom{13}{2} \times \binom{1}{1} \times \binom{13}{1}}{\binom{52}{13}}$$

$$= \frac{29{,}858{,}811{,}840}{635{,}013{,}559{,}600} \quad (about \;\; 21 : 1)$$



Binomial Coefficients: First Attempt

```
public class SlowBinomial
{
   // Natural recursive implementation
   public static long binomial(long n, long k)
   {
      if (k == 0) return 1;
      if (n == 0) return 0;
      return binomial(n-1, k-1) + binomial(n-1, k);
   }

   public static void main(String[] args)
   {
      int N = Integer.parseInt(args[0]);
      int K = Integer.parseInt(args[1]);
      StdOut.println(binomial(N, K));
   }

}
```

TEQ on Performance 3

Is this an efficient way to compute binomial coefficients?

```
public static long binomial(long n, long k)
{
   if (k == 0) return 1;
   if (n == 0) return 0;
   return binomial(n-1, k-1) + binomial(n-1, k);
}
```

TEQ on Performance 4

Let F(N) be the time to compute binomial(2N, N) using the naive algorithm.

```
public static long binomial(long n, long k)
{
   if (k == 0) return 1;
   if (n == 0) return 0;
   return binomial(n-1, k-1) + binomial(n-1, k);
}
```

Observation: F(N+1)/F(N) is about 4.

What is the order of growth of the running time?

Dynamic Programming

Key idea. Save solutions to subproblems to avoid recomputation.

n \ k	0	1	2	3	4
0	1	0	0	0	0
1	1	1	0	0	0
2	1	2	1	0	0
3	1	3	3	1	0
4	1	4	6	4	1
5	1	5	10	10	5
6	1	6	15	20	15

binomial(n, k)

$$\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k}$$

$20 = 10 + 10$

Tradeoff. Trade (a little) memory for (a huge amount of) time.

Binomial Coefficients: Dynamic Programming

```
public class Binomial
{
   public static void main(String[] args) {
      int N = Integer.parseInt(args[0]);
      int K = Integer.parseInt(args[1]);
      long[][] bin = new long[N+1][K+1];

      // base cases
      for (int k = 1; k <= K; k++) bin[0][K] = 0;
      for (int n = 0; n <= N; n++) bin[N][0] = 1;

      // bottom-up dynamic programming
      for (int n = 1; n <= N; n++)
         for (int k = 1; k <= K; k++)
            bin[n][k] = bin[n-1][k-1] + bin[n-1][k];

      // print results
      StdOut.println(bin[N][K]);
   }
}
```

TEQ on Performance 5

Let F(N) be the time to compute binomial(2N, N) using dynamic programming.

```
for (int n = 1; n <= 2*N; n++)
   for (int k = 1; k <= N; k++)
      bin[n][k] = bin[n-1][k-1] + bin[n-1][k];
```

What is the order of growth of the running time?

Empirical Analysis

Timing experiments for computing binomial coefficients.

$\binom{2N}{N}$	direct recursive solution	dynamic programming
$\binom{26}{13}$	0.46	instant
$\binom{28}{14}$	1.27	instant
$\binom{30}{15}$	15.69	instant
$\binom{32}{16}$	57.40	instant
$\binom{34}{17}$	230.42	instant

increase n by 1,
running time
increases by about 4x

Stirling's Approximation

An alternative approach: $\binom{n}{k} = \frac{n!}{n!\ (n-k)!}$

Doesn't work: 52! overflows a long, even though final result doesn't.

Instead of computing exact values, use Stirling's approximation:

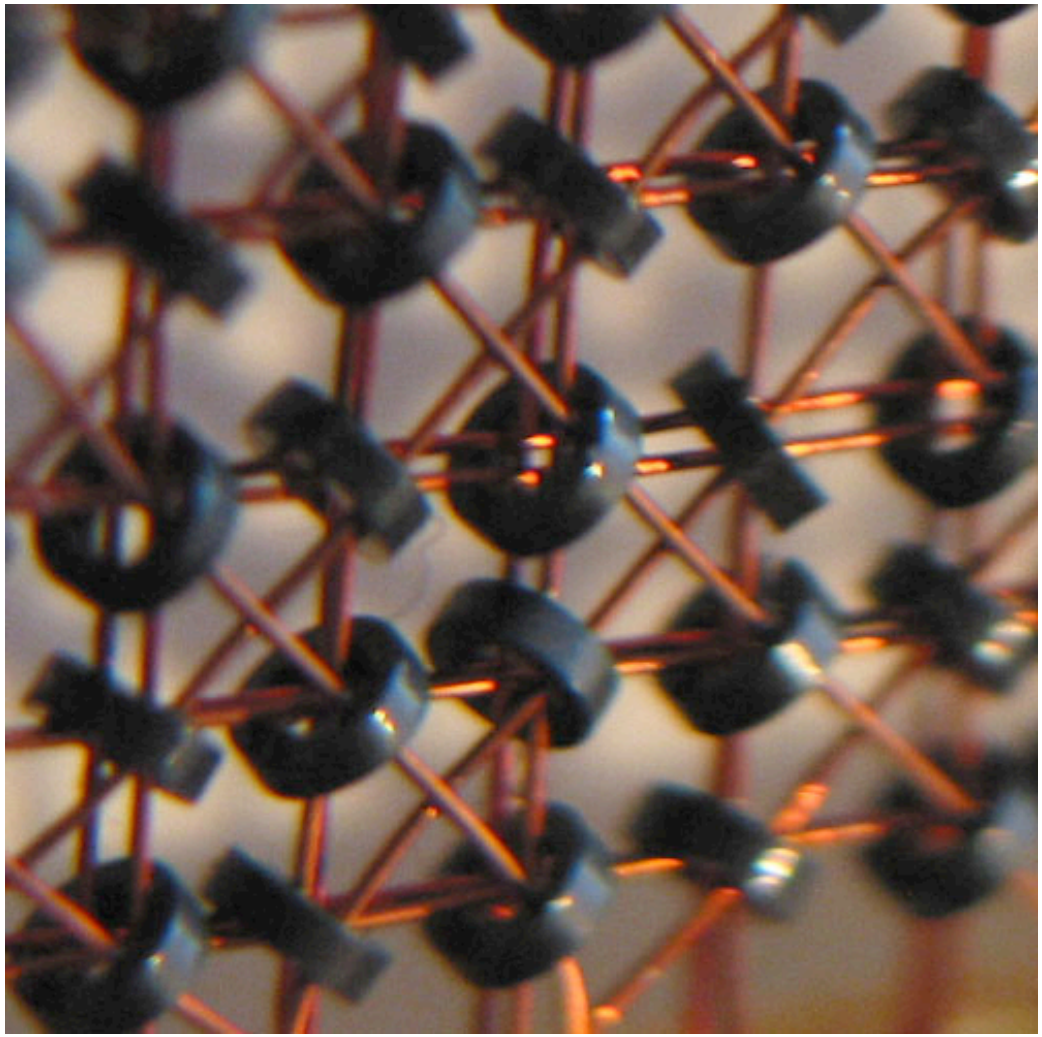
$$\ln n! \approx n \ln n - n + \frac{\ln(2\pi n)}{2} + \frac{1}{12n} - \frac{1}{360n^3} + \frac{1}{1260n^5}$$

Application. Probability of exact k heads in n flips with a biased coin.

$$\binom{n}{k} p^k (1-p)^{n-k}$$

Easy to compute approximate value with Stirling's formula

Memory



Typical Memory Requirements for Java Data Types

Bit. 0 or 1.

Byte. 8 bits.

Megabyte (MB). 2^{10} bytes ~ 1 million bytes.

Gigabyte (GB). 2^{20} bytes ~ 1 billion bytes.

type	*bytes*
`boolean`	1
`byte`	1
`char`	2
`int`	4
`float`	4
`long`	8
`double`	8

type	*bytes*
`int[]`	$4N + 16$
`double[]`	$8N + 16$
`Charge[]`	$36N + 16$
`int[][]`	$4N^2 + 20N + 16$
`double[][]`	$8N^2 + 20N + 16$
`String`	$2N + 40$

typical computer '10 has about 2GB memory

Q. What's the biggest `double` array you can store on your computer?

TEQ on Performance 6

How much memory does this program use (as a function of N)?

```
public class RandomWalk
{
   public static void main(String[] args)
   {
      int N = Integer.parseInt(args[0]);
      int[][] count = new int[N][N];
      int x = N/2;
      int y = N/2;

      for (int i = 0; i < N; i++)  {
         // no new variable declared in loop
         ...
         count[x][y]++;
      }
   }
}
```

Summary

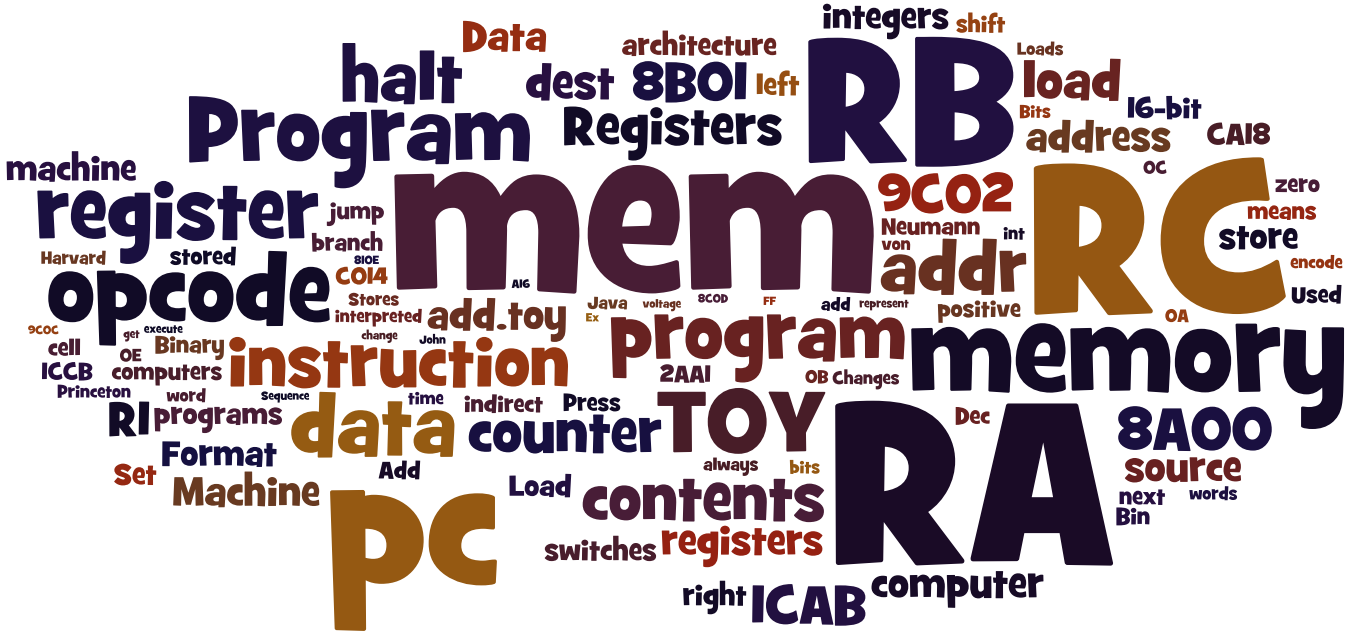
Q. How can I evaluate the performance of my program?

A. Computational experiments, mathematical analysis, scientific method

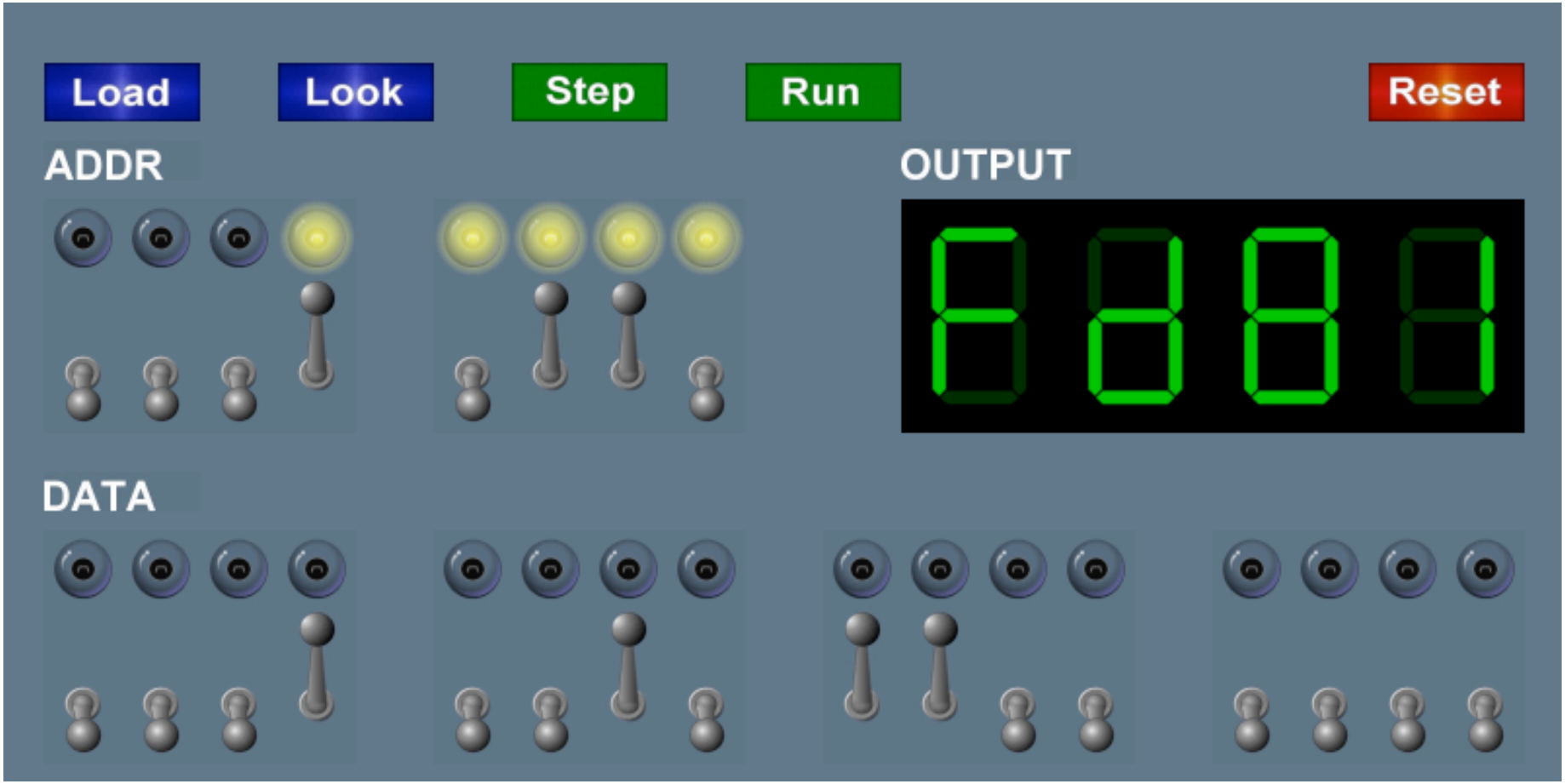
Q. What if it's not fast enough? Not enough memory?

- Understand why.
- Buy a faster computer.
- Learn a better algorithm (COS 226, COS 423).
- Discover a new algorithm.

attribute	better machine	better algorithm
cost	$$$ or more.	$ or less.
applicability	makes "everything" run faster	does not apply to some problems
improvement	incremental quantitative improvements expected	dramatic qualitative improvements possible


integers
shift
Data
architecture
Loads
halt
dest
8B01
left
RB
load
Program
Registers
Bits
16-bit
address
CA18
machine
OC
register
mem
9C02
RC
zero
jump
means
Neumann
von
int
store
branch
Harvard
stored
addr
encode
opcode
C014
8B0E
Stores
Java
voltage
8C0D
FF
add
represent
positive
Used
add.toy
interpreted
OA
9C0C
get
execute
program
memory
cell
OE
Binary
change
John
instruction
2AA1
OB
Changes
ICCB
computers
Princeton
word
Sequence
time
indirect
Press
Dec
RI
programs
data
counter
TOY
RA
8A00
Format
always
bits
source
Set
Add
Machine
Load
contents
next
words
Bin
pc
switches
registers
right
ICAB
computer

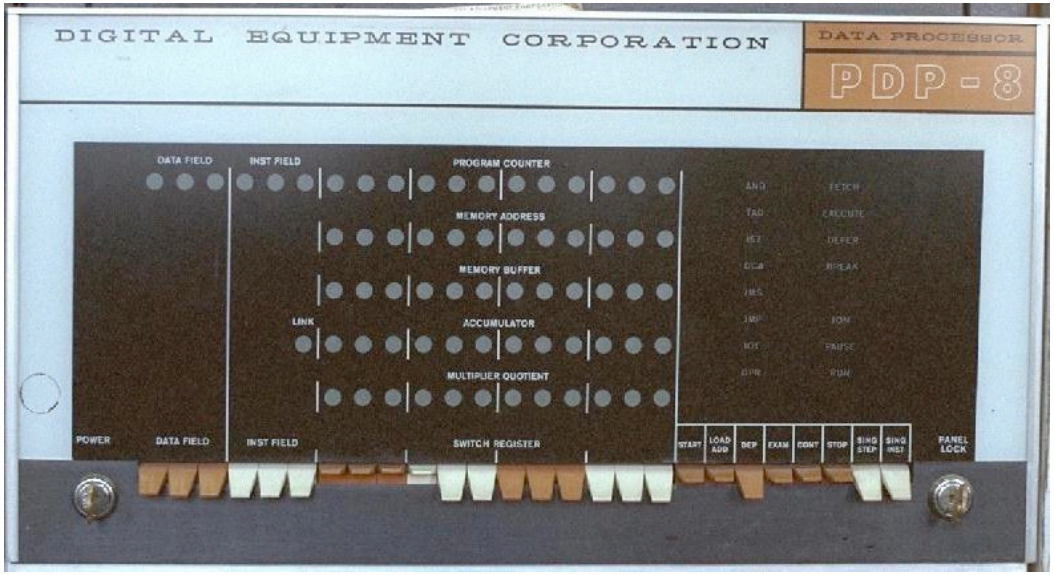
5. The TOY Machine



What is TOY?

An imaginary machine similar to:

- Ancient computers.
- Today's microprocessors.



Why Study TOY?

Machine language programming.

- How do Java programs relate to computer?
- Key to understanding Java references.
- Still situations today where it is really necessary.

multimedia, computer games, embedded devices, scientific computing, MMX, Altivec

Computer architecture.

- How does it work?
- How is a computer put together?

TOY machine. Optimized for simplicity, not cost or performance.

Inside the Box

Switches. Input data and programs.

Lights. View data.

Memory.

- Stores data and programs.
- 256 16-bit "words."
- Special word for stdin / stdout.

Program counter (PC).

- An extra 8-bit register.
- Keeps track of next instruction to be executed.

Registers.

- Fastest form of storage.
- Scratch space during computation.
- 16 16-bit registers.
- Register 0 is always 0.

Arithmetic-logic unit (ALU). Manipulate data stored in registers.

Standard input, standard output. Interact with outside world.

Data and Programs Are Encoded in Binary

Each bit consists of two states:

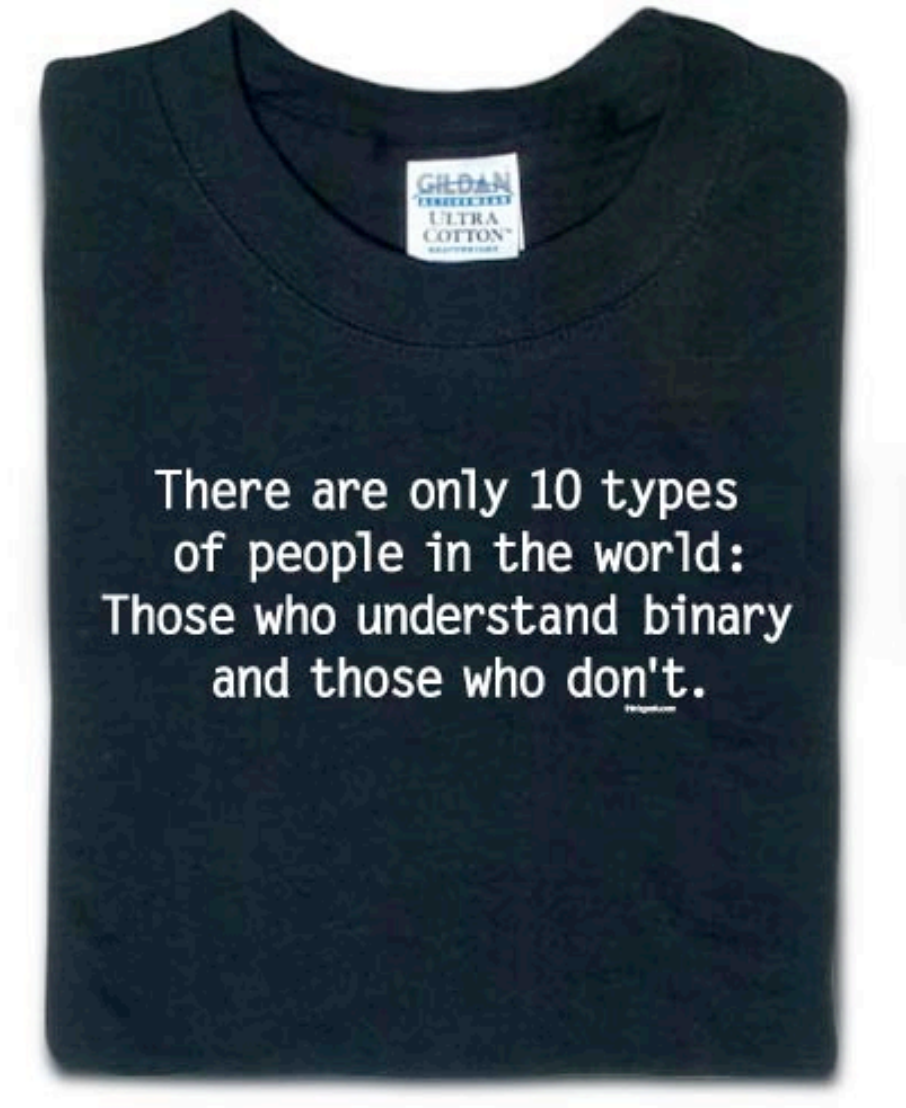
- 1 or 0; true or false.
- Switch is on or off; wire has high voltage or low voltage.

Everything stored in a computer is a sequence of bits.

- Data and programs.
- Text, documents, pictures, sounds, movies, executables, …

$\mathtt{M} = 77_{10} = 01001101_2 = 4D_{16}$
$\mathtt{O} = 79_{10} = 01001111_2 = 4F_{16}$
$\mathtt{M} = 77_{10} = 01001101_2 = 4D_{16}$

Binary People



http://www.thinkgeek.com/tshirts/frustrations/5aa9/zoom/

Binary Encoding

How to represent integers?

- Use binary encoding.
- Ex: $6375_{10} = 0001100011100111_2$

Dec	Bin
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111

Dec	Bin
8	1000
9	1001
10	1010
11	1011
12	1100
13	1101
14	1110
15	1111

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	1	0	0	0	1	1	1	0	0	1	1	1

$$6375_{10} = +2^{12} \; +2^{11} \; +2^{7} \; +2^{6} \; +2^{5} \; +2^{2} \; +2^{1} \; +2^{0}$$

$$= 4096 \; +2048 \; +128 \; +64 \; +32 \; +4 \; +2 \; +1$$

Hexadecimal Encoding

How to represent integers?

- Use hexadecimal encoding.
- Binary code, four bits at a time.
- Ex: 6375_{10} = 0001100011100111_2
  = $18E7_{16}$

Dec	Bin	Hex	Dec	Bin	Hex
0	0000	0	8	1000	8
1	0001	1	9	1001	9
2	0010	2	10	1010	A
3	0011	3	11	1011	B
4	0100	4	12	1100	C
5	0101	5	13	1101	D
6	0110	6	14	1110	E
7	0111	7	15	1111	F

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	1	0	0	0	1	1	1	0	0	1	1	1
1				8				E				7			

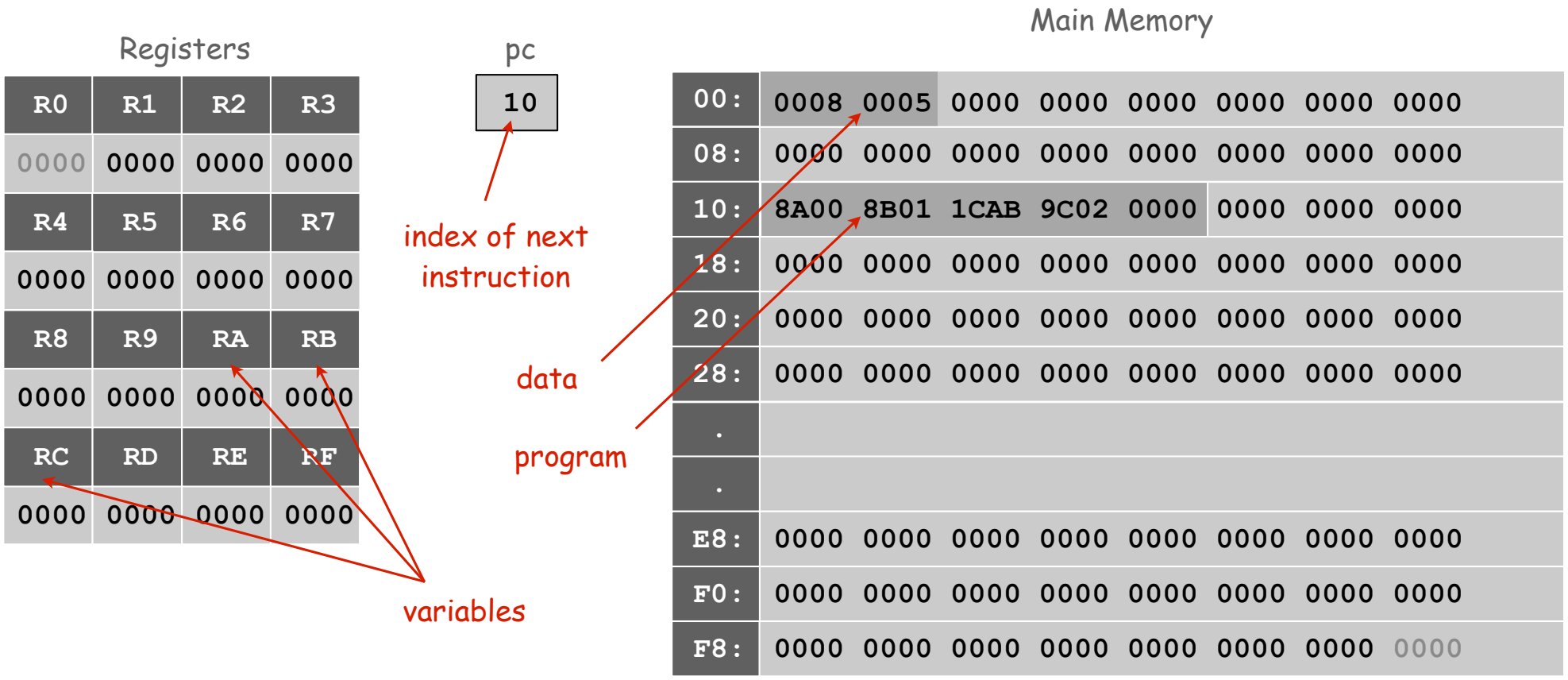
$6375_{10} = 1 \times 16^3 + 8 \times 16^2 + 14 \times 16^1 + 7 \times 16^0$

$= 4096 + 2048 + 224 + 7$

Machine "Core" Dump

Machine contents at a particular place and time.

- Record of what program has done.
- Completely determines what machine will do.



A Sample Program

A sample program. Adds 0008 + 0005 = 000D.

RA	RB	RC
0000	0000	0000

Registers

pc
10

Program counter

TOY memory (program and data)

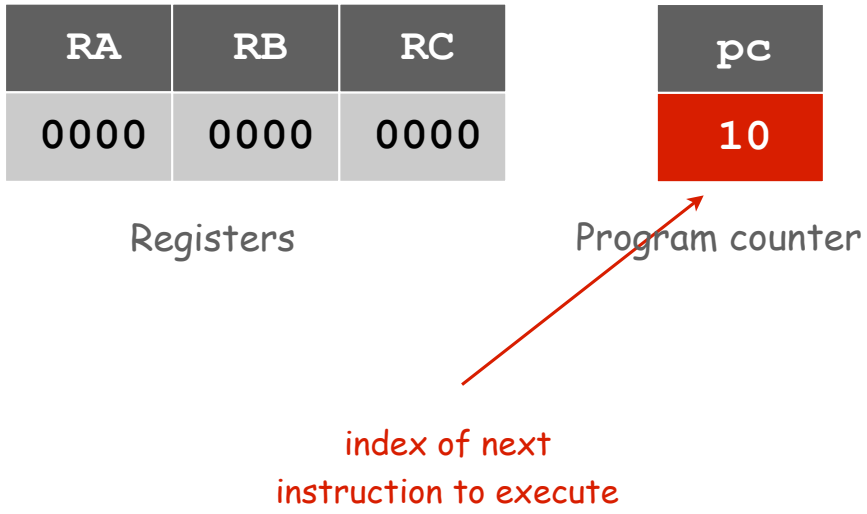
comments

```
00: 0008    8
01: 0005    5
02: 0000    0

10: 8A00    RA ← mem[00]
11: 8B01    RB ← mem[01]
12: 1CAB    RC ← RA + RB
13: 9C02    mem[02] ← RC
14: 0000    halt
```

add.toy

A Sample Program

Program counter. The `pc` is initially `10`, so the machine interprets `8A00` as an instruction.




```
00: 0008    8
01: 0005    5
02: 0000    0

10: 8A00    RA ← mem[00]
11: 8B01    RB ← mem[01]
12: 1CAB    RC ← RA + RB
13: 9C02    mem[02] ← RC
14: 0000    halt
```

add.toy

Load

Load. [opcode 8]

- Loads the contents of some memory location into a register.
- `8A00` means load the contents of memory cell `00` into register `A`.

RA	RB	RC
0000	0000	0000

Registers

pc
10

Program counter

```
00: 0008   8
01: 0005   5
02: 0000   0

10: 8A00   RA ← mem[00]
11: 8B01   RB ← mem[01]
12: 1CAB   RC ← RA + RB
13: 9C02   mem[02] ← RC
14: 0000   halt
```

add.toy

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0
8_{16}				A_{16}				00_{16}							
opcode				dest d				addr							

Load

Load. [opcode 8]

- Loads the contents of some memory location into a register.
- `8B01` means load the contents of memory cell `01` into register `B`.

RA	RB	RC
0008	0000	0000

Registers

pc
11

Program counter

```
00: 0008    8
01: 0005    5
02: 0000    0

10: 8A00    RA ← mem[00]
11: 8B01    RB ← mem[01]
12: 1CAB    RC ← RA + RB
13: 9C02    mem[02] ← RC
14: 0000    halt
```

add.toy

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	1	0	1	1	0	0	0	0	0	0	0	1
8_{16}				B_{16}				01_{16}							
opcode				dest d				addr							

Add

Add. [opcode 1]

- Add contents of two registers and store sum in a third.
- `1CAB` means add the contents of registers `A` and `B` and put the result into register `C`.

RA	RB	RC
0008	0005	0000

Registers

pc
12

Program counter

```
00: 0008   8
01: 0005   5
02: 0000   0

10: 8A00   RA ← mem[00]
11: 8B01   RB ← mem[01]
12: 1CAB   RC ← RA + RB
13: 9C02   mem[02] ← RC
14: 0000   halt
```

add.toy

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	1	1	0	0	1	0	1	0	1	0	1	1
1_{16}				C_{16}				A_{16}				B_{16}			
opcode				dest d				source s				source t			

Store

Store. [opcode 9]

- Stores the contents of some register into a memory cell.
- `9C02` means store the contents of register `C` into memory cell `02`.

RA	RB	RC
0008	0005	000D

Registers

pc
13

Program counter

```
00: 0008   8
01: 0005   5
02: 0000   0

10: 8A00   RA ← mem[00]
11: 8B01   RB ← mem[01]
12: 1CAB   RC ← RA + RB
13: 9C02   mem[02] ← RC
14: 0000   halt
```

add.toy

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	1	1	1	0	0	0	0	0	0	0	0	1	0
9_{16}				C_{16}				02_{16}							
opcode				dest d				addr							

Halt

Halt. [opcode 0]

- Stop the machine.

RA	RB	RC
0008	0005	000D

Registers

pc
14

Program counter

```
00: 0008    8
01: 0005    5
02: 000D    D

10: 8A00    RA ← mem[00]
11: 8B01    RB ← mem[01]
12: 1CAB    RC ← RA + RB
13: 9C02    mem[02] ← RC
14: 0000    halt
```

add.toy

Program and Data

Program. Sequence of 16-bit integers, interpreted one way.

Data. Sequence of 16-bit integers, interpreted other way.

Program counter (pc). Holds memory address of the "next instruction" and determines which integers get interpreted as instructions.

16 instruction types. Changes contents of registers, memory, and pc in specified, well-defined ways.

Instructions

→	0:	halt
→	1:	add
	2:	subtract
	3:	and
	4:	xor
	5:	shift left
	6:	shift right
	7:	load address
→	8:	load
→	9:	store
	A:	load indirect
	B:	store indirect
	C:	branch zero
	D:	branch positive
	E:	jump register
	F:	jump and link

TOY Instruction Set Architecture

TOY instruction set architecture (ISA).

- Interface that specifies behavior of machine.
- 16 register, 256 words of main memory, 16-bit words.
- 16 instructions.

Each instruction consists of 16 bits.

- Bits 12-15 encode one of 16 instruction types or opcodes.
- Bits 8-11 encode destination register `d`.
- Bits 0-7 encode:

  [Format 1] source registers `s` and `t`

  [Format 2] 8-bit memory address or constant

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	1	0	1	1	1	0	1	0	0	0	0	0	0	1	0	0
Format 1	opcode				dest d				source s				source t			
Format 2	opcode				dest d				addr							

TOY Reference Card

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Format 1	opcode				dest d				source s				source t			
Format 2	opcode				dest d				addr							

#	Operation	Fmt	Pseudocode
0:	halt	1	exit(0)
1:	add	1	R[d] ← R[s] + R[t]
2:	subtract	1	R[d] ← R[s] - R[t]
3:	and	1	R[d] ← R[s] & R[t]
4:	xor	1	R[d] ← R[s] ^ R[t]
5:	shift left	1	R[d] ← R[s] << R[t]
6:	shift right	1	R[d] ← R[s] >> R[t]
7:	load addr	2	R[d] ← addr
8:	load	2	R[d] ← mem[addr]
9:	store	2	mem[addr] ← R[d]
A:	load indirect	1	R[d] ← mem[R[t]]
B:	store indirect	1	mem[R[t]] ← R[d]
C:	branch zero	2	if (R[d] == 0) pc ← addr
D:	branch positive	2	if (R[d] > 0) pc ← addr
E:	jump register	2	pc ← R[d]
F:	jump and link	2	R[d] ← pc; pc ← addr

Register 0 always 0.
Loads from `mem[FF]` from stdin.
Stores to `mem[FF]` to stdout.

TEQ on TOY 1

What is the interpretation of **`1A75`**

A. as a TOY instruction?

B. as an integer value?

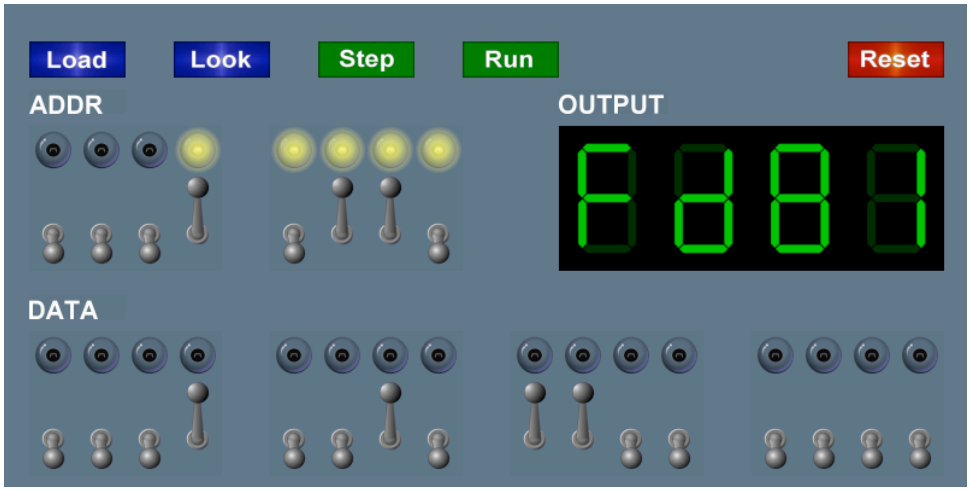
Interfacing with the TOY Machine

To enter a program or data:

- Set 8 memory address switches.
- Set 16 data switches.
- Press `Load`: data written into addressed word of memory.

To view the results of a program:

- Set 8 memory address switches.
- Press `Look`: contents of addressed word appears in lights.



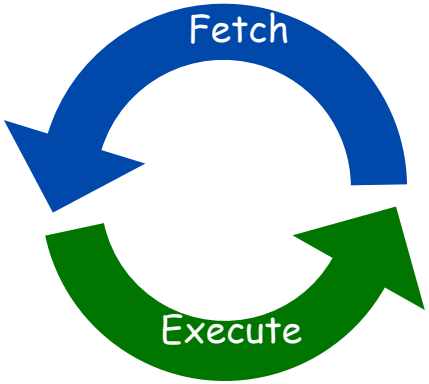
Using the TOY Machine: Run

To run the program:

- Set 8 memory address switches to address of first instruction.
- Press `Look` to set `pc` to first instruction.
- Press `Run` button to repeat fetch-execute cycle until halt opcode.

Fetch-execute cycle.

- Fetch: get instruction from memory.
- Execute: update `pc` move data to or from memory and registers, perform calculations.



Flow Control

Flow control.

- To harness the power of TOY, need loops and conditionals.
- Manipulate `pc` to control program flow.

Branch if zero. [opcode C]

- Changes `pc` depending on whether value of some register is zero.
- Used to implement: `for`, `while`, `if-else`.

Branch if positive. [opcode D]

- Changes `pc` depending on whether value of some register is positive.
- Used to implement: `for`, `while`, `if-else`.

An Example: Multiplication

Multiply. Given integers `a` and `b`, compute `c = a × b`.

TOY multiplication. No direct support in TOY hardware.

Brute-force multiplication algorithm:

- Initialize `c` to `0`.
- Add `b` to `c`, `a` times.

```
int a = 3;
int b = 9;
int c = 0;

while (a != 0) {
   c = c + b;
   a = a - 1;
}
```

brute force multiply in Java

Issues ignored. Slow, overflow, negative numbers.

Multiply

```
0A: 0003    3
0B: 0009    9          ←  inputs
0C: 0000    0          ←  output

0D: 0000    0          ←  constants
0E: 0001    1

10: 8A0A    RA ← mem[0A]                 a
11: 8B0B    RB ← mem[0B]                 b
12: 8C0D    RC ← mem[0D]                 c = 0

13: 810E    R1 ← mem[0E]                 always 1

14: CA18    if (RA == 0) pc ← 18         while (a != 0) {
15: 1CCB    RC ← RC + RB                    c = c + b
16: 2AA1    RA ← RA - R1                    a = a - 1
17: C014    pc ← 14                      }

18: 9C0C    mem[0C] ← RC
19: 0000    halt
```

loop

multiply.toy

Step-By-Step Trace

		R1	RA	RB	RC
10: 8A0A	RA ← mem[0A]		0003		
11: 8B0B	RB ← mem[0B]			0009	
12: 8C0D	RC ← mem[0D]				0000
13: 810E	R1 ← mem[0E]	0001			
14: CA18	if (RA == 0) pc ← 18				
15: 1CCB	RC ← RC + RB				0009
16: 2AA1	RA ← RA – R1		0002		
17: C014	pc ← 14				
14: CA18	if (RA == 0) pc ← 18				
15: 1CCB	RC ← RC + RB				0012
16: 2AA1	RA ← RA – R1		0001		
17: C014	pc ← 14				
14: CA18	if (RA == 0) pc ← 18				
15: 1CCB	RC ← RC + RB				001B
16: 2AA1	RA ← RA – R1		0000		
17: C014	pc ← 14				
14: CA18	if (RA == 0) pc ← 18				
18: 9C0C	mem[0C] ← RC				
19: 0000	halt				

multiply.toy

TEQ on TOY 2

What does the following TOY program leave in R2?

```
10: 7C0A
11: 7101
12: 7201
13: 5221
14: 2CC1
15: DC13
16: 0000
```

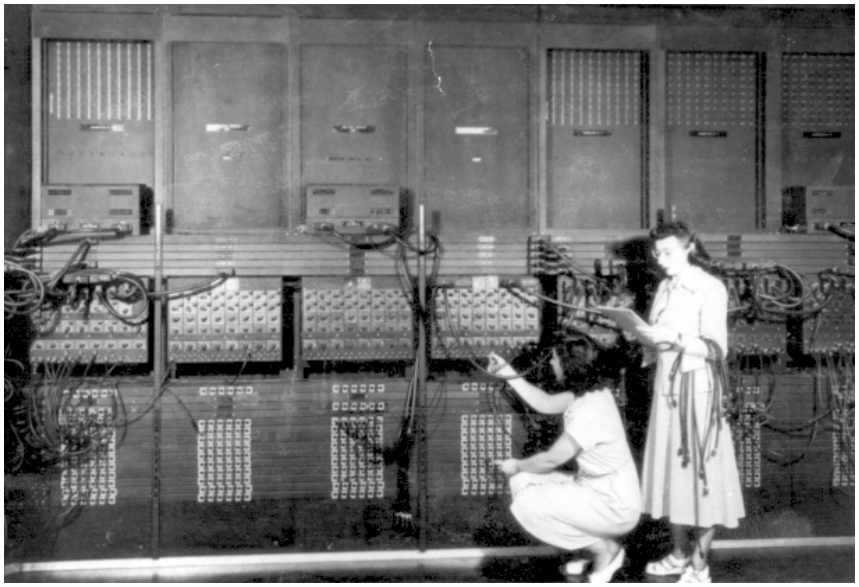
A Little History

Electronic Numerical Integrator and Calculator (ENIAC).

- First widely known general purpose electronic computer.
- Conditional jumps, programmable.
- Programming: change switches and cable connections.
- Data: enter numbers using punch cards.

30 tons
30 x 50 x 8.5 ft
17,468 vacuum tubes
300 multiply/sec

John Mauchly (left) and J. Presper Eckert (right)
http://cs.swau.edu/~durkin/articles/history_computing.html



ENIAC, Ester Gerston (left), Gloria Gordon (right)
US Army photo: http://ftp.arl.mil/ftp/historic-computers

Basic Characteristics of TOY Machine

TOY is a general-purpose computer.

- Sufficient power to perform ANY computation.
- Limited only by amount of memory and time.

Stored-program computer. [von Neumann memo, 1944]

- Data and program encoded in binary.
- Data and program stored in SAME memory.
- Can change program without rewiring.

Outgrowth of Alan Turing's work. (stay tuned)

All modern computers are general-purpose computers and have same (von Neumann) architecture.

John von Neumann

Maurice Wilkes (left)
EDSAC (right)

Harvard vs. Princeton

Harvard architecture.

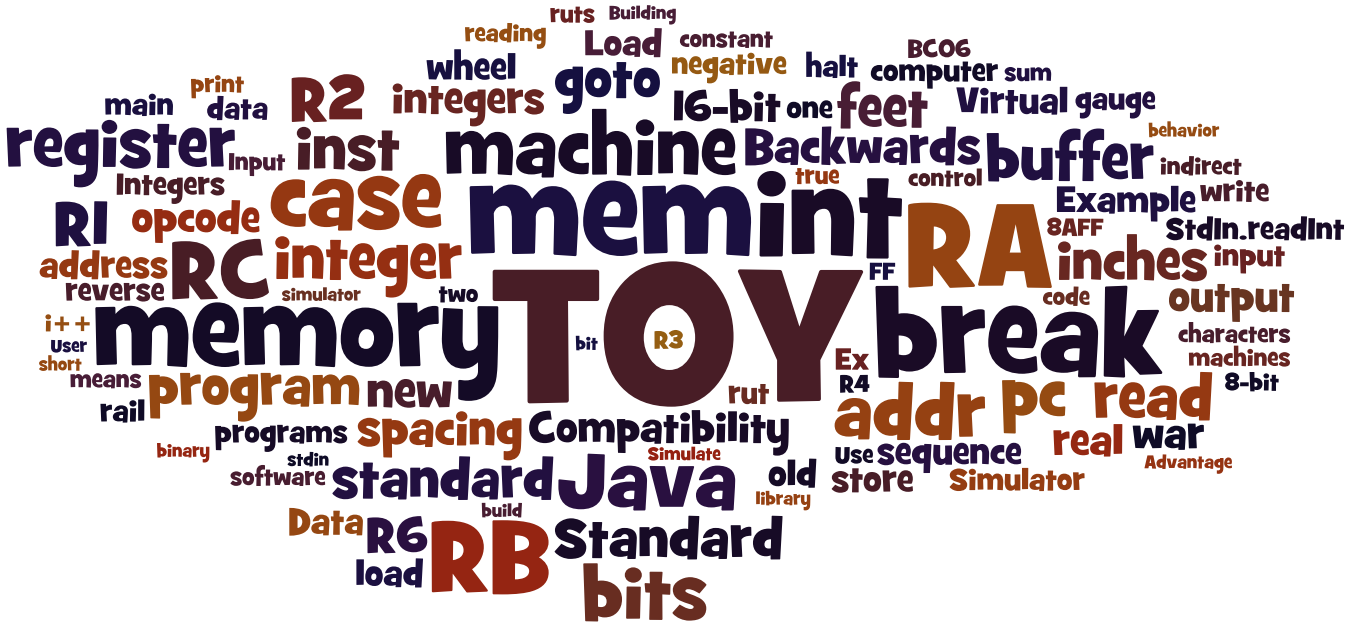
- Separate program and data memories.
- Can't load game from disk (data) and execute (program).
- Used in some microcontrollers.

Von Neumann architecture.

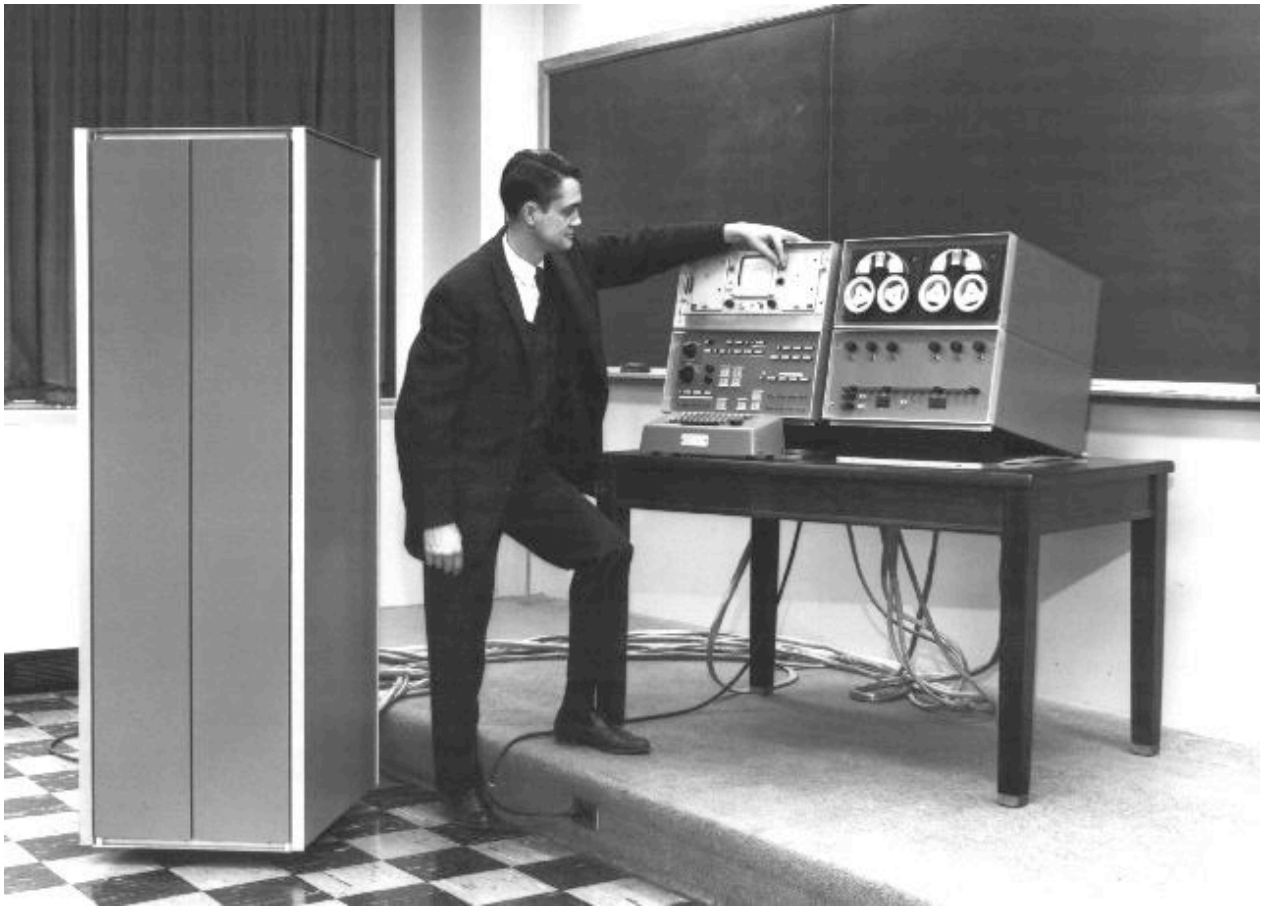
- Program and data stored in same memory.
- Used in almost all computers.

Q. What's the difference between Harvard and Princeton?

A. At Princeton, data and programs are the same.


ruts
Building
reading
Load
constant
BC06
wheel
goto
negative
halt
computer
sum
print
main
data
R2
integers
16-bit
one
feet
Virtual
gauge
behavior
register
input
inst
machine
Backwards
buffer
indirect
Integers
true
control
Example
write
RI
opcode
case
memint
RA
8AFF
StdIn.readInt
address
RC
integer
FF
inches
input
reverse
simulator
two
code
output
i++
TOY
break
memory
bit
R3
User
Ex
characters
short
machines
means
program
new
R4
8-bit
rail
rut
addr
pc
read
programs
spacing
Compatibility
real
war
binary
Simulate
Use
sequence
Advantage
stdin
software
standard
Java
old
store
Simulator
library
build
Data
R6
RB
Standard
load
bits

TOY II



LINC

What We've Learned About TOY

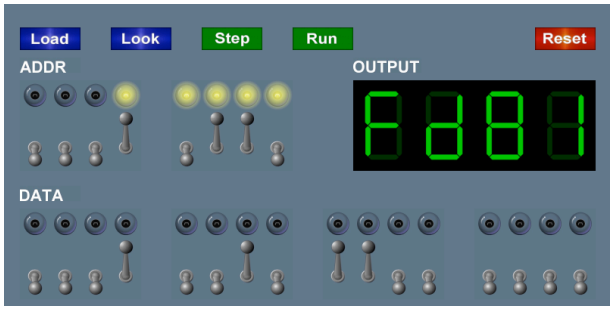
Data representation. Binary and hex.

TOY.

- Box with switches and lights.
- 16-bit memory locations, 16-bit registers, 8-bit pc.
- 4,328 bits = (255 × 16) + (15 × 16) + (8) = 541 bytes!
- von Neumann architecture.

TOY instruction set architecture. 16 instruction types.

TOY machine language programs. Variables, arithmetic, loops.



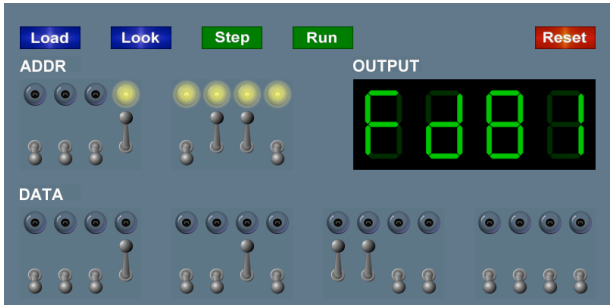
What We Do Today

Data representation. Negative numbers.

Input and output. Standard input, standard output.

Manipulate addresses. References (pointers) and arrays.

TOY simulator in Java and implications.



Data Representation

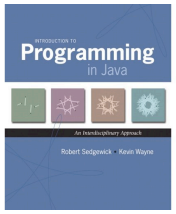
Digital World

Data is a sequence of bits. (interpreted in different ways)

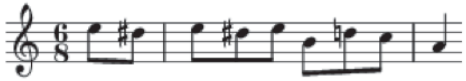
- Integers, real numbers, characters, strings, …
- Documents, pictures, sounds, movies, Java programs, …

Ex. `01110101`

- As binary integer: 1 + 4 + 16 + 32 + 64 = 117 (base ten).
- As character: 117th Unicode character = `'u'`.
- As music: 117/256 position of speaker.
- As grayscale value: 45.7% black.

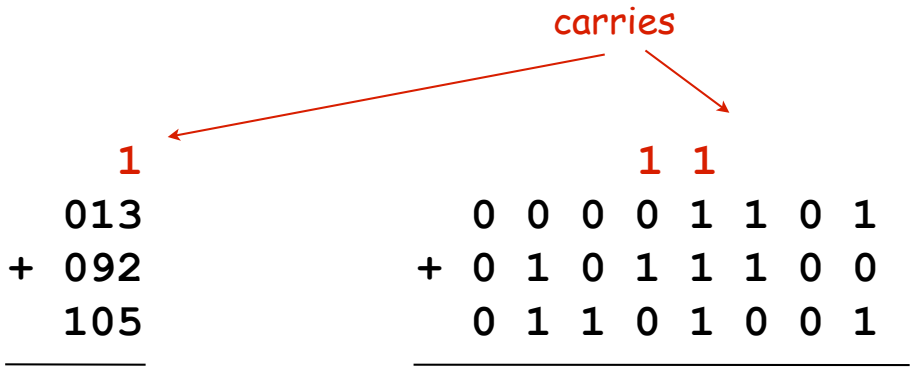



```
public class HelloWorld {

    public static void main(String[] args) {
        System.out.println("Hello, World");
    }

}
```



Adding and Subtracting Binary Numbers

Decimal and binary addition.



Subtraction. Add a negative integer.

e.g., 6 - 4 = 6 + (-4))

Q. How to represent negative integers?

Representing Negative Integers

TOY words are 16 bits each.

- We could use 16 bits to represent 0 to $2^{16} - 1$.
- We want negative integers too.
- Reserving half the possible bit-patterns for negative seems fair.

Highly desirable property. If x is an integer, then the representation of -x, when added to x, is zero.

```
    x          0 0 1 1 0 1 0 0
+(-x)       + ? ? ? ? ? ? ? ?
-------    ---------------------
    0          0 0 0 0 0 0 0 0
```

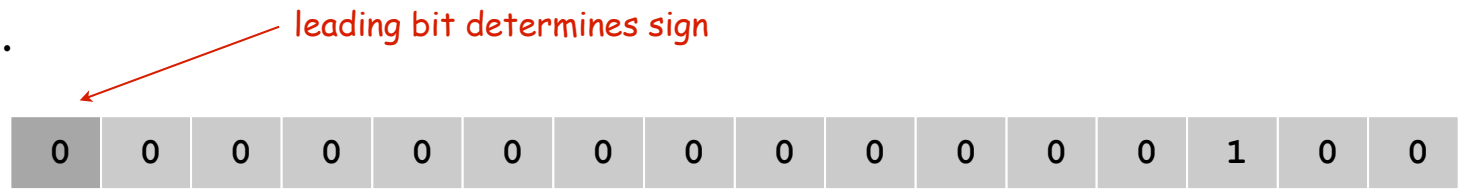
```
    x          0 0 1 1 0 1 0 0
               + 1 1 0 0 1 0 1 1
+(-x)          ---------------------
               1 1 1 1 1 1 1 1
               +               1
-------    ---------------------
    0          0 0 0 0 0 0 0 0
```

-x: flip bits and add 1

Two's Complement Integers

To compute -x from x:

- Start with x.



- Flip bits.

- Add one.

Two's Complement Integers

		15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
dec	hex	binary															
+32767	**7FFF**	**0**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**
		. . .															
+4	**0004**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**0**	**0**
+3	**0003**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**1**
+2	**0002**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**0**
+1	**0001**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**
+0	**0000**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
-1	**FFFF**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**
-2	**FFFE**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**0**
-3	**FFFD**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**0**	**1**
-4	**FFFC**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**0**	**0**
		. . .															
-32768	**8000**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**

Properties of Two's Complement Integers

Properties.

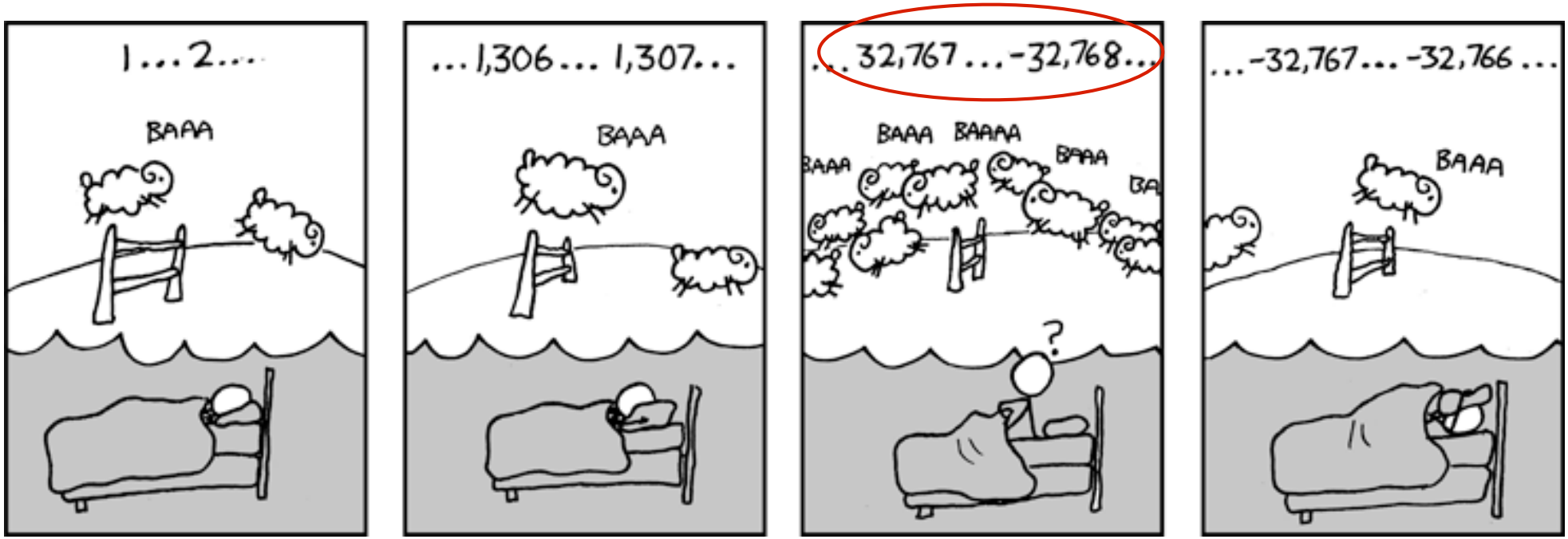
- Leading bit (bit 15) signifies sign.
- `0000000000000000` represents zero.
- Negative integer -x represented by 2^{16} - x.
- Addition is easy.
- Checking for arithmetic overflow is easy.

Not-so-nice property. Can represent one more negative integer than positive integer.

-32,768 = -2^{15}

32,767 = $2^{15}-1$

Remark. Java `int` data type is 32-bit two's complement integer.



http://xkcd.com/571/

Representing Other Primitive Data Types in TOY

Bigger integers. Use two 16-bit words per `int`.

Real numbers.

- Use "floating point" (like scientific notation).
- Use four 16-bit words per `double`.

Characters.

- Use ASCII code (8 bits / character).
- Pack two characters per 16-bit word.

Note. Real microprocessors add hardware support for `int` and `double`.

Standard Input and Output

Standard Output

Standard output.

- Writing to memory location `FF` sends one word to TOY stdout.
- Ex. `9AFF` writes the integer in register `A` to stdout.

```
00: 0000    0
01: 0001    1

10: 8A00    RA ← mem[00]            a = 0
11: 8B01    RB ← mem[01]            b = 1
                                    do {
12: 9AFF    write RA to stdout          print a
13: 1AAB    RA ← RA + RB                a = a + b
14: 2BAB    RB ← RA - RB                b = a - b
15: DA12    if (RA > 0) goto 12     } while (a > 0)
16: 0000    halt
```

fibonacci.toy

standard output

```
0000
0001
0001
0002
0003
0005
0008
000D
0015
0022
0037
0059
0090
00E9
0179
0262
03DB
063D
0A18
1055
1A6D
2AC2
452F
6FF1
```

Standard Input

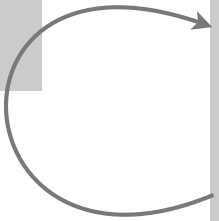
Standard input.

- Loading from memory address `FF` loads one word from TOY stdin.
- Ex. `8AFF` reads an integer from stdin and store it in register `A`.

Ex: read in a sequence of integers and print their sum.

- In Java, stop reading when EOF.
- In TOY, stop reading when user enters `0000`.

```
while (!StdIn.isEmpty()) {
   a = StdIn.readInt();
   sum = sum + a;
}
StdOut.println(sum);
```



```
00: 0000   0


10: 8C00   RC <- mem[00]
11: 8AFF   read RA from stdin
12: CA15   if (RA == 0) pc ← 15
13: 1CCA   RC ← RC + RA
14: C011   pc ← 11
15: 9CFF   write RC
16: 0000   halt
```

```
00AE
0046
0003
0000
00F7
```

Standard Input and Output: Implications

Standard input and output enable you to:

- Get information out of machine.
- Put information from real world into machine.
- Process more information than fits in memory.
- Interact with the computer while it is running.

TEQ on TOY 3

What does the following TOY program do?

```
10: 7C0A
11: 7101
12: 7201
13: 92FF
14: 5221
15: 2CC1
16: DC13
17: 0000
```

Pointers

Load Address (a.k.a. Load Constant)

Load address. [opcode 7]

- Loads an 8-bit integer into a register.
- `7A30` means load the value `30` into register `A`.

Applications.

- Load a small constant into a register.
- Load an 8-bit memory address into a register.

register stores "pointer" to a memory cell

```
a = 0x30;
```

Java code

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	1	1	0	1	0	0	0	1	1	0	0	0	0
7_{16}				A_{16}				3_{16}				0_{16}			
opcode				dest d				addr							

Arrays in TOY

TOY main memory is a giant array.

- Can access memory cell `30` using load and store.
- `8C30` means load `mem[30]` into register `C`.
- Goal: access memory cell `i` where `i` is a variable.

...	...
30	0000
31	0001
32	0001
33	0002
34	0003
35	0005
36	0008
37	000D
...	...

TOY memory

Load indirect. [opcode A]

- `AC06` means load `mem[R6]` into register `C`.

a variable index

Store indirect. [opcode B]

- `BC06` means store contents of register `C` into `mem[R6]`.

a variable index

```
for (int i = 0; i < N; i++)
   a[i] = StdIn.readInt();
for (int i = 0; i < N; i++)
   StdOut.println(a[N-i-1]);
```

TOY Implementation of Reverse

TOY implementation of reverse.

- Read in a sequence of integers and store in memory `30, 31, 32, ...`
- Stop reading if `0000`.
- Print sequence in reverse order.

```
10: 7101  R1 ← 0001                    constant 1
11: 7A30  RA ← 0030                    a[]
12: 7B00  RB ← 0000                    n


                                       while(true) {
13: 8CFF  read RC                         c = StdIn.readInt();
14: CC19  if (RC == 0) goto 19            if (c == 0) break;
15: 16AB  R6 ← RA + RB                    memory address of a[n]
16: BC06  mem[R6] ← RC                    a[n] = c;
17: 1BB1  RB ← RB + R1                    n++;
18: C013  goto 13                      }
```

read in the data

TOY Implementation of Reverse

TOY implementation of reverse.

- Read in a sequence of integers and store in memory `30, 31, 32, ...`
- Stop reading if `0000`.
- Print sequence in reverse order.

```
10: 7101  R1 ← 0001                          constant 1
11: 7A30  RA ← 0030                          a[]
12: 7B00  RB ← 0000                          n



                                             while(true) {
13: 8CFF  read RC                               c = StdIn.readInt();
14: CC19  if (RC == 0) goto 19                  if (c == 0) break;
15: 16AB  R6 ← RA + RB                          memory address of a[n]
16: BC06  mem[R6] ← RC                          a[n] = c;
17: 1BB1  RB ← RB + R1                          n++;
18: C013  goto 13                            }
```

print in reverse order

Unsafe Code at any Speed

Q. What happens if we make array start at 00 instead of 30?

```
10: 7101   R1 ← 0001                  constant 1
11: 7A00   RA ← 0000                  a[]
12: 7B00   RB ← 0000                  n

                                      while(true) {
13: 8CFF   read RC                      c = StdIn.readInt();
14: CC19   if (RC == 0) goto 19         if (c == 0) break;
15: 16AB   R6 ← RA + RB                 address of a[n]
16: BC06   mem[R6] ← RC                 a[n] = c;
17: 1BB1   RB ← RB + R1                 n++;
18: C013   goto 13                    }
```

```
% more crazy8.txt
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
8888 8810
98FF C011
```

A. With enough data, becomes a self-modifying program

- can overflow buffer
- and run arbitrary code!

What Can Happen When We Lose Control (in C or C++)?

Buffer overrun.

- Array `buffer[]` has size 100.
- User might enter 200 characters.
- Might lose control of machine behavior.

```
#include <stdio.h>
int main(void) {
   char buffer[100];
   scanf("%s", buffer);
   printf("%s\n", buffer);
   return 0;
}
```

unsafe C program

Consequences. Viruses and worms.

What Can Happen When We Lose Control (in C or C++)?

Buffer overrun.

- Array `buffer[]` has size 100.
- User might enter 200 characters.
- Might lose control of machine behavior.

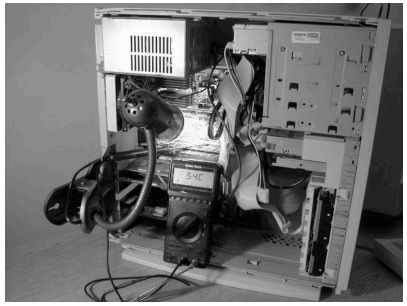
```
#include <stdio.h>
int main(void) {
   char buffer[100];
   scanf("%s", buffer);
   printf("%s\n", buffer);
   return 0;
}
```

unsafe C program

Consequences. Viruses and worms.

Java enforces security.

- Type safety.
- Array bounds checking.
- Not foolproof.



shine 50W bulb at DRAM
[Appel-Govindavajhala '03]

Buffer Overrun Example: JPEG of Death

Microsoft Windows JPEG bug. [September, 2004]

- Step 1. User views malicious JPEG in IE or Outlook.
- Step 2. Machine is Owned.
- Data becomes code by exploiting buffer overrun in GDI+ library.

Fix. Update old library with patched one.

but many applications install independent copies of GDI library

Moral.

- Not easy to write error-free software.
- Embrace Java security features.
- Don't try to maintain several copies of the same file.
- Keep your OS patched.

Dumping

Q. Work all day to develop operating system. How to save it?

A. Write short program `dump.toy` and run it to dump contents of memory onto tape.

```
00: 7001    R1 ← 0001
01: 7210    R2 ← 0010                    i = 10
02: 73FF    R3 ← 00FF
                                         do {
03: AA02    RA ← mem[R2]                    a = mem[i]
04: 9AFF    write RA                        print a
05: 1221    R2 ← R2 + R1                    i++
06: 2432    R4 ← R3 - R2
07: D403    if (R4 > 0) goto 03 } while (i < 255)
08: 0000    halt
```

dump.toy

Booting

Q. How do you get it back?

A. Write short program `boot.toy` and run it to read contents of memory from tape.

```
00: 7001    R1 ← 0001
01: 7210    R2 ← 0010                    i = 10
02: 73FF    R3 ← 00FF
                                         do {
03: 8AFF    read RA                         read a
04: BA02    mem[R2] ← RA                    mem[i] = a
05: 1221    R2 ← R2 + R1                    i++
06: 2432    R4 ← R3 - R2
07: D403    if (R4 > 0) goto 03 } while (i < 255)
08: 0000    halt
```

boot.toy

Simulating the TOY machine

TOY Simulator

Goal. Write a program to "simulate" the behavior of the TOY machine.

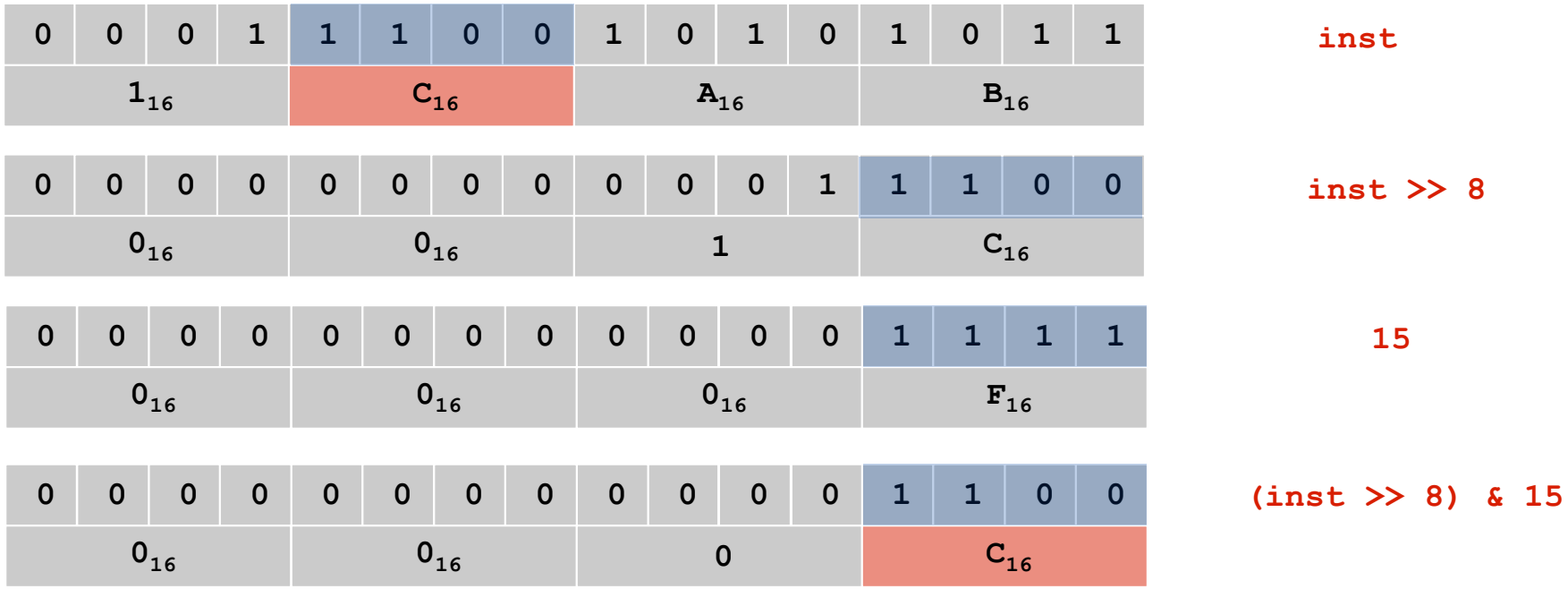
- TOY simulator in Java.
- TOY simulator in TOY!

```
public class TOY
{
   public static void main(String[] args)
   {
      int pc    = 0x10;          // program counter
      int[] R   = new int[16];   // registers
      int[] mem = new int[256]; // main memory

      // READ .toy FILE into mem[10..]

      while (true)
      {
         int inst = mem[pc++];  // fetch and increment
         // DECODE
         // EXECUTE
      }
   }
}
```

```
% more add-stdin.toy
8C00   <-- TOY program to load at 10
8AFF
CA15
1CCA
C011
9CFF
0000

% java TOY add-stdin.toy
00AE   <-- standard input
0046
0003
0000
00F7   <-- standard output
```

TOY Simulator: Fetch

Ex. Extract destination register of `1CAB` by shifting and masking.




```
int inst = mem[pc++];           // fetch and increment
int op   = (inst >> 12) &  15;  // opcode    (bits 12-15)
int d    = (inst >>  8) &  15;  // dest d    (bits 08-11)
int s    = (inst >>  4) &  15;  // source s  (bits 04-07)
int t    = (inst >>  0) &  15;  // source t  (bits 00-03)
int addr = (inst >>  0) & 255;  // addr      (bits 00-07)
```

TOY Simulator: Execute

```
if (op == 0) break;        // halt

switch (op)
{
   case  1: R[d] = R[s] +  R[t];      break;
   case  2: R[d] = R[s] -  R[t];      break;
   case  3: R[d] = R[s] &  R[t];      break;
   case  4: R[d] = R[s] ^  R[t];      break;
   case  5: R[d] = R[s] << R[t];      break;
   case  6: R[d] = R[s] >> R[t];      break;
   case  7: R[d] = addr;              break;
   case  8: R[d] = mem[addr];         break;
   case  9: mem[addr] = R[d];         break;
   case 10: R[d] = mem[R[t]];         break;
   case 11: mem[R[t]] = R[d];         break;
   case 12: if (R[d] == 0) pc = addr; break;
   case 13: if (R[d] >  0) pc = addr; break;
   case 14: pc = R[d]; pc; pc = addr; break;
   case 15: R[d] = pc; pc = addr;     break;
}
```

TOY Simulator: Omitted Details

Omitted details.

- Register 0 is always 0.
  - reset `R[0]=0` after each fetch-execute step

- Standard input and output.
  - if `addr` is `FF` and opcode is load (indirect) then read in data
  - if `addr` is `FF` and opcode is store (indirect) then write out data

- TOY registers are 16-bit integers; program counter is 8-bit.
  - Java `int` is 32-bit; Java `short` is 16-bit
  - use casts and bit-whacking

Complete implementation. See `TOY.java` on booksite.

Simulation

Important ideas stemming from simulation.

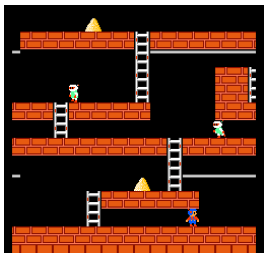
- Backwards compatiblity
- Virtual machines
- Layers of abstraction

Backwards Compatibility

Building a new computer? Need a plan for old software.

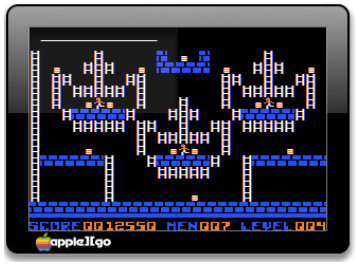
Two possible approaches

- Rewrite software (costly, error-prone, boring, and time-consuming).
- Simulate old computer on new computer.



Lode Runner

Apple IIe



Mac OS X Apple IIe emulator widget running Lode Runner

Ancient programs still running on modern computers.

- Payroll
- Power plants
- Air traffic control
- Ticketron.
- Games.

Backwards Compatibility

Q. Why is standard US rail gauge 4 feet, 8.5 inches?

A. Same spacing as wheel ruts on old English roads.

Q. Why is wheel rut spacing 4 feet, 8.5 inches?

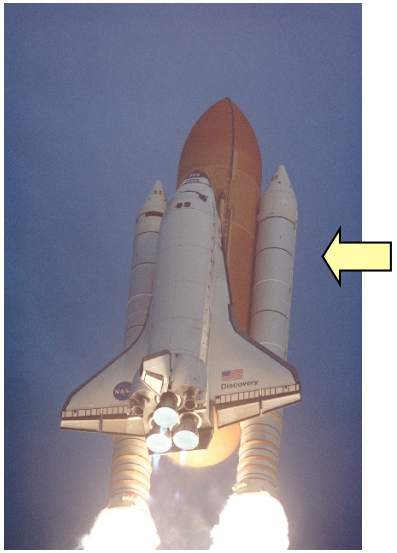
A. For Roman war chariots.

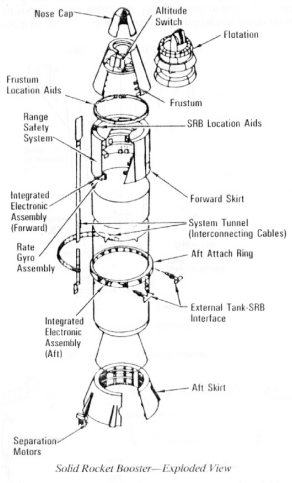
Q. Why is war chariot rut spacing 4 feet, 8.5 inches?

A. Fits "back ends" of two war horses!

Effects of Backwards Compatibility: example 1

Q. Why is Space Shuttle SRB long and narrow?





A. Fits on standard US rail guage.

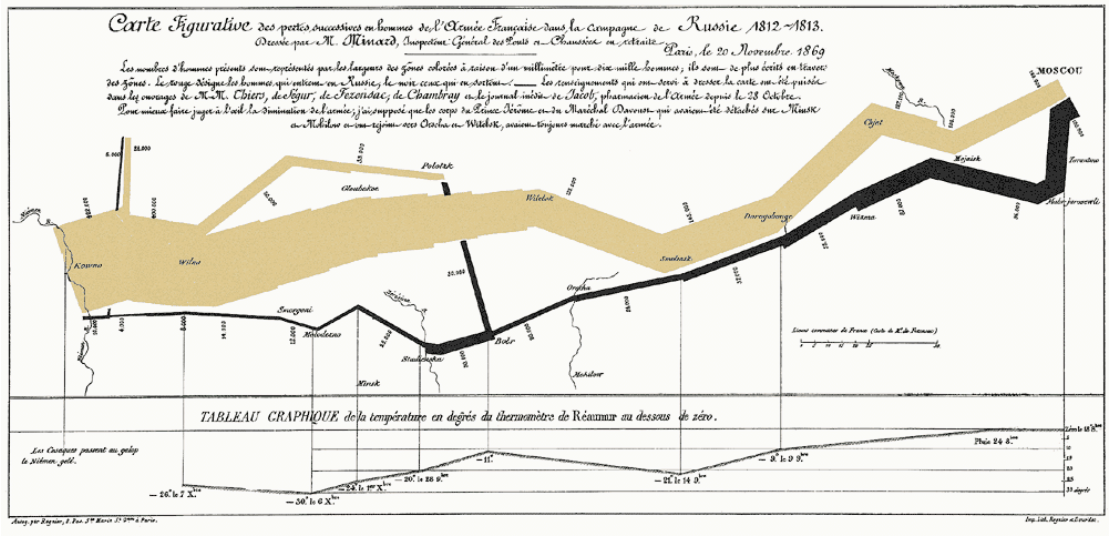
. . .

A. Fits "back ends" of two war horses!

Effects of Backwards Compatibility: Example 2

Napoleon's march on Russia.

- Progress slower than expected.
- Eastern European ruts didn't match Roman gauge.
- Stuck in the field during Russian winter instead of Moscow.
- Lost war.



Lessons.

- Maintaining backwards compatibility can lead to inelegance and inefficiency.
- Maintaining backwards compatibility is Not Always A Good Thing.
- May need fresh ideas to conquer civilized world.

Virtual machines

Building a new rocket? Simulate it to test it.

- Issue 1: Simulation may not reflect reality.
- Issue 2: May not be able to afford simulation.

Building a new computer? Simulate it to test it.

- Advantage 1: Simulation is reality (it defines the new machine).
- Advantage 2: Can develop software without having machine.
- Advantage 3: Can simulate machines you wouldn't build.

Example 1: Operating systems implement Virtual Memories that are much larger than real memories by simulating programs and going to disk or the web to reference "memory"

Example 2: Operating systems implement multiple Virtual Machines on a single real machine by keeping track of multiple PCs and rotating control to the different machines

Example 3: The Java Virtual Machine provides machine independence for Java programs. It is simulated on the real machine (PC, cellphone, toaster) you happen to be using.

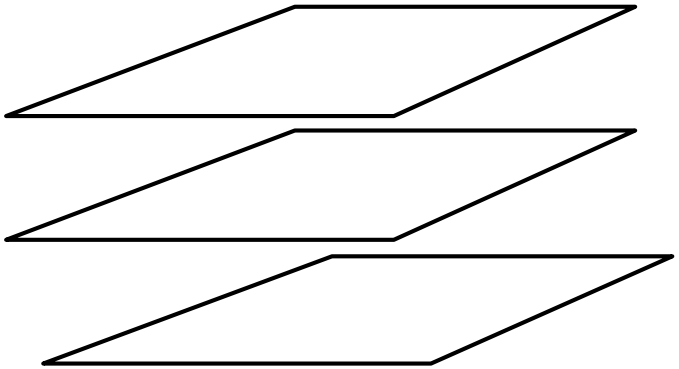
Example 4: The Amazon Virtual Computing Environment provides "computing in the cloud". It gives the illusion that your device has the power of a web server farm.

Layers of Abstraction

Is TOY real?

programmer

Is Java real?



Java language specification

Java virtual machine

Instruction set architecture

machine

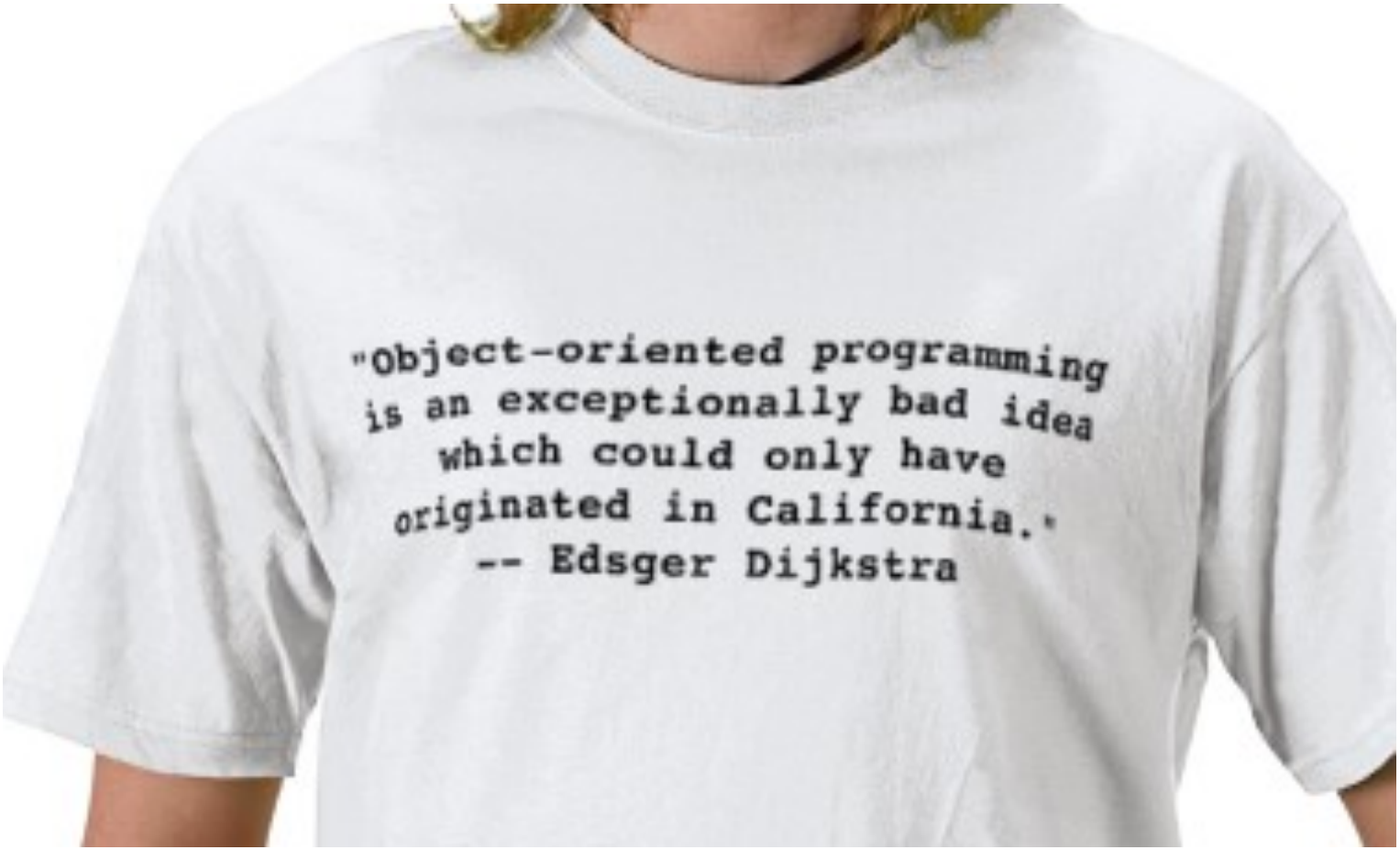
Approaching a new problem?

- build an (abstract) language for expressing solutions
- design an (abstract) machine to execute the language
- food for thought: Why build the machine? [instead, simulate it!]

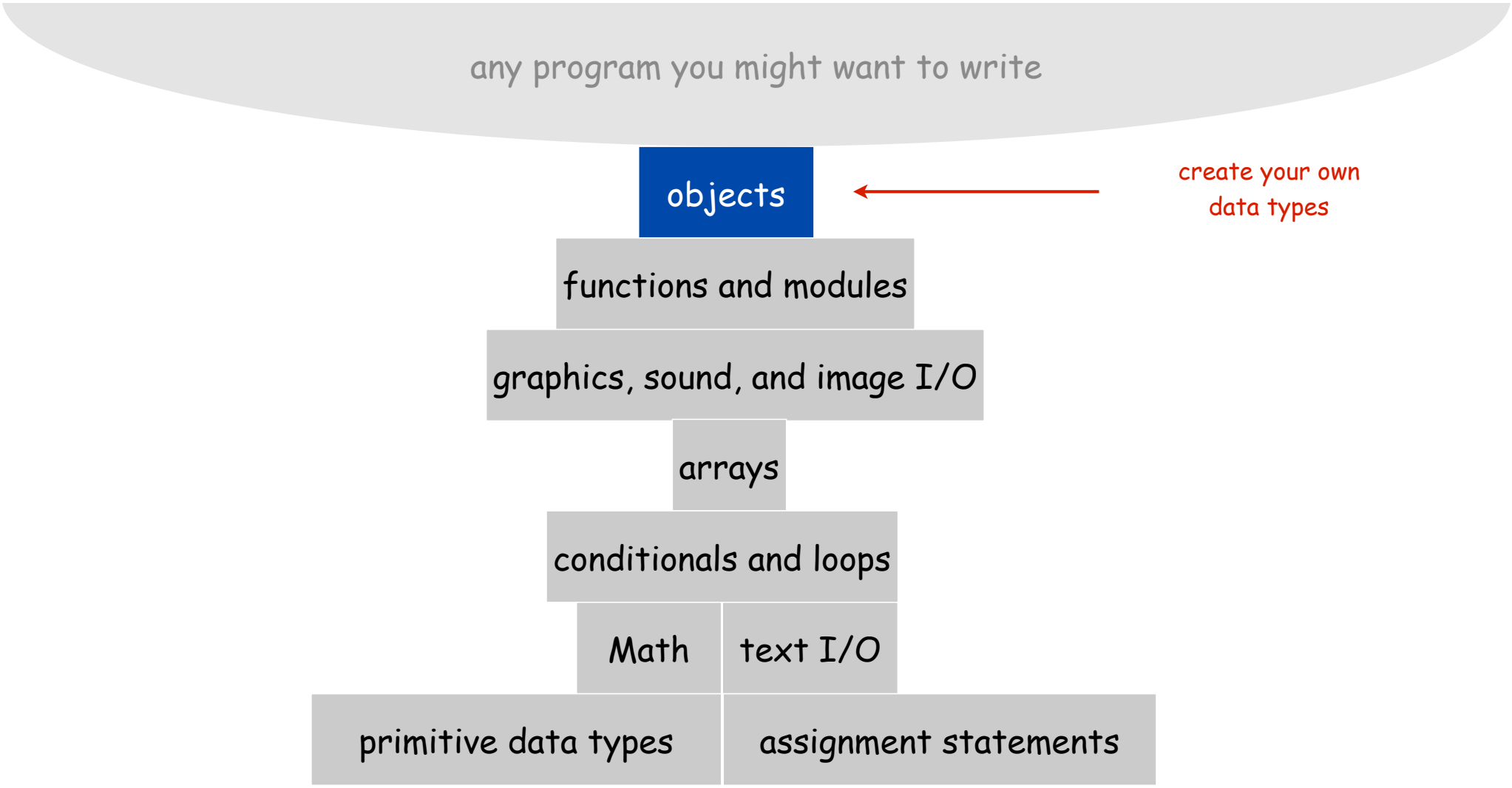
Examples: MATLAB, BLAST, AMP ....


known
streams
two
variable
instance
consider
strings
static
processing
one
images
Programming
args
data
argument
program
creates
pixel
gene
input
web
primitive
example
object
values
Java's
target
memory
many
string
implement
process
line
statement
codon
set
value
Java
cl
String
int
several
color
genome
Data
programs
objects
find
now
Types
output
multiple
class
colors
using
Charge
java
change
start
reference
arrays
stream
name
API
image
file
Color
type
Object-Oriented
also
types
price
point
TAG
new
code
computer
method
operations
standard
double
variables
need
public
methods
create
Picture
programming
use
basic
just
grayscale
potential
source
array
write
return
information
files
client

3.1 Data Types



A Foundation for Programming



Data Types

Data type. Set of values and operations on those values.

Primitive types.

- values directly map to machine representations
- operations directly translate to machine instructions.

Data Type	Set of Values	Operations
`boolean`	`true`, `false`	not, and, or, xor
`int`	-2^{31} to $2^{31} - 1$	add, subtract, multiply
`double`	any of 2^{64} possible reals	add, subtract, multiply

We want to write programs that process other types of data.

- Colors, pictures, strings, input streams, ...
- Complex numbers, vectors, matrices, polynomials, ...
- Points, polygons, charged particles, celestial bodies, ...

Objects

Object. Holds a data type value; variable name refers to object.

Impact. Enables us to create our own data types; define operations on them; and integrate into our programs.

Data Type	Set of Values	Operations
`Color`	24 bits	get red component, brighten
`Picture`	2D array of colors	get/set color of pixel (i, j)
`String`	sequence of characters	length, substring, compare

Constructors and Methods

To construct a new object: Use keyword `new` and name of data type.

To apply an operation: Use name of object, the dot operator, and the name of the method.

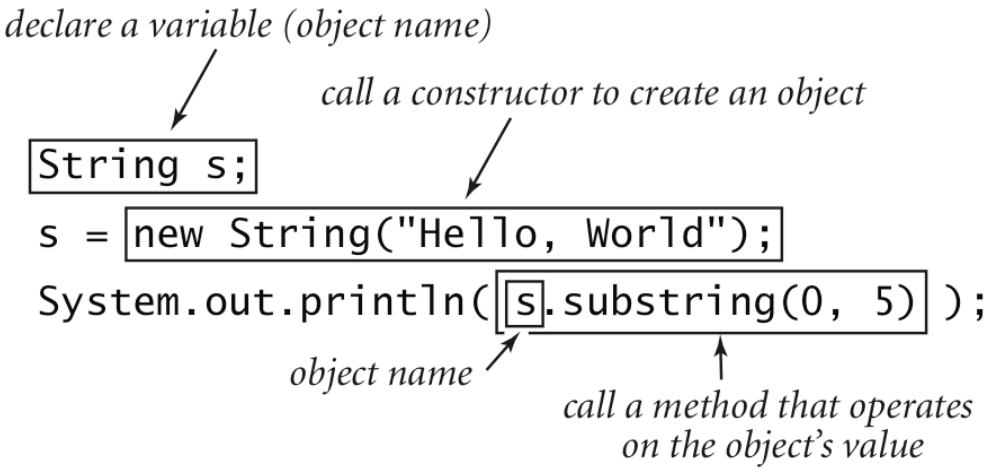


Image Processing

Color Data Type

Color. A sensation in the eye from electromagnetic radiation.

Set of values. [RGB representation] 256^3 possible values, which quantify the amount of red, green, and blue, each on a scale of 0 to 255.

R	G	B	Color
255	0	0	
0	255	0	
0	0	255	
255	255	255	
0	0	0	
255	0	255	
105	105	105	

Color Data Type

Color. A sensation in the eye from electromagnetic radiation.

Set of values. [RGB representation] 256^3 possible values, which quantify the amount of red, green, and blue, each on a scale of 0 to 255.

API (Application Programming Interface) specifies set of operations.

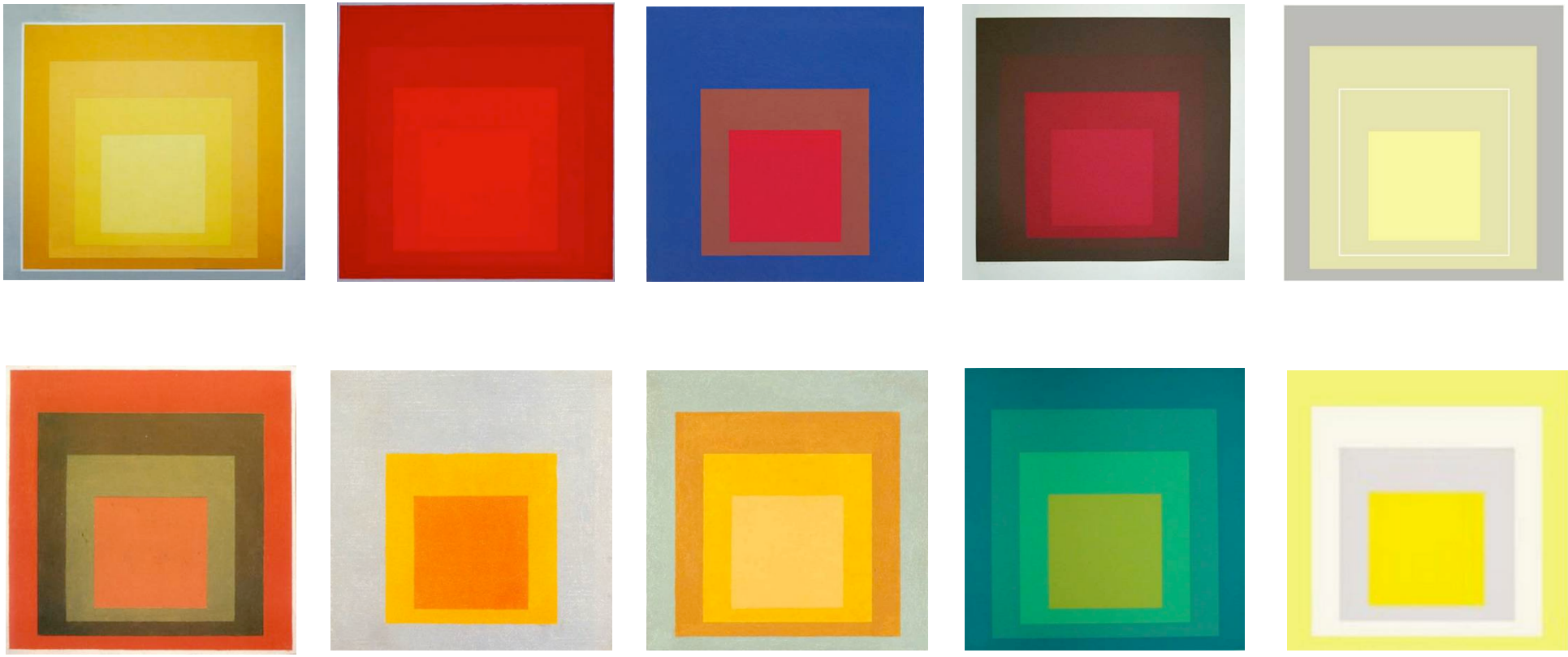
public class java.awt.Color

	Color(int r, int g, int b)	
int	getRed()	*red intensity*
int	getGreen()	*green intensity*
int	getBlue()	*blue intensity*
Color	brighter()	*brighter version of this color*
Color	darker()	*darker version of this color*
String	toString()	*string representation of this color*
boolean	equals(Color c)	*is this color's value the same as c's?*

http://java.sun.com/j2se/1.5.0/docs/api/java/awt/Color.html

Albers Squares

Josef Albers. Revolutionized the way people think about color.



Homage to the Square by Josef Albers (1949-1975)

Albers Squares

Josef Albers. Revolutionized the way people think about color.

blue gray

```
% java AlbersSquares 9 90 166 100 100 100
```

Using Colors in Java

```
import java.awt.Color;
```

to access `Color` library

```
public class AlbersSquares
{
   public static void main(String[] args)
   {
      int r1 = Integer.parseInt(args[0]);
      int g1 = Integer.parseInt(args[1]);
      int b1 = Integer.parseInt(args[2]);
      Color c1 = new Color(r1, g1, b1);
```

first color

```
      int r2 = Integer.parseInt(args[3]);
      int g2 = Integer.parseInt(args[4]);
      int b2 = Integer.parseInt(args[5]);
      Color c2 = new Color(r2, g2, b2);
```

second color

```
      StdDraw.setPenColor(c1);
      StdDraw.filledSquare(.25, .5, .2);
      StdDraw.setPenColor(c2);
      StdDraw.filledSquare(.25, .5, .1);
```

first square

```
      StdDraw.setPenColor(c2);
      StdDraw.filledSquare(.75, .5, .2);
      StdDraw.setPenColor(c1);
      StdDraw.filledSquare(.75, .5, .1);
   }
}
```

second square

Monochrome Luminance

Monochrome luminance. Effective brightness of a color.

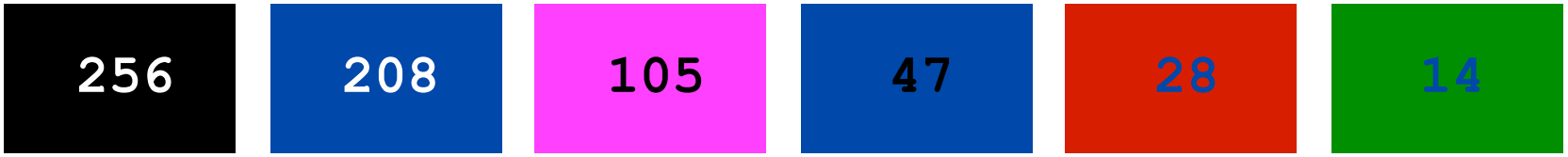
NTSC formula. $Y = 0.299r + 0.587g + 0.114b.$

```
import java.awt.Color;


public class Luminance
{
   public static double lum(Color c)
   {
      int r = c.getRed();
      int g = c.getGreen();
      int b = c.getBlue();
      return .299*r + .587*g + .114*b;
   }
}
```

Color Compatibility

Q. Which font colors will be most readable with which background colors on computer monitors and cell phone screens?

A. Rule of thumb: difference in luminance should be ≥ 128.




```
public static boolean compatible(Color a, Color b)
{
   return Math.abs(lum(a) - lum(b)) >= 128.0;
}
```

Grayscale

Grayscale. When all three R, G, and B values are the same, resulting color is on grayscale from 0 (black) to 255 (white).

Convert to grayscale. Use luminance to determine value.

```
public static Color toGray(Color c)
{
   int y = (int) Math.round(lum(c));
   Color gray = new Color(y, y, y);
   return gray;
}
```

round `double` to nearest `int`

red	*green*	*blue*		
9	90	166	*this color*	
74	74	74	*grayscale version*	
0	0	0	*black*	

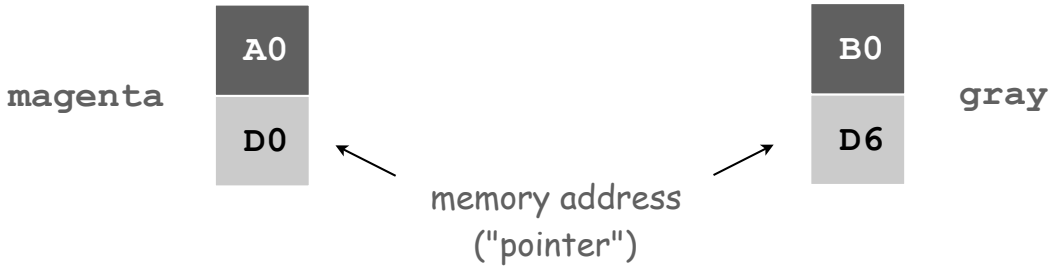
$0.299 * 9 + 0.587 * 90 + 0.114 * 166 = 74.445$

Bottom line. We are writing programs that manipulate color.

OOP Context for Color

Possible memory representation (in TOY).

D0	D1	D2	D3	D4	D5	D6	D7	D8
255	0	255	0	0	0	105	105	105



Object reference is analogous to variable name.

- We can manipulate the value that it holds.
- We can pass it to (or return it from) a method.

References

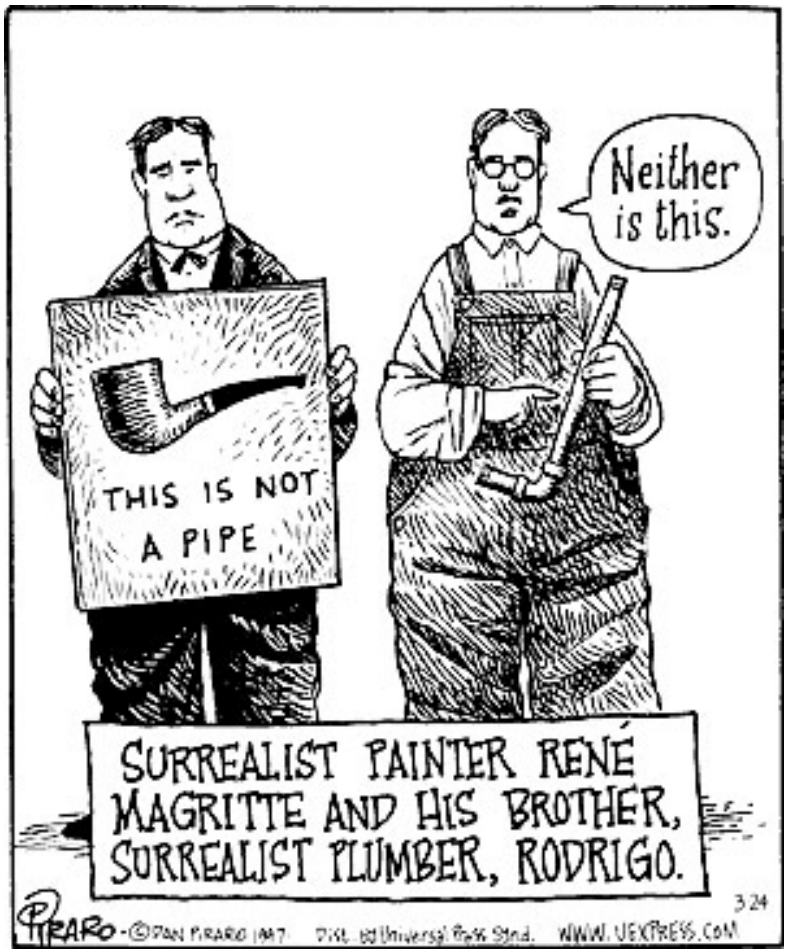
René Magritte. "This is not a pipe."

Java. This is not a color.

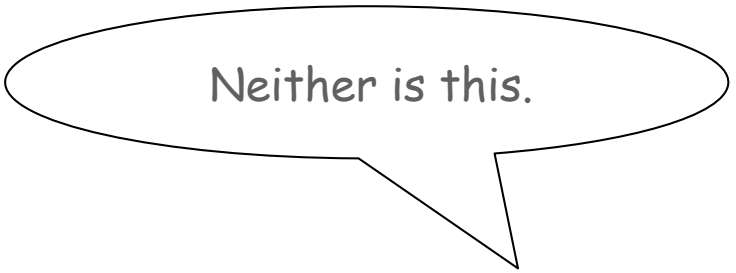
```
Color sienna = new Color(160, 82,  45);
Color c = sienna.darker();
```

OOP. Natural vehicle for studying abstract models of the real world.

This is Not a Pipe

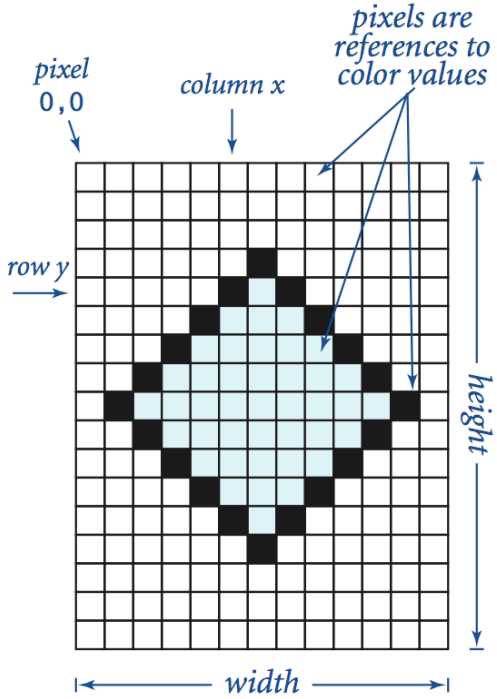


Dan Piraro, `http://www.uexpress.com`




```
% java RandomSeq 10000 | java Average
```

Picture Data Type



Raster graphics. Basis for image processing.

Set of values. 2D array of `Color` objects (pixels).

API.

`public class Picture`

`Picture(String filename)`	*create a picture from a file*
`Picture(int w, int h)`	*create a blank* `w`*-by-*`h` *picture*
`int width()`	*return the width of the picture*
`int height()`	*return the height of the picture*
`Color get(int i, int j)`	*return the color of pixel (i, j)*
`void set(int i, int j, Color c)`	*set the color of pixel (i, j) to* `c`
`void show()`	*display the image in a window*
`void save(String filename)`	*save the image to a file*

Image Processing: Grayscale Filter

Goal. Convert color image to grayscale according to luminance formula.

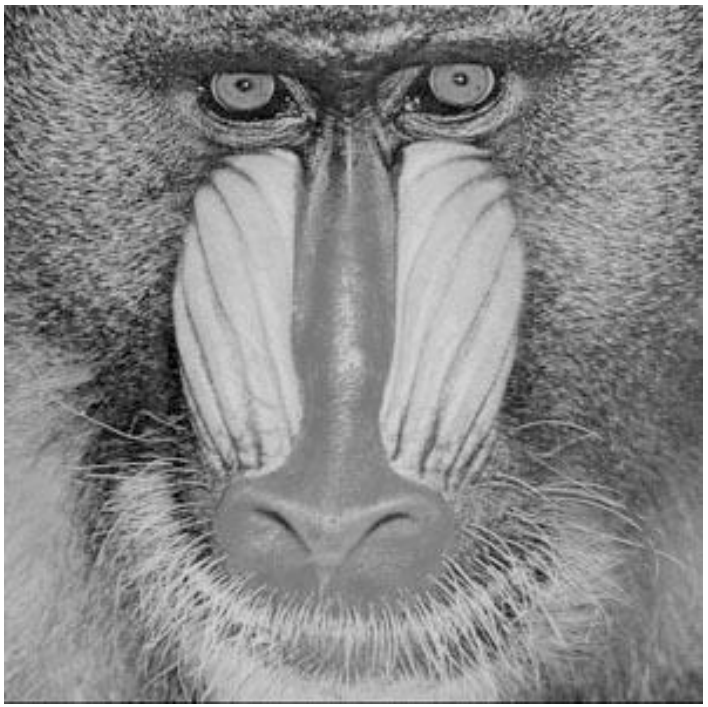
```
import java.awt.Color;

public class Grayscale
{
   public static void main(String[] args)
   {
      Picture pic = new Picture(args[0]);
      for (int i = 0; i < pic.width(); i++)
         for (int j = 0; j < pic.height(); j++)
         {
            Color color = pic.get(i, j);
            Color gray  = Luminance.toGray(color);
            pic.set(i, j, gray);
         }

      pic.show();
   }
}
```

set each pixel to gray

Image Processing: Grayscale Filter

Goal. Convert color image to grayscale according to luminance formula.

mandrill.jpg



```
% java Grayscale mandrill.jpg
```

TEQ on Image Processing 1

What does the following code do? (Easy question!)

```
Picture pic = new Picture(args[0]);
for (int i = 0; i < pic.width(); i++)
   for (int j = 0; j < pic.height(); j++)
      pic.set(i, j, pic.get(i, j)); pic.show();
```

TEQ on Image Processing 2

What does the following code do? (Hard question.)

```
Picture pic = new Picture(args[0]);
for (int i = 0; i < pic.width(); i++)
   for (int j = 0; j < pic.height(); j++)
      pic.set(i, pic.height()-j-1, pic.get(i, j));
pic.show();
```

TEQ on Image Processing 3

What does the following code do?

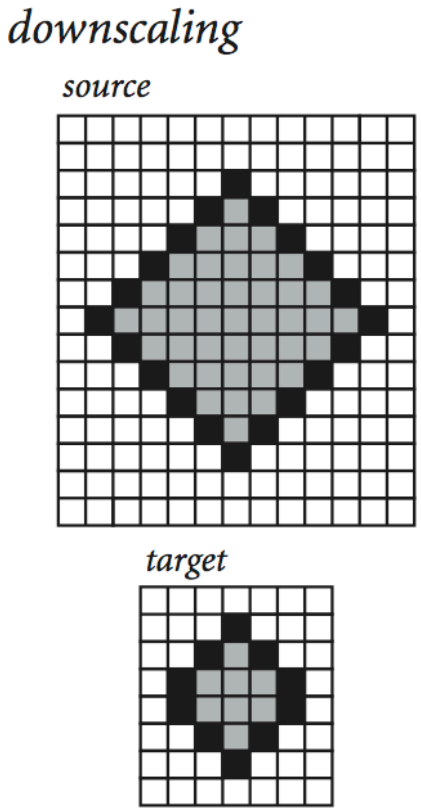
```
Picture source = new Picture(args[0]);
int width  = source.width();
int height = source.height();
Picture target = new Picture(width, height);
for (int i = 0; i < width; i++)
   for (int j = 0; j < height; j++)
      target.set(i, height-j-1, source.get(i, j));
target.show();
```

Image Processing: Scaling Filter

Goal. Shrink or enlarge an image to desired size.

Downscaling. To shrink in half, delete half the rows and columns.

Upscaling. To enlarge to double, replace each pixel by 4 copies.



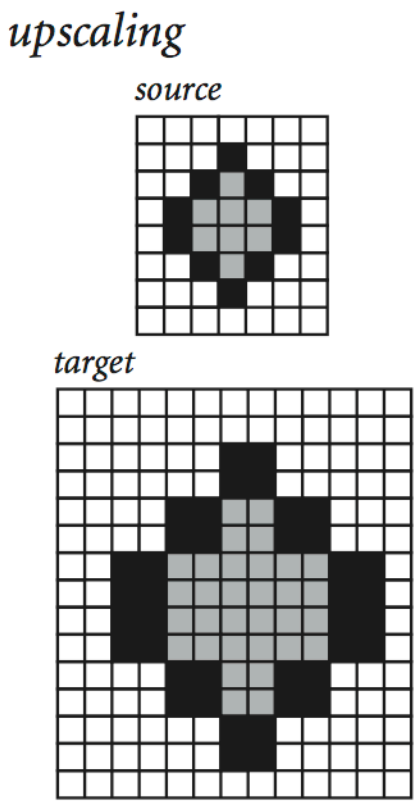


Image Processing: Scaling Filter

Goal. Shrink or enlarge an image to desired size.

Uniform strategy. To convert from w_s-by-h_s to w_t-by-h_t:

- Scale row index by w_s / w_t.
- Scale column index by h_s / h_t.
- Set color of pixel (i, j) in target image to color of pixel $(i \times w_s / w_t,\ j \times h_s / h_t)$ in source image.

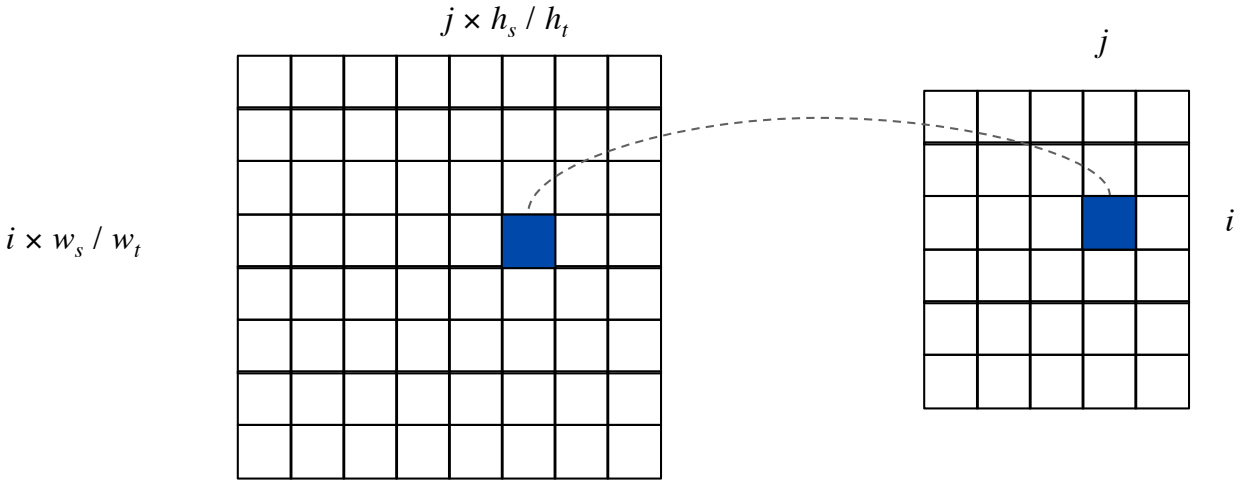
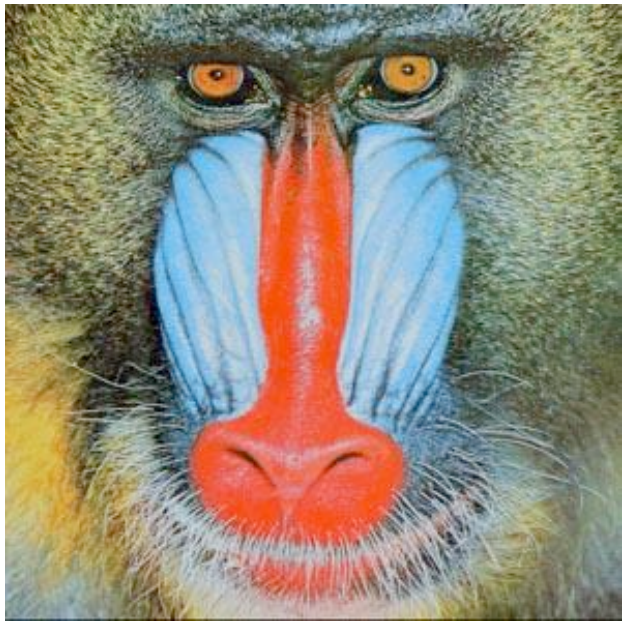


Image Processing: Scaling Filter

```
import java.awt.Color;

public class Scale
{
   public static void main(String args[])
   {
      String filename = args[0];
      int w = Integer.parseInt(args[1]);
      int h = Integer.parseInt(args[2]);
      Picture source = new Picture(filename);
      Picture target = new Picture(w, h);
      for (int ti = 0; ti < w; ti++)
         for (int tj = 0; tj < h; tj++)
         {
            int si = ti * source.width()  / w;
            int sj = tj * source.height() / h;
            Color color = source.get(si, sj);
            target.set(ti, tj, color);
         }
     source.show();
     target.show();
  }
}
```

Image Processing: Scaling Filter

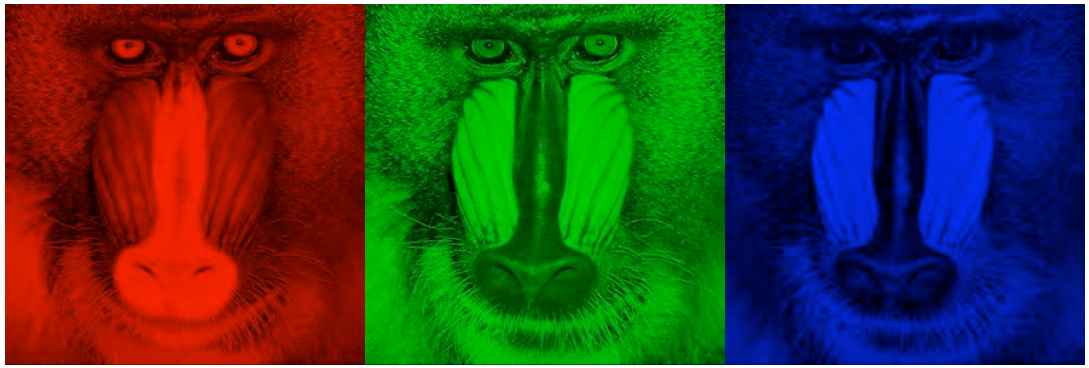
Scaling filter. Creates two `Picture` objects and two windows.



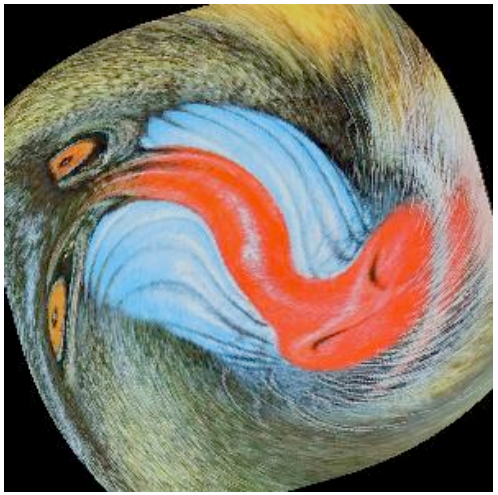
mandrill.jpg

```
% java Scale 400 200 mandrill.jpg
```

More Image Processing Effects



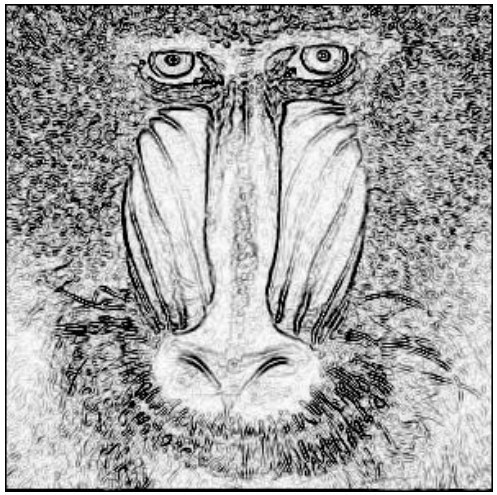
RGB color separation



swirl filter

wave filter

glass filter



Sobel edge detection

String Processing

String Data Type

String data type. Basis for text processing.

Set of values. Sequence of Unicode characters.

API.

public class String (Java string data type)

	String(String s)	*create a string with the same value as* s
int	length()	*string length*
char	charAt(int i)	*i th character*
String	substring(int i, int j)	*i th through (j-1)st characters*
boolean	contains(String sub)	*does string contain* sub *as a substring?*
boolean	startsWith(String pre)	*does string start with* pre*?*
boolean	endsWith(String post)	*does string end with* post*?*
int	indexOf(String p)	*index of first occurrence of* p
int	indexOf(String p, int i)	*index of first occurrence of* p *after* i
String	concat(String t)	*this string with* t *appended*
int	compareTo(String t)	*string comparison*
String	replaceAll(String a, String b)	*result of changing* a *to* b*s*
String[]	split(String delim)	*strings between occurrences of* delim
boolean	equals(String t)	*is this string's value the same as* t*'s?*

http://java.sun.com/javase/6/docs/api/java/lang/String.html

Typical String Processing Code

is the string a palindrome?

```
public static boolean isPalindrome(String s)
{
   int N = s.length();
   for (int i = 0; i < N/2; i++)
      if (s.charAt(i) != s.charAt(N-1-i))
         return false;
   return true;
}
```

extract file name and extension from a command-line argument

```
String s = args[0];
int dot = s.indexOf(".");
String base      = s.substring(0, dot);
String extension = s.substring(dot + 1, s.length());
```

print all lines in standard input that contain a string specified on the command line

```
String query = args[0];
while (!StdIn.isEmpty())
{
   String s = StdIn.readLine();
   if (s.contains(query)) StdOut.println(s);
}
```

print all the hyperlinks (to educational institutions) in the text file on standard input

```
while (!StdIn.isEmpty())
{
   String s = StdIn.readString();
   if (s.startsWith("http://") && s.endsWith(".edu"))
      StdOut.println(s);
}
```

Gene Finding

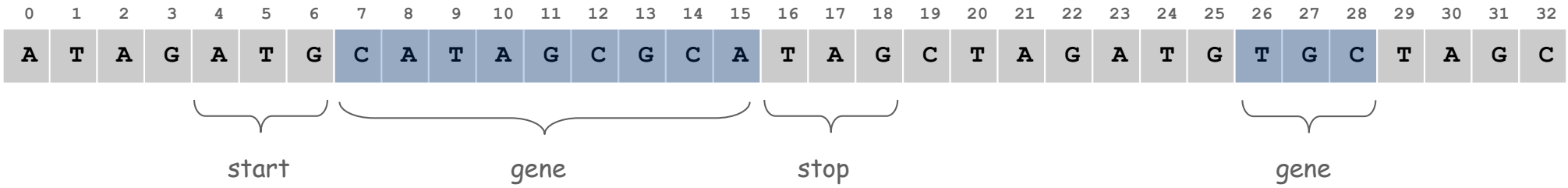
Pre-genomics era. Sequence a human genome.

Post-genomics era. Analyze the data and understand structure.

Genomics. Represent genome as a string over { A, C, T, G } alphabet.

Gene. A substring of genome that represents a functional unit.

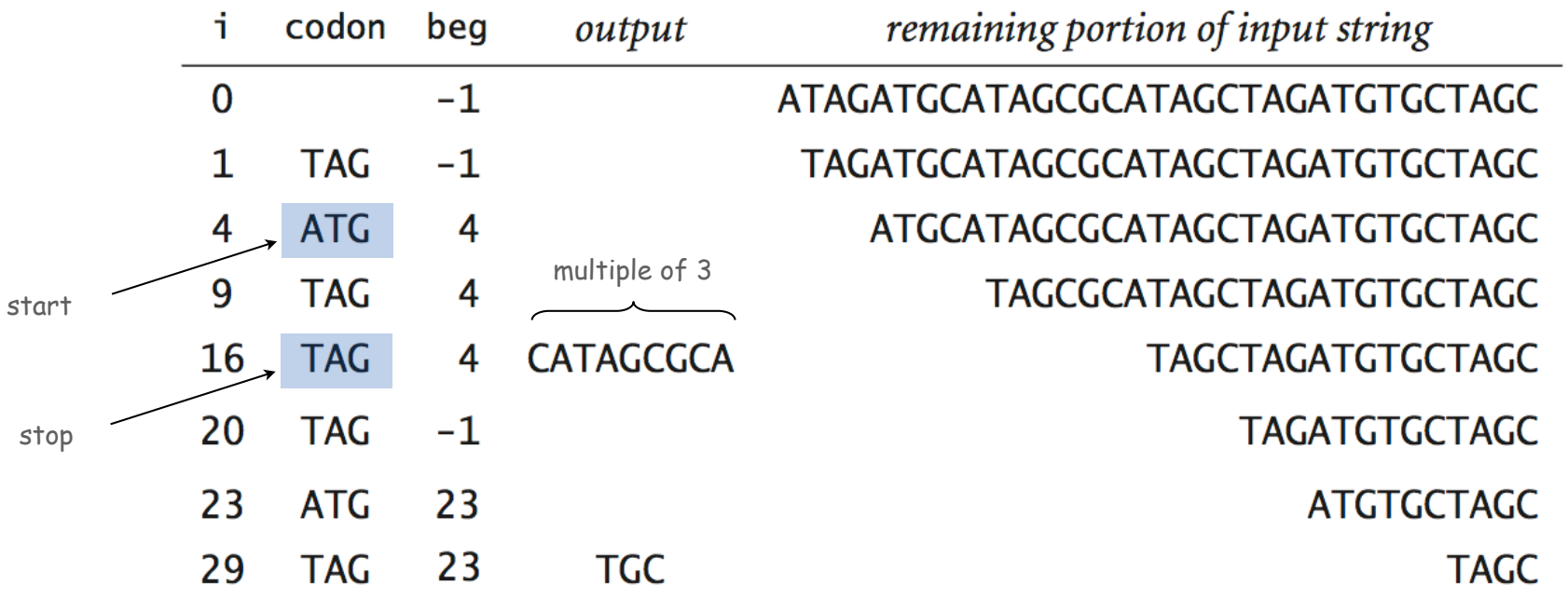
- Preceded by `ATG`. [start codon]
- Multiple of 3 nucleotides. [codons other than start/stop]
- Succeeded by `TAG`, `TAA`, or `TGA`. [stop codons]



Gene Finding: Algorithm

Algorithm. Scan left-to-right through genome.

- If start codon, then set `beg` to index `i`.
- If stop codon and substring is a multiple of 3
  - output gene
  - reset `beg` to -1



i	codon	beg	*output*	*remaining portion of input string*
0		-1		ATAGATGCATAGCGCATAGCTAGATGTGCTAGC
1	TAG	-1		TAGATGCATAGCGCATAGCTAGATGTGCTAGC
4	ATG	4		ATGCATAGCGCATAGCTAGATGTGCTAGC
9	TAG	4		TAGCGCATAGCTAGATGTGCTAGC
16	TAG	4	CATAGCGCA	TAGCTAGATGTGCTAGC
20	TAG	-1		TAGATGTGCTAGC
23	ATG	23		ATGTGCTAGC
29	TAG	23	TGC	TAGC

Gene Finding: Implementation

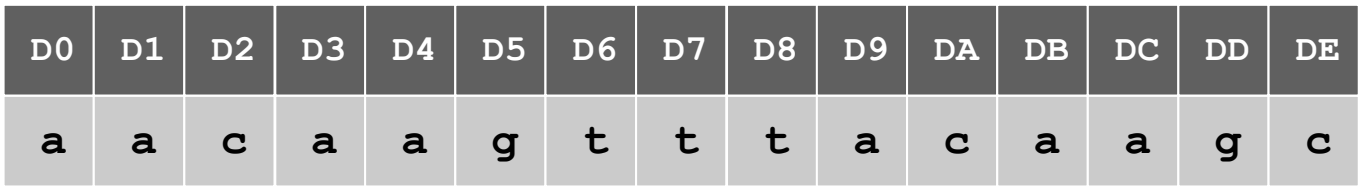
```
public class GeneFind
{
   public static void main(String[] args)
   {
      String start  = args[0];
      String stop   = args[1];
      String genome = StdIn.readAll();

      int beg = -1;
      for (int i = 0; i < genome.length() - 2; i++)
      {
         String codon = genome.substring(i, i+3);
         if (codon.equals(start)) beg = i;
         if (codon.equals(stop) && beg != -1)
         {
            String gene = genome.substring(beg+3, i);
            if (gene.length() % 3 == 0)
            {
               StdOut.println(gene);
               beg = -1;
            }
         }
      }
   }
}
```

```
% more genomeTiny.txt
ATAGATGCATAGCGCATAGCTAGATGTGCTAGC


% java GeneFind ATG TAG < genomeTiny.txt
CATAGCGCA
TGC
```

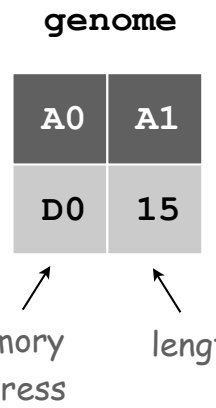
OOP Context for Strings

Possible memory representation of a string (using TOY addresses).

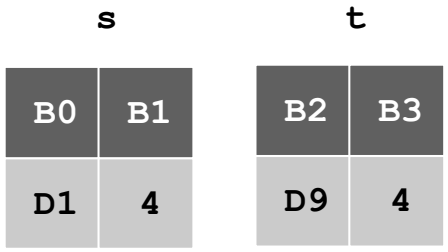
- `genome = "aacaagtttacaagc";`



D0	D1	D2	D3	D4	D5	D6	D7	D8	D9	DA	DB	DC	DD	DE
a	a	c	a	a	g	t	t	t	a	c	a	a	g	c



- `s = genome.substring(1, 5);`
- `t = genome.substring(9, 13);`



s and t are different strings that share the same value `"acaa"`

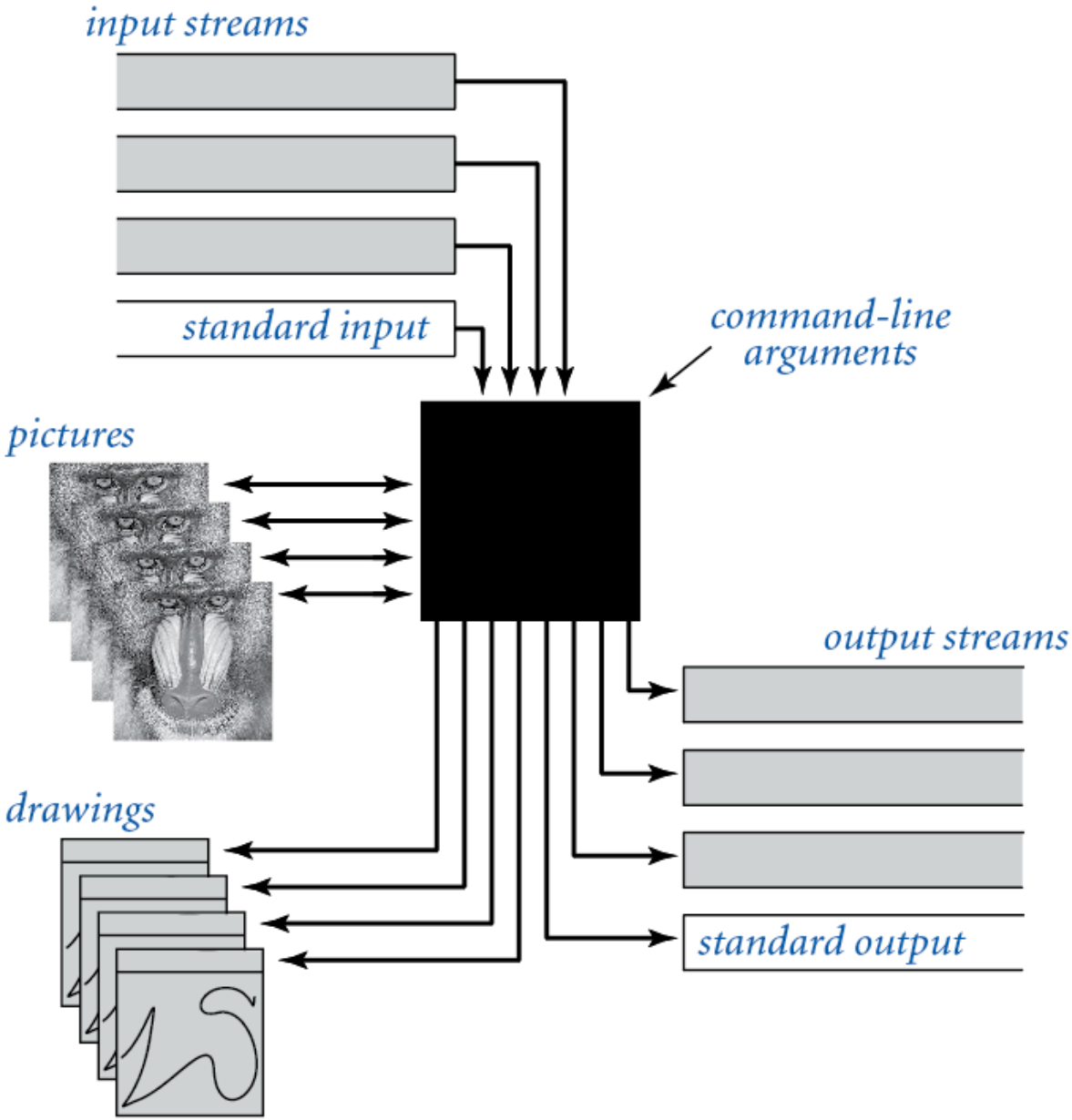
- `(s == t)` is `false`, but `(s.equals(t))` is `true`.

compares pointers

compares character sequences

In and Out

Bird's Eye View (Revisited)



Non-Standard Input

Standard input. Read from terminal window.

or use OS to redirect from one file

Goal. Read from several different input streams.

`In` data type. Read text from stdin, a file, a web site, or network.

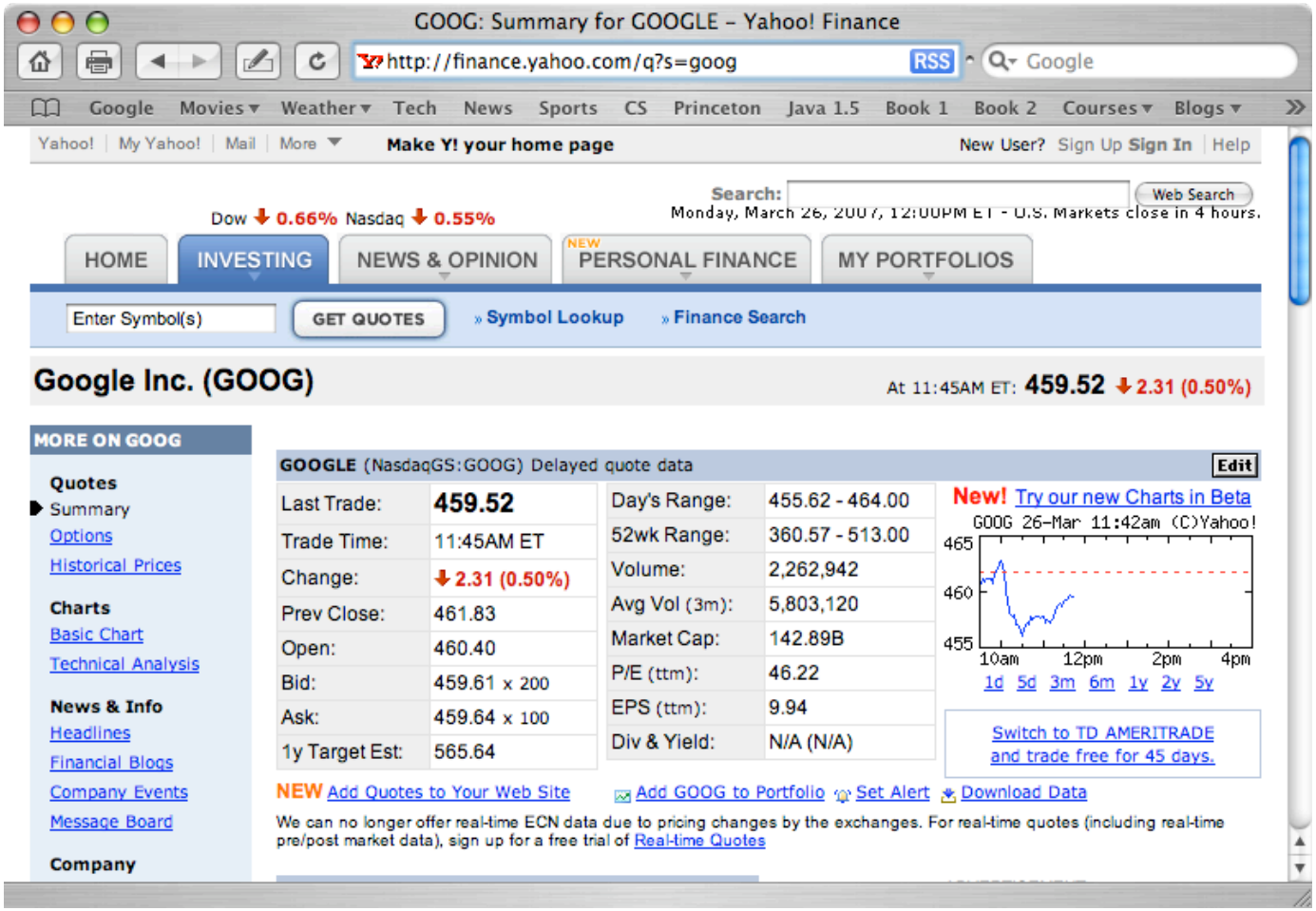
Ex: Are two text files identical?

```
public class Diff
{
   public static void main(String[] args)
   {
      In in0 = new In(args[0]);
      In in1 = new In(args[1]);
      String s = in0.readAll();
      String t = in1.readAll();
      StdOut.println(s.equals(t));
   }
}
```

Screen Scraping

Goal. Find current stock price of Google.

Step 1. Find web source.



`http://finance.yahoo.com/q?s=goog`

NYSE symbol

Screen Scraping

Goal. Find current stock price of Google.

Step 2. Find string representation (HTML code) of web source.

```
…
<tr>
<td class="yfnc_tablehead1" width="48%">
Last Trade:
</td>
<td class="yfnc_tabledata1">
<big>
<b>459.52</b>
</big>
</td>
</tr>
<tr>
<td class="yfnc_tablehead1" width="48%">
Trade Time:
</td>
<td class="yfnc_tabledata1">
11:45AM ET
</td>
</tr>
…
```

price is string between and after "Last Trade"

Screen Scraping

Goal. Find current stock price of Google.

Step 3. Write code to extract stock price from HTML code.

```
public class StockQuote
{
   public static void main(String[] args)
   {
      String name = "http://finance.yahoo.com/q?s=";
      In in = new In(name + args[0]);
      String input = in.readAll();
      int start    = input.indexOf("Last Trade:", 0);
      int from     = input.indexOf("<b>",  start);
      int to       = input.indexOf("</b>", from);
      String price = input.substring(from + 3, to);
      StdOut.println(price);
   }
}
```

price is string between and after "Last Trade"

```
% java StockQuote goog
459.52
```

- `s.indexOf(t, i)`: index of first occurrence of `t` in `s`, starting at offset `i`.
- Read raw html from `http://finance.yahoo.com/q?s=goog`.
- Find first string delimited by `<b>` and `</b>` after `Last Trade`.

Day Trader

Add bells and whistles.

- Plot price in real-time.
- Notify user if price dips below a certain price.
- Embed logic to determine when to buy and sell.
- Automatically send buy and sell orders to trading firm.

Warning. Use at your own financial risk.

The New Yorker, September 6, 1999

OOP Summary

Object. Holds a data type value; variable name refers to object.

In Java, programs manipulate references to objects.

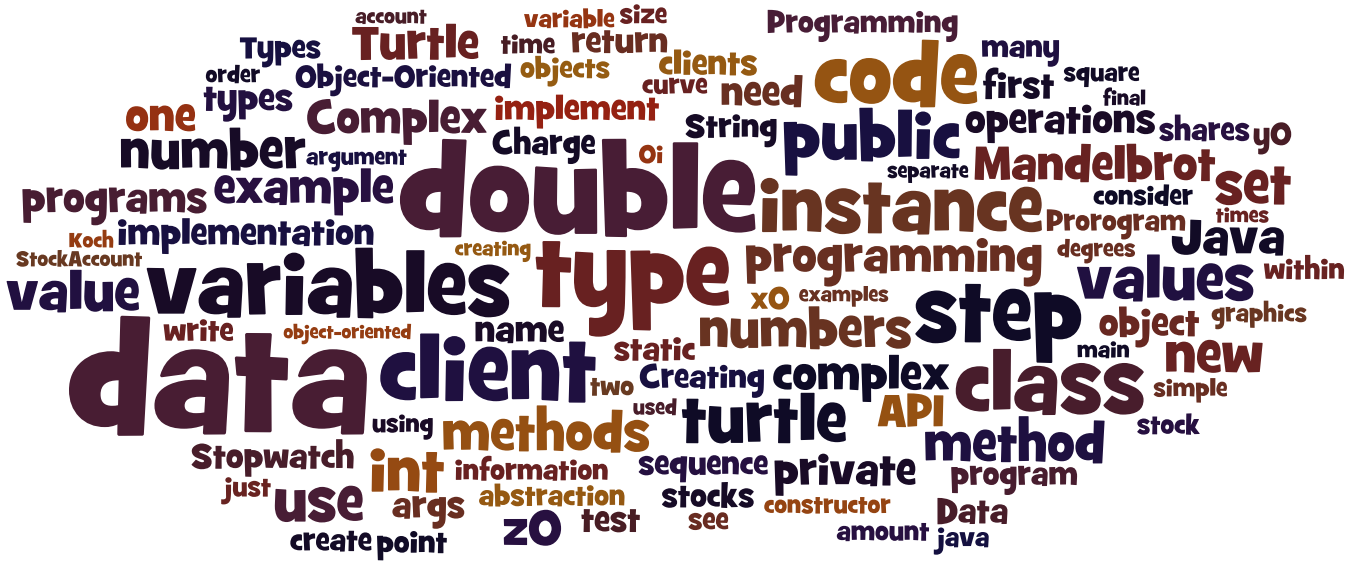
- Exception: primitive types, e.g., `boolean`, `int`, `double`.
- Reference types: `String`, `Picture`, `Color`, arrays, everything else.
- OOP purist: language should not have separate primitive types.

Bottom line.

You learned to write programs that manipulate colors, pictures, strings, and I/O streams.

Next time.

You will learn to define your own abstractions and to write programs that manipulate them.


account
variable
size
Programming
Types
Turtle
time
return
many
order
Object-Oriented
objects
clients
code
square
curve
need
first
final
types
Complex
implement
one
String
public
operations
shares
yo
number
argument
Charge
oi
separate
Mandelbrot
set
programs
example
double
instance
consider
Prorogram
times
Koch
implementation
creating
Java
StockAccount
degrees
value
variables
type
programming
within
xo
examples
values
step
write
object-oriented
name
numbers
object
graphics
data
client
static
main
new
two
Creating
complex
class
simple
used
turtle
API
using
methods
stock
method
Stopwatch
int
information
sequence
private
program
just
use
args
abstraction
stocks
constructor
Data
create
point
zo
test
see
amount
java

3.2 Creating Data Types

Data Types

Data type. Set of values and operations on those values.

Basic types.

Data Type	Set of Values	Some Operations
`boolean`	`true, false`	not, and, or, xor
`int`	-2^{31} to $2^{31} - 1$	add, subtract, multiply
`String`	sequence of Unicode characters	concatenate, compare

Last time. Write programs that use data types.

Today. Write programs to create our own data types.

Defining Data Types in Java

To define a data type, define:

- Set of values.
- Operations defined on them.

Java class. Allows us to define data types by specifying:

- Instance variables. (set of values)
- Methods. (operations defined on them)
- Constructors. (create and initialize new objects)

Point Charge Data Type

Goal. Create a data type to manipulate point charges.

Set of values. Three real numbers. [position and electrical charge]

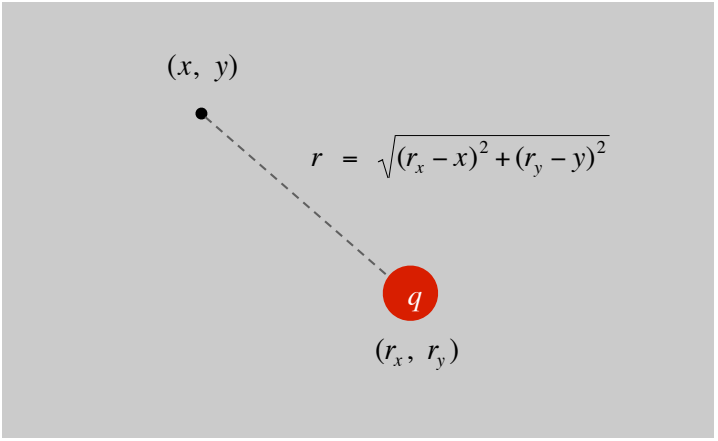
Operations.

- Create a new point charge at (r_x, r_y) with electric charge q.
- Determine electric potential V at (x, y) due to point charge.
- Convert to `String`.

$$V = k \frac{q}{r}$$

r = distance between (x, y) and (r_x, r_y)

k = electrostatic constant = $8.99 \times 10^9 \ \mathrm{N \cdot m^2 / C^2}$



Point Charge Data Type

Goal. Create a data type to manipulate point charges.

Set of values. Three real numbers. [position and electrical charge]

API.

public class Charge		
	Charge(double x0, double y0, double q0)	
double	potentialAt(double x, double y)	*electric potential at (x, y) due to charge*
String	toString()	*string representation*

Charge Data Type: A Simple Client

Client program. Uses data type operations to calculate something.

```
public static void main(String[] args)
{
   double x = Double.parseDouble(args[0]);
   double y = Double.parseDouble(args[1]);
   Charge c1 = new Charge(.51, .63, 21.3);
   Charge c2 = new Charge(.13, .94, 81.9);
   double v1 = c1.potentialAt(x, y);
   double v2 = c2.potentialAt(x, y);
   StdOut.println(c1);
   StdOut.println(c2);
   StdOut.println(v1 + v2);
}
```

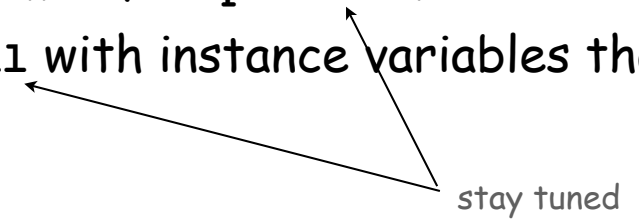
automagically invokes the `toString()` method

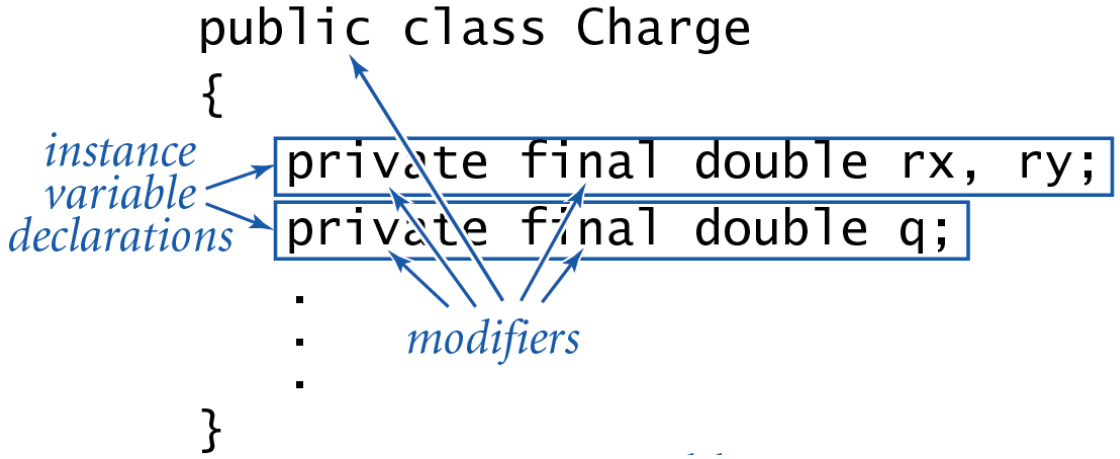
```
% java Charge .50 .50
21.3 at (0.51, 0.63)
81.9 at (0.13, 0.94)
2.749369070859l2e12
```

Anatomy of Instance Variables

Instance variables. Specifies the set of values.

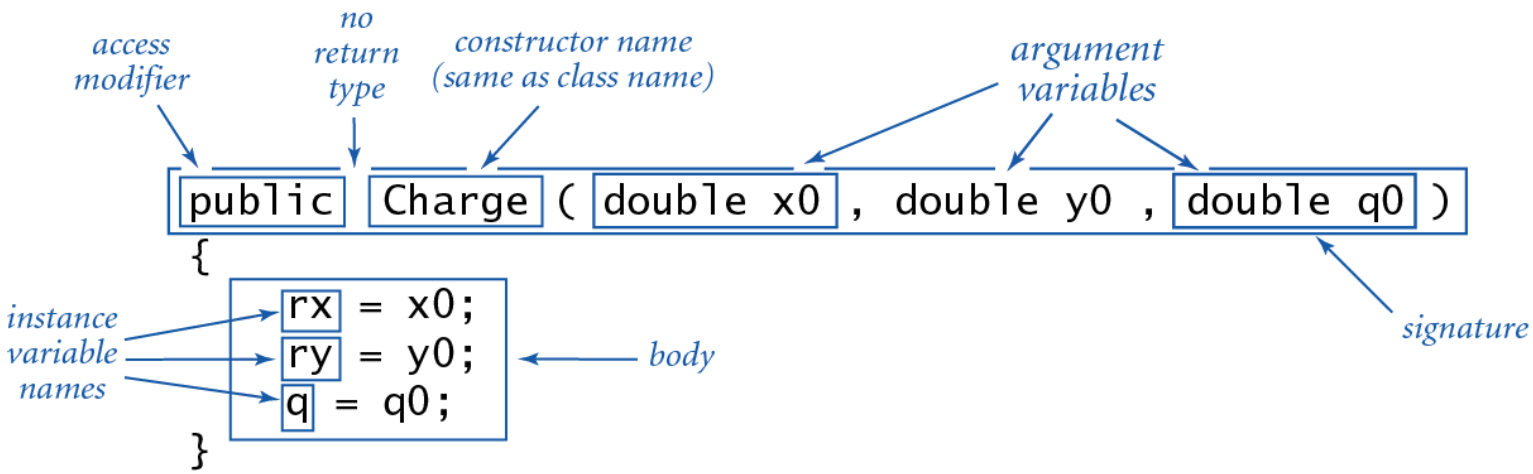
- Declare outside any method.
- Always use access modifier `private`.
- Use modifier `final` with instance variables that never change.





Anatomy of a Constructor

Constructor. Specifies what happens when you create a new object.



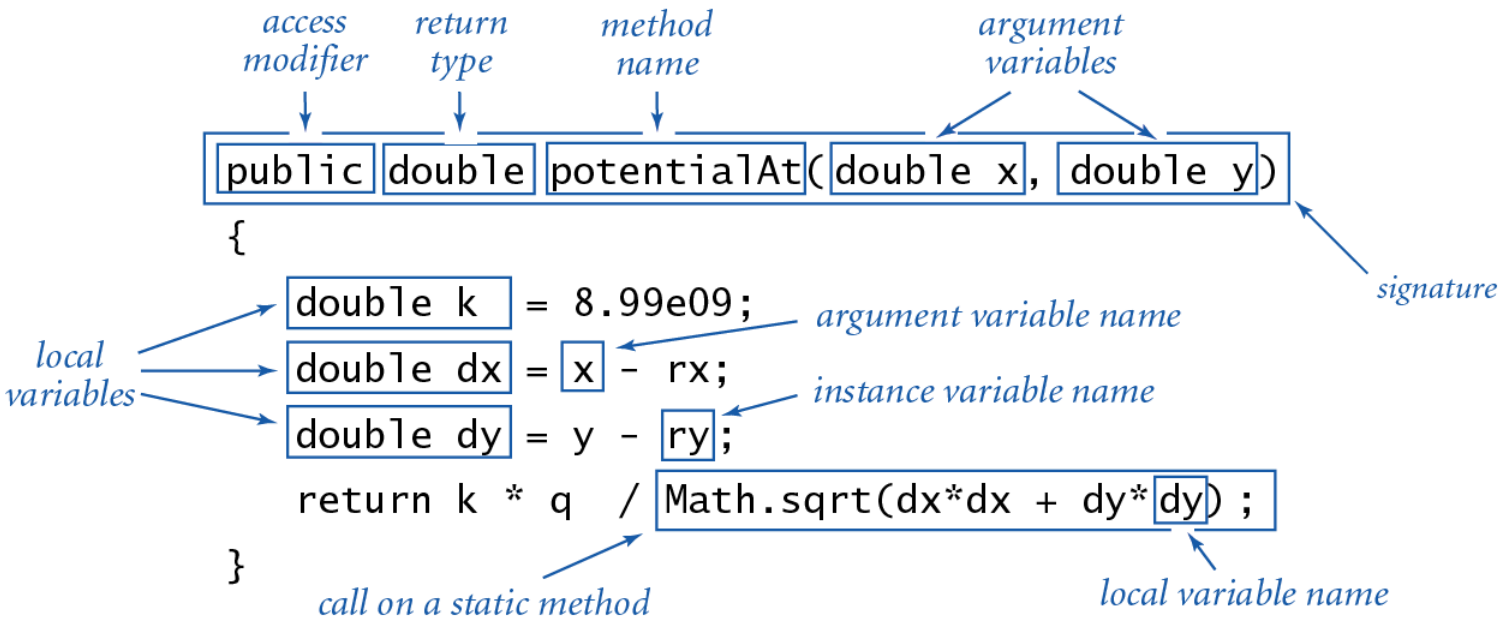
Invoking a constructor. Use `new` operator to create a new object.

```
Charge c1 = new Charge(.51, .63, 21.3);
Charge c2 = new Charge(.13, .94, 81.9);
```

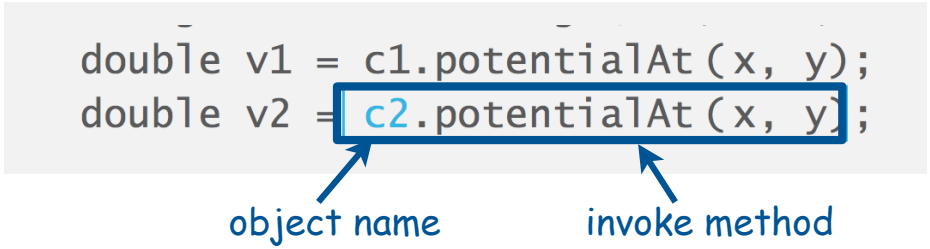
invoke constructor

Anatomy of a Data Type Method

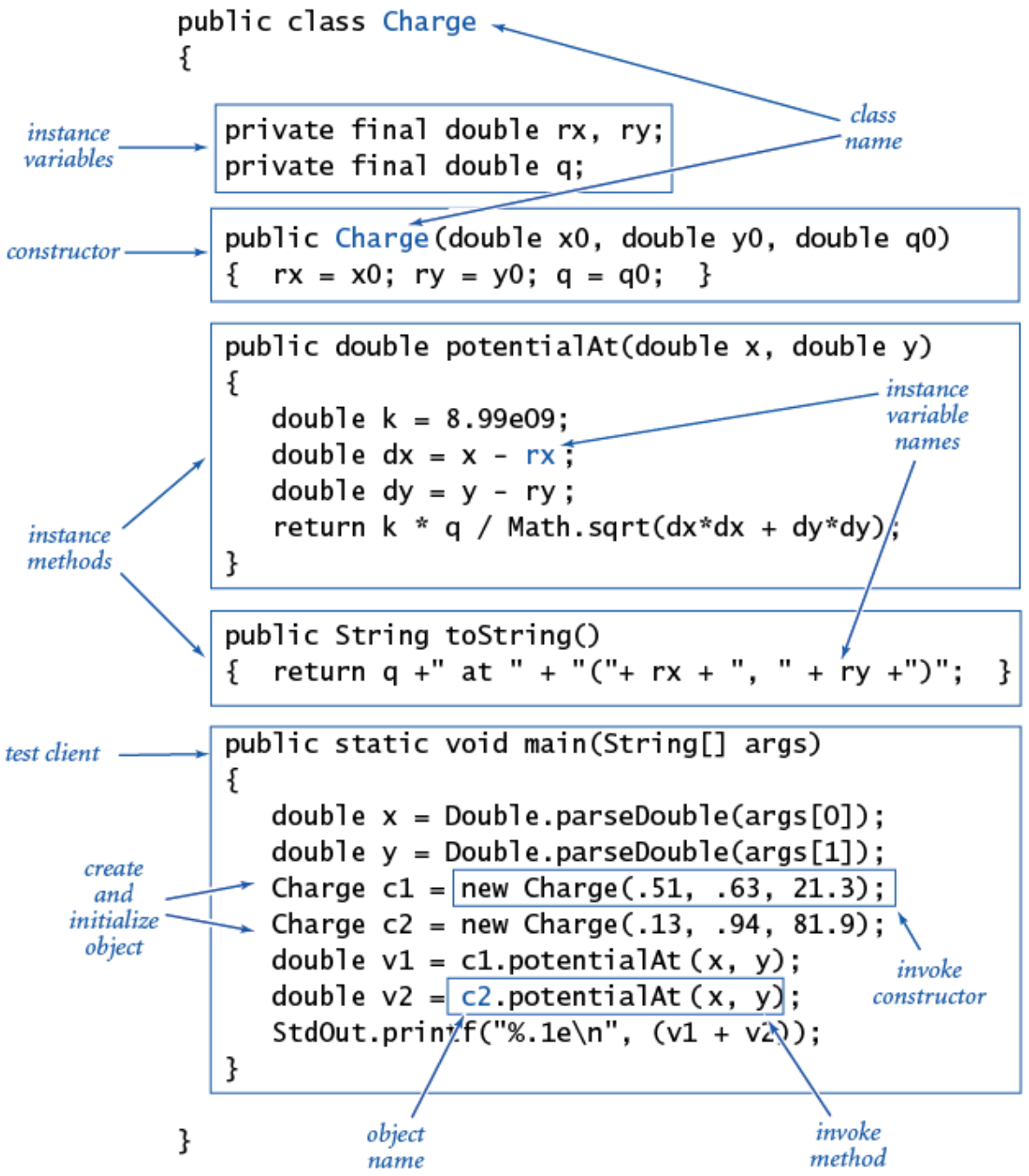
Method. Define operations on instance variables.



Invoking a method. Use dot operator to invoke a method.



Anatomy of a Class

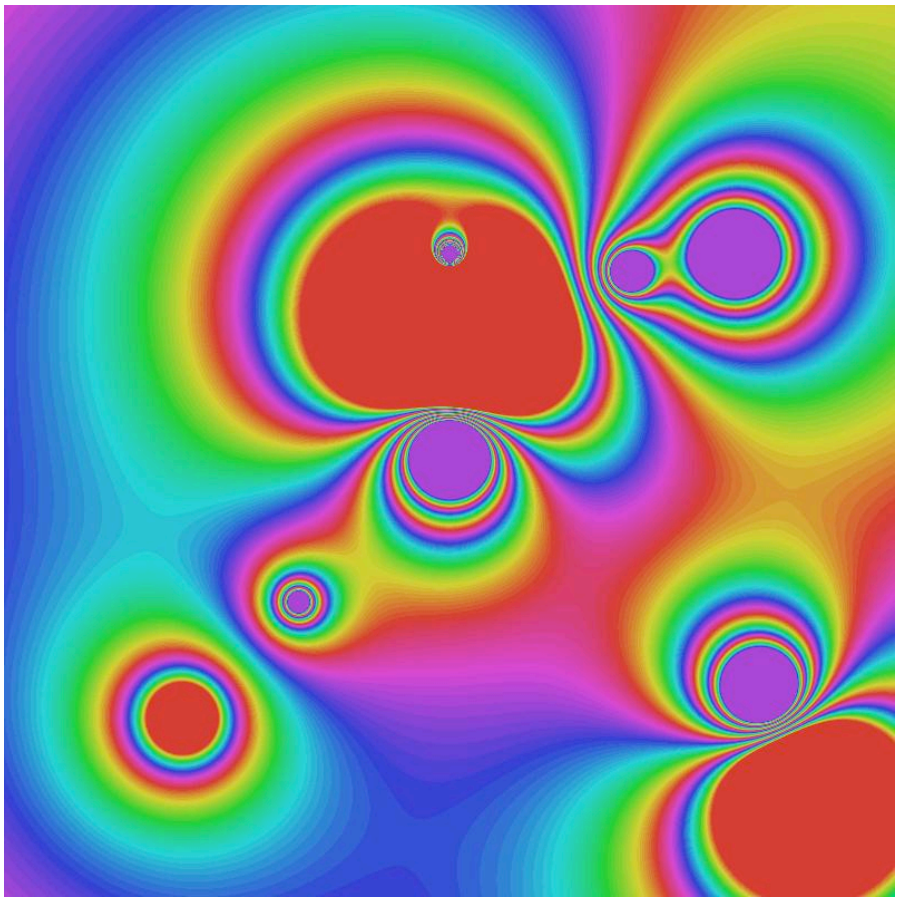


Charge Client Example: Potential Visualization

Potential visualization. Read in N point charges from a file; compute total potential at each point in unit square.

```
% more charges.txt
.51 .63 -100
.50 .50   40
.50 .72   10
.33 .33    5
.20 .20  -10
.70 .70   10
.82 .72   20
.85 .23   30
.90 .12  -50
```

```
% java Potential < charges.txt
```



Potential Visualization

Arrays of objects. Allocate memory for the array; then allocate memory for each individual object.

```
// Read in the data.
int N = StdIn.readInt();
Charge[] a = new Charge[N];
for (int i = 0; i < N; i++)
{
   double x0 = StdIn.readDouble();
   double y0 = StdIn.readDouble();
   double q0 = StdIn.readDouble();
   a[i] = new Charge(x0, y0, q0);
}
```

Potential Visualization

```
// Plot the data.
int SIZE = 512;
Picture pic = new Picture(SIZE, SIZE);
for (int row = 0; row < SIZE; row++)
   for (int col = 0; col < SIZE; col++)
   {
      double V = 0.0;
      for (int i = 0; i < N; i++)
      {
         double x = 1.0 * row / SIZE;
         double y = 1.0 * col / SIZE;
         V += a[i].potentialAt(x, y);
      }
      Color color = getColor(V);  // Arbitrary double-Color map.
      pic.set(row, SIZE-1-col, color);
   }
pic.show();
```

$$V = \sum_i \left(k \, q_i \, / \, r_i \right)$$

(0, 0) is upper left

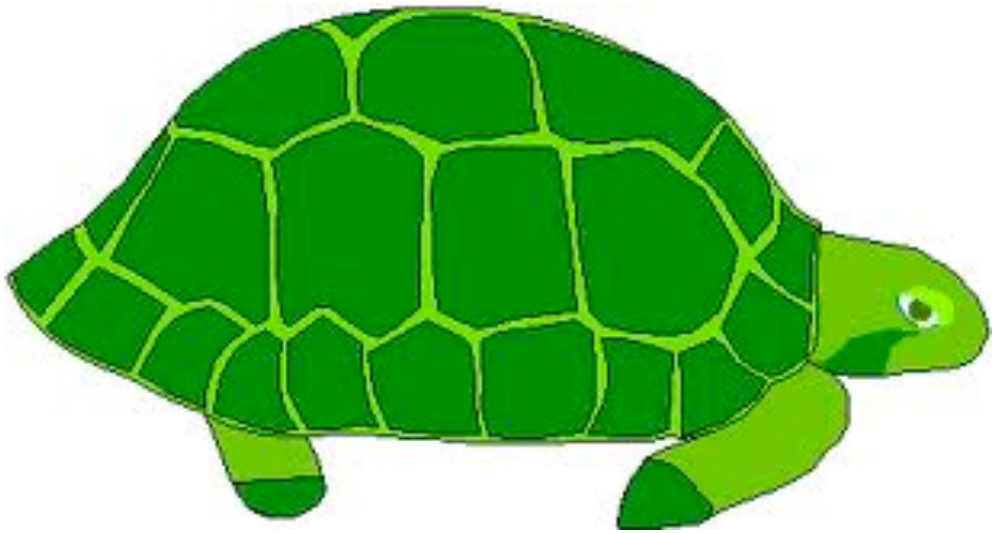
TEQ on Data Types

[easy if you read Exercise 3.2.5]

Fix the serious bug in the following code.

```
public class Charge
{
   private double rx, ry;
   private double q;
   public Charge double x0, double y0, double q0)
   {
      double rx = x0;
      double ry = y0;
      double q = q0;
   }
}
```

Turtle Graphics



Turtle Graphics

Goal. Create a data type to manipulate a turtle moving in the plane.

Set of values. Location and orientation of turtle.

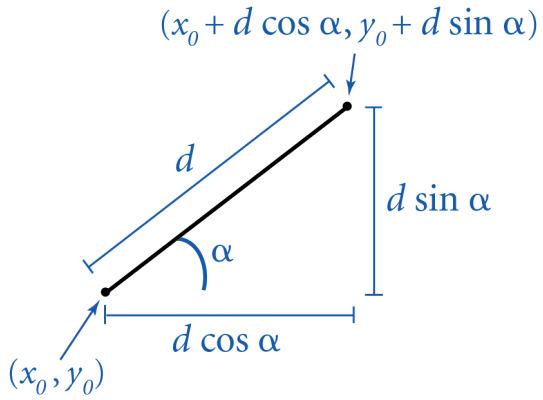
API.

public class Turtle	
Turtle(double x0, double y0, double a0)	*create a new turtle at* (x_0, y_0) *facing* a_0 *degrees counterclockwise from the x-axis*
void turnLeft(double delta)	*rotate* `delta` *degrees counterclockwise*
void goForward(double step)	*move distance* `step`*, drawing a line*

```
// Draw a square.
Turtle turtle = new Turtle(0.0, 0.0, 0.0);
turtle.goForward(1.0);          turtle.turnLeft
(90.0);          turtle.goForward(1.0);
turtle.turnLeft(90.0); turtle.goForward(1.0);
turtle.turnLeft(90.0);
turtle.goForward(1.0);
turtle.turnLeft(90.0);
```

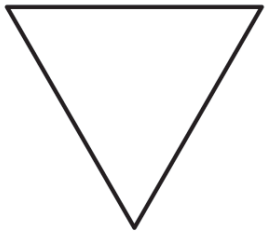
Turtle Graphics Implementation

```
public class Turtle
{
   private double x, y;   // turtle is at (x, y)
   private double angle;  // facing this direction

   public Turtle(double x0, double y0, double a0)
   {
      x = x0;
      y = y0;
      angle = a0;
   }

   public void turnLeft(double delta)
   {
      angle += delta;
   }

   public void goForward(double d)
   {
      double oldx = x;
      double oldy = y;
      x += d * Math.cos(Math.toRadians(angle));
      y += d * Math.sin(Math.toRadians(angle));
      StdDraw.line(oldx, oldy, x, y);
   }
}
```

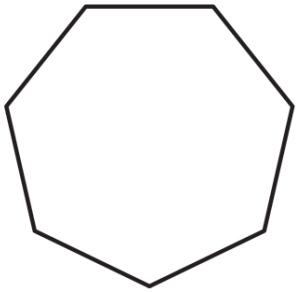


Turtle client example: N-gon

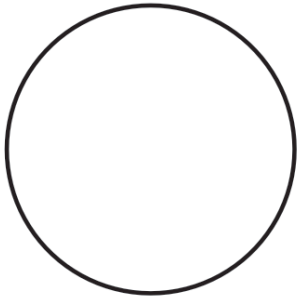
```
public class Ngon
{
   public static void main(String[] args)
   {
      int N        = Integer.parseInt(args[0]);
      double angle = 360.0 / N;
      double step  = Math.sin(Math.toRadians(angle/2.0));
      Turtle turtle = new Turtle(0.5, 0, angle/2.0);
      for (int i = 0; i < N; i++)
      {
         turtle.goForward(step);
         turtle.turnLeft(angle);
      }
   }
}
```



3



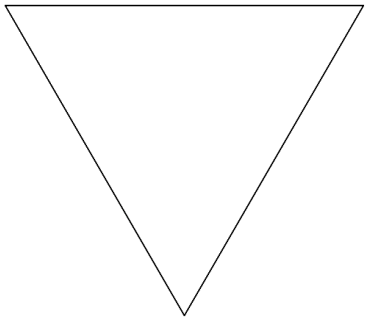
7



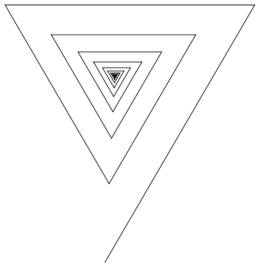
1440

Turtle client example: Spira Mirabilis

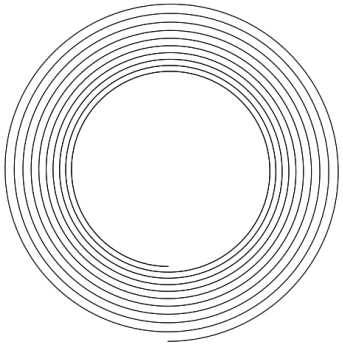
```
public class Spiral
{
   public static void main(String[] args)
   {
      int N        = Integer.parseInt(args[0]);
      double decay = Double.parseDouble(args[1]);
      double angle = 360.0 / N;
      double step  = Math.sin(Math.toRadians(angle/2.0));
      Turtle turtle = new Turtle(0.5, 0, angle/2.0);
      for (int i = 0; i < 10 * N; i++)
      {
         step /= decay;
         turtle.goForward(step);
         turtle.turnLeft(angle);
      }
   }
}
```



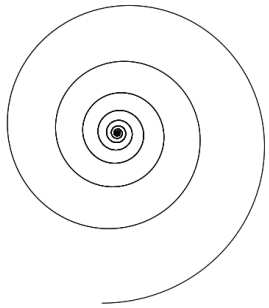
3 1.0



3 1.2

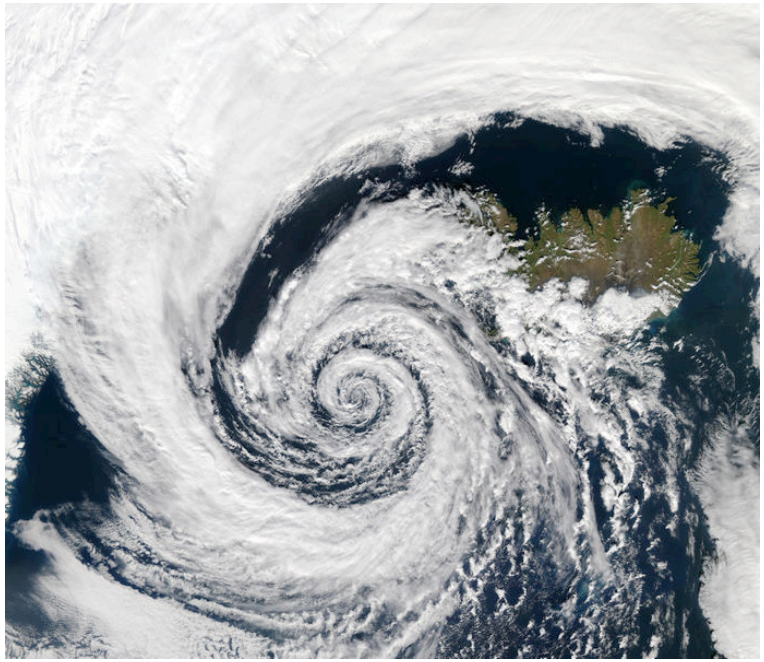


1440 1.00004

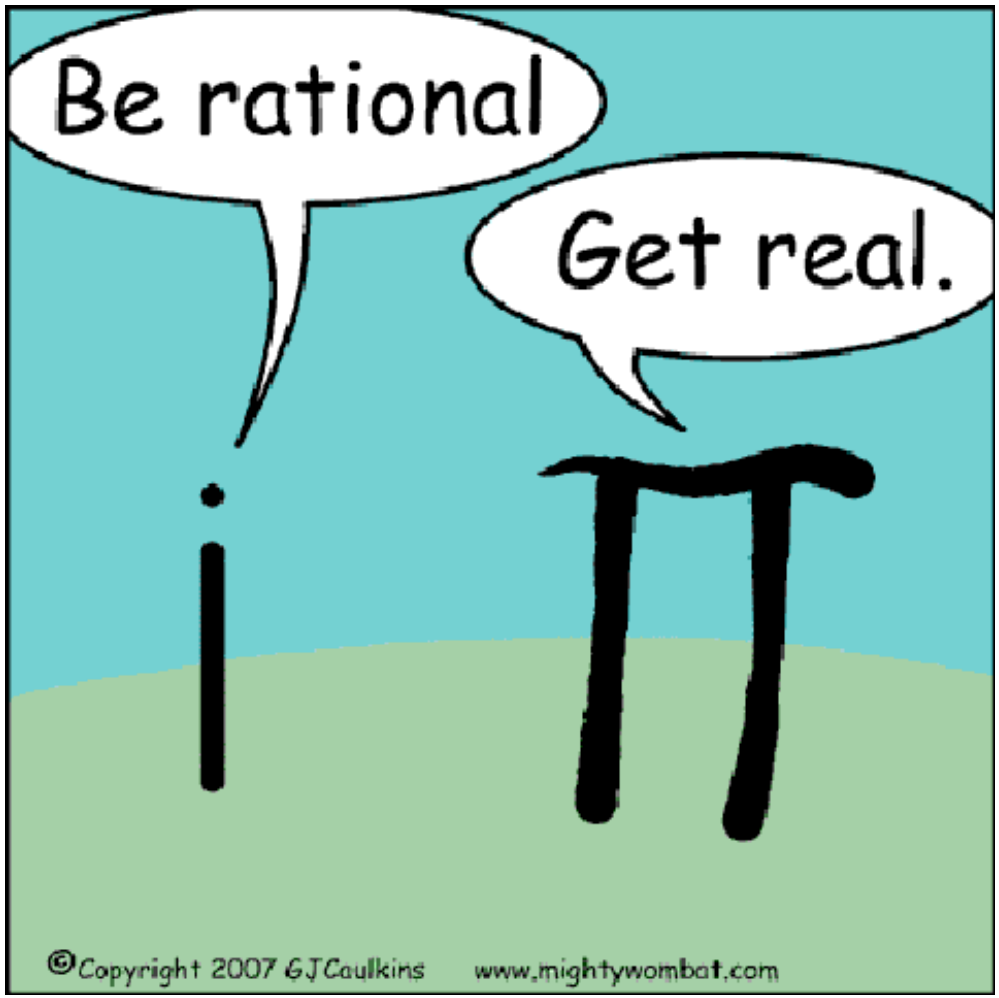


1440 1.0004

Spira Mirabilis in Nature



Complex Numbers



Complex Number Data Type

Goal. Create a data type to manipulate complex numbers.

Set of values. Two real numbers: real and imaginary parts.

API.

public class Complex	
Complex(double real, double imag)	
Complex plus(Complex b)	*sum of this number and* b
Complex times(Complex b)	*product of this number and* b
double abs()	*magnitude*
String toString()	*string representation*

$a = 3 + 4i,\ b = -2 + 3i$

$a + b = 1 + 7i$

$a \times b = -18 + i$

$|\, a \,| = 5$

Applications of Complex Numbers

Relevance. A quintessential mathematical abstraction.

Applications.

- Fractals.
- Impedance in RLC circuits.
- Signal processing and Fourier analysis.
- Control theory and Laplace transforms.
- Quantum mechanics and Hilbert spaces.
- …

Complex Number Data Type: A Simple Client

Client program. Uses data type operations to calculate something.

```
public static void main(String[] args)
{
   Complex a = new Complex( 3.0, 4.0);
   Complex b = new Complex(-2.0, 3.0);
   Complex c = a.times(b);
   StdOut.println("a = " + a);
   StdOut.println("b = " + b);
   StdOut.println("c = " + c);
}
```

result of `c.toString()`

```
% java TestClient
a = 3.0 + 4.0i
b = -2.0 + 3.0i
c = -18.0 + 1.0i
```

Remark. Can't write `a = b*c` since no operator overloading in Java.

Complex Number Data Type: Implementation

```
public class Complex
{

   private final double re;
   private final double im;

   public Complex(double real, double imag)
   {
      re = real;
      im = imag;
   }

   public String toString()
   {  return re + " + " + im + "i";  }

   public double abs()
   { return Math.sqrt(re*re + im*im); }

   public Complex plus(Complex b)
   {
      double real = re + b.re;
      double imag = im + b.im;
      return new Complex(real, imag);
   }

   public Complex times(Complex b)
   {
      double real = re * b.re - im * b.im;
      double imag = re * b.im + im * b.re;
      return new Complex(real, imag);
   }

}
```

instance variables

constructor

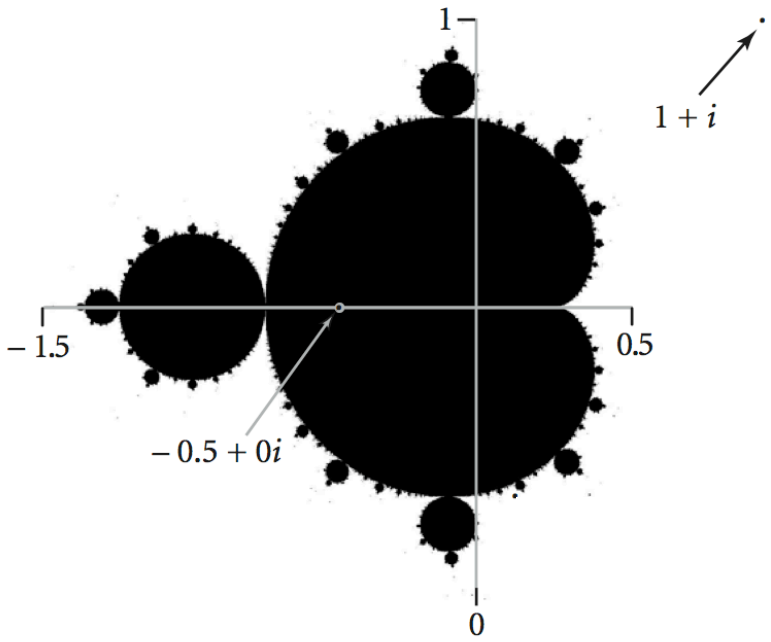
methods

refers to b's instance variable

Mandelbrot Set

Mandelbrot set. A set of complex numbers.

Plot. Plot (x, y) black if $z = x + y\,i$ is in the set, and white otherwise.

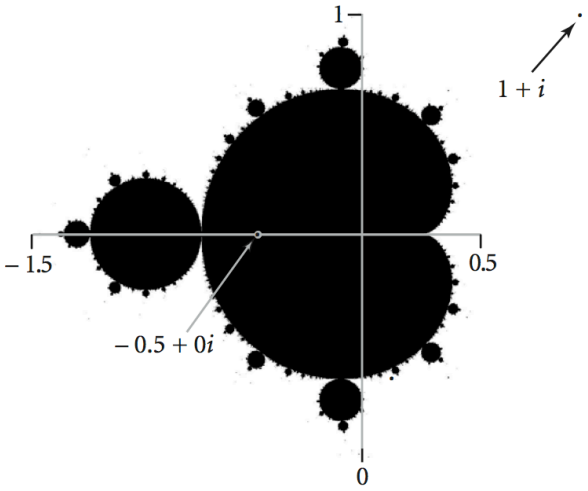


- No simple formula describes which complex numbers are in set.
- Instead, describe using an algorithm.

Mandelbrot Set

Mandelbrot set. Is complex number z_0 is in set?

- Iterate $z_{t+1} = (z_t)^2 + z_0$.
- If $|z_t|$ diverges to infinity, then z_0 not in set; otherwise z_0 is in set.



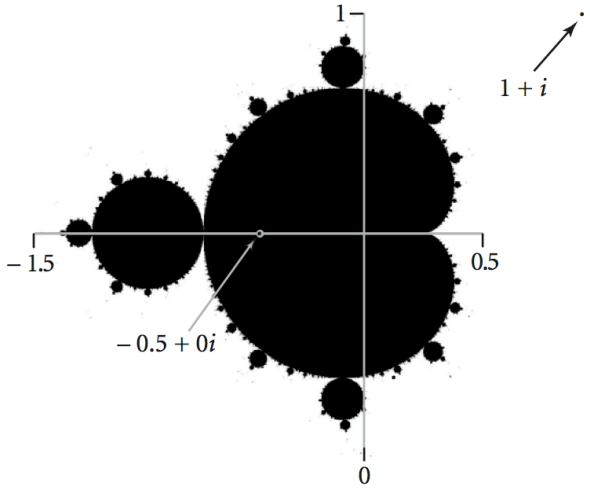
t	z_t
0	-1/2 + 0i
1	-1/4 + 0i
2	-7/16 + 0i
3	-79/256 + 0i
4	-26527/65536 + 0i
5	-1443801919/4294967296 + 0i

z = -1/2 is in Mandelbrot set

t	z_t
0	1 + i
1	1 + 3i
2	-7 + 7i
3	1 - 97i
4	-9407 – 193i
5	88454401 + 3631103i

z = 1 + i not in Mandelbrot set

Plotting the Mandelbrot Set

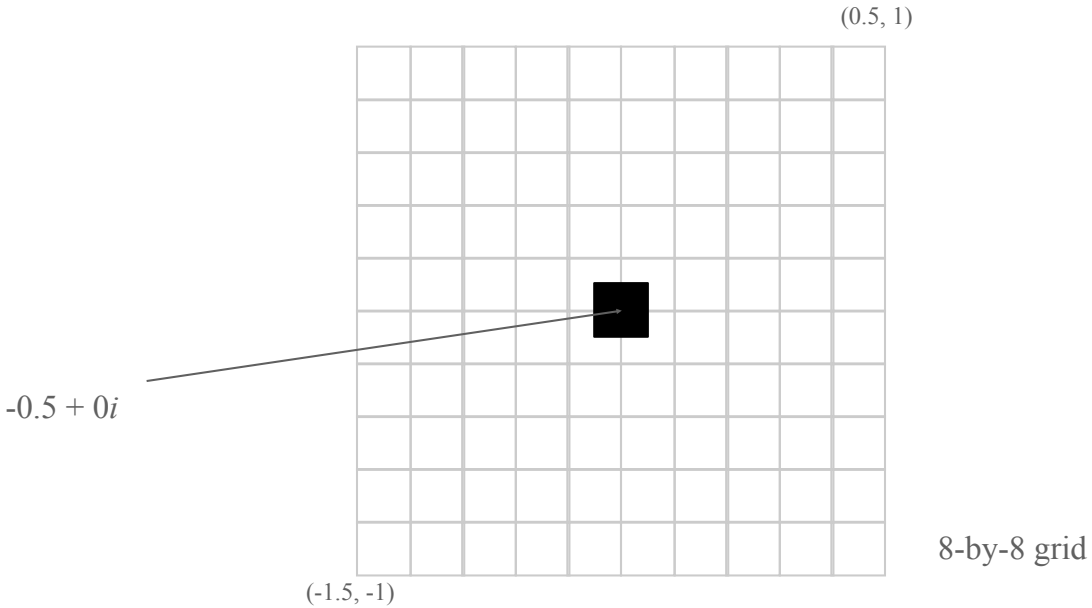


Practical issues.

- Cannot plot infinitely many points.
- Cannot iterate infinitely many times.

Approximate solution.

- Sample from an N-by-N grid of points in the plane.
- Fact: if $|z_t| > 2$ for any t, then z not in Mandelbrot set.
- Pseudo-fact: if $|z_{255}| \le 2$ then z "likely" in Mandelbrot set.



Complex Number Data Type: Another Client

Mandelbrot function with complex numbers.

- Is z in the Mandelbrot set?
- Returns white (definitely no) or black (probably yes).

```
public static Color mand(Complex z0)
{
   Complex z = z0;
   for (int t = 0; t < 255; t++)
   {
      if (z.abs() > 2.0) return Color.WHITE;
      z = z.times(z);
      z = z.plus(z0);
   }
   return Color.BLACK;
}
```

$z = z^2 + z_0$

More dramatic picture: replace `Color.WHITE` with grayscale or color.

```
new Color(255-t, 255-t, 255-t)
```

Complex Number Data Type: Another Client

Plot the Mandelbrot set in gray scale.

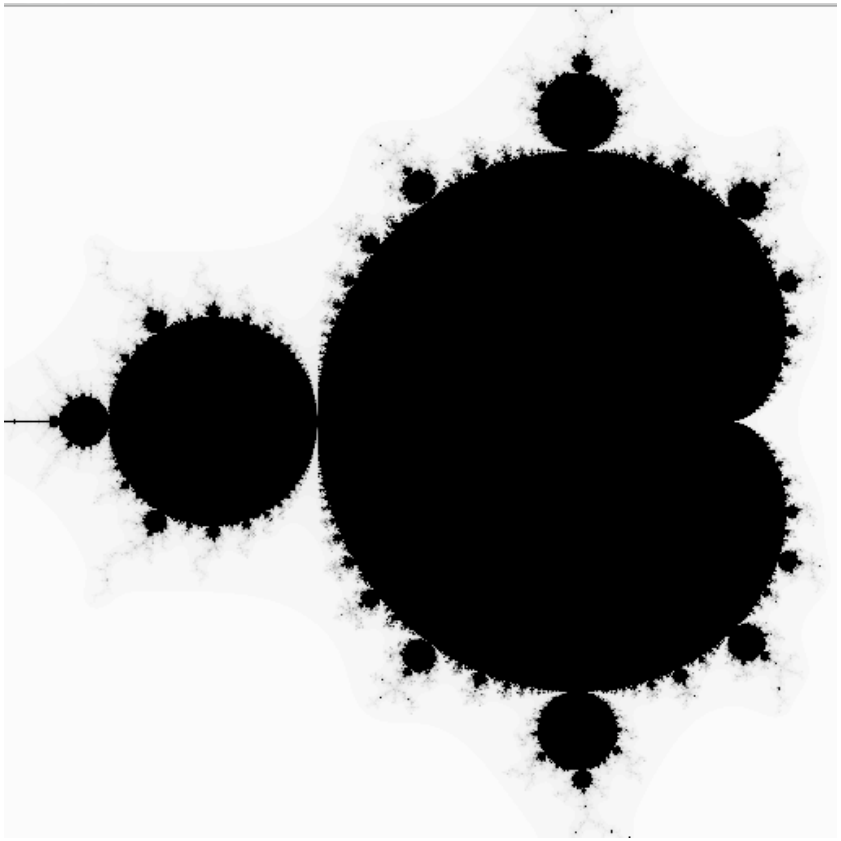
```
public static void main(String[] args)
{
   double xc   = Double.parseDouble(args[0]);
   double yc   = Double.parseDouble(args[1]);
   double size = Double.parseDouble(args[2]);
   int N = 512;
   Picture pic = new Picture(N, N);


   for (int i = 0; i < N; i++)
      for (int j = 0; j < N; j++)
      {
         double x0 = xc - size/2 + size*i/N;
         double y0 = yc - size/2 + size*j/N;
         Complex z0 = new Complex(x0, y0);
         Color color = mand(z0);
         pic.set(i, N-1-j, color);
      }
   pic.show();
}
```

scale to screen coordinates

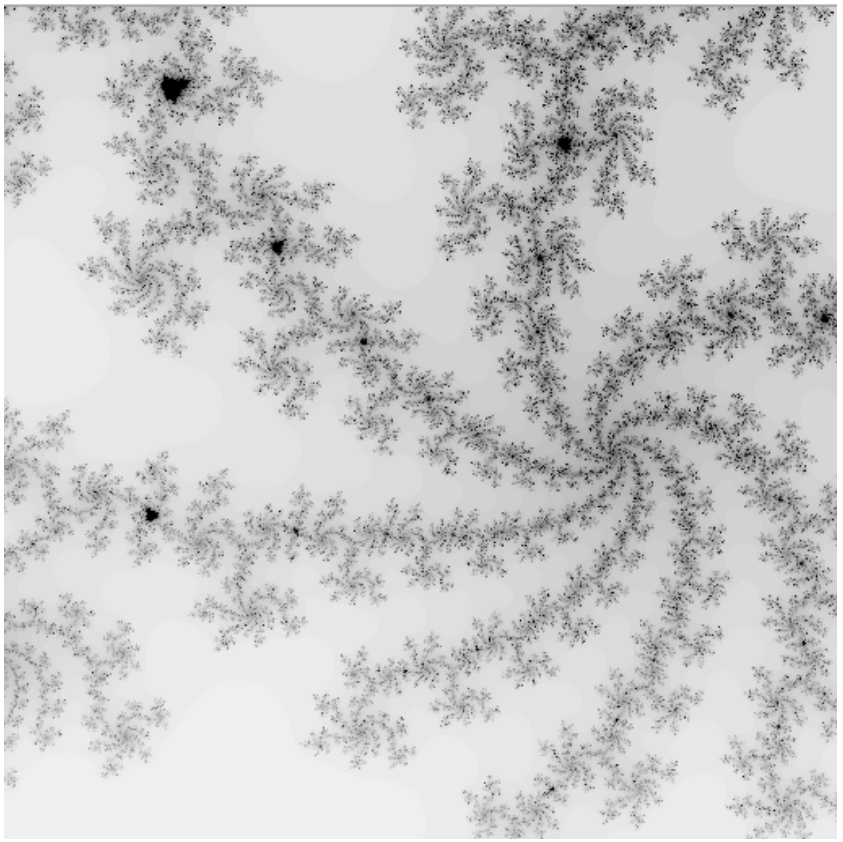
(0, 0) is upper left

Mandelbrot Set

% **java Mandelbrot -.5 0 2**

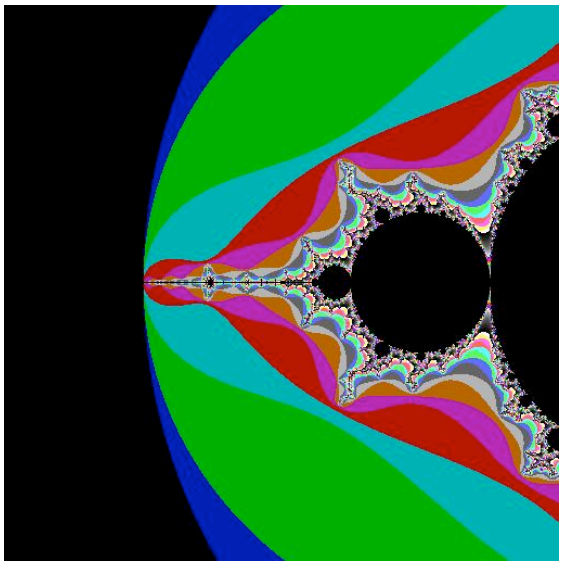


% **java Mandelbrot .1045 -.637 .01**

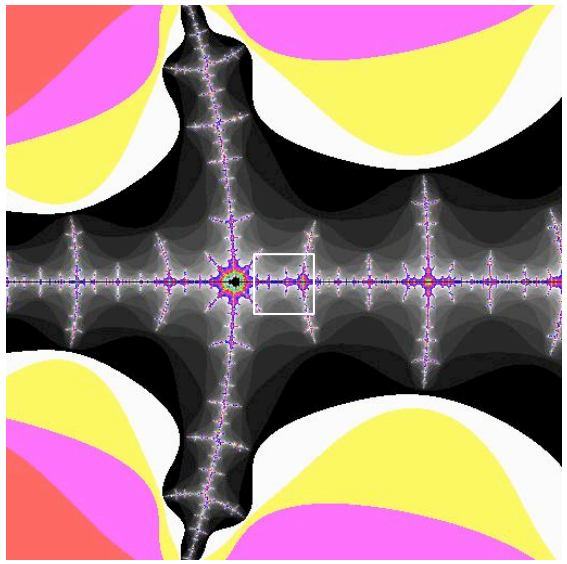


Mandelbrot Set

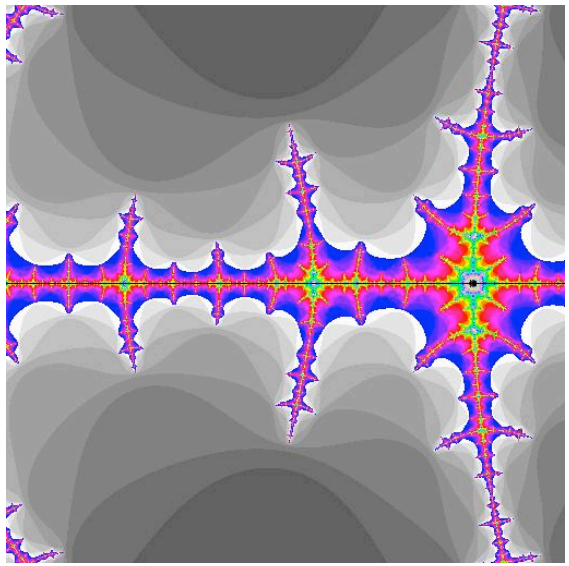
```
% java ColorMandelbrot -1.5 0 2 < mandel.txt
```

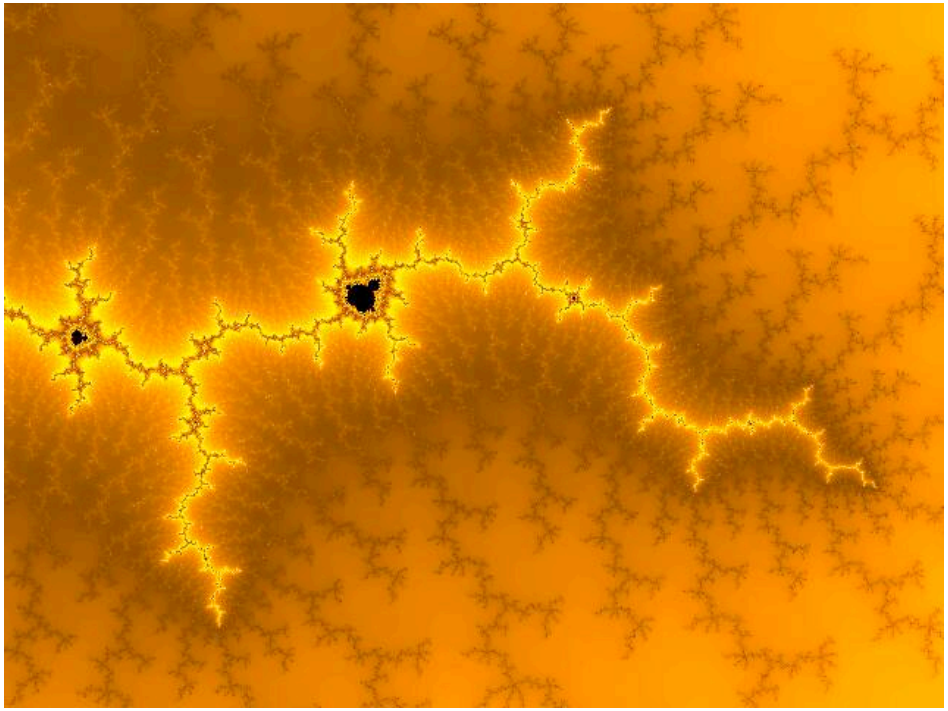


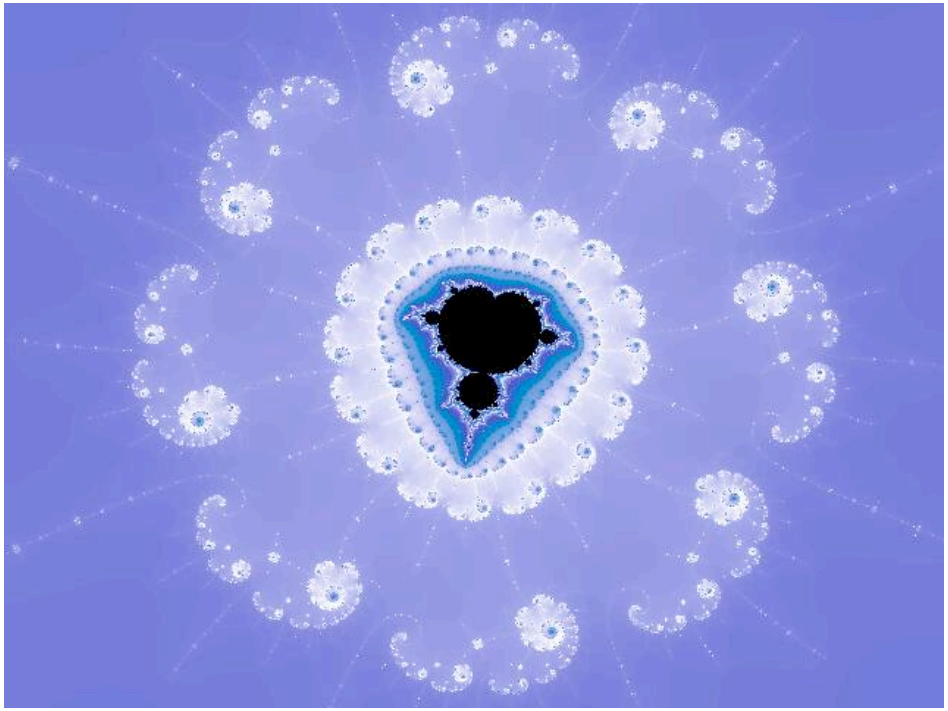
```
-1.5 0 .02
```

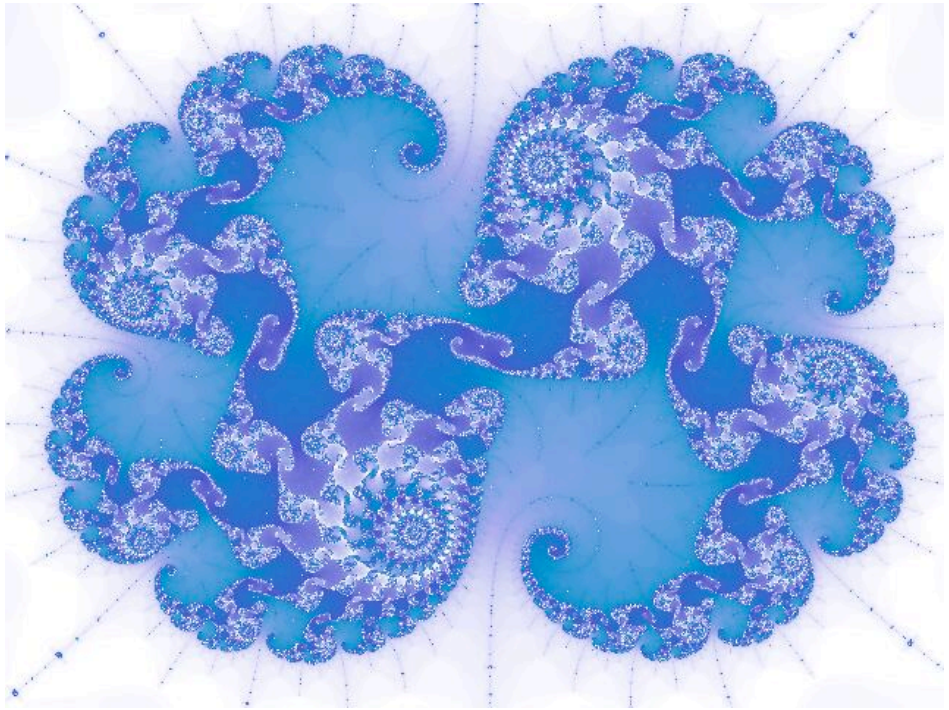


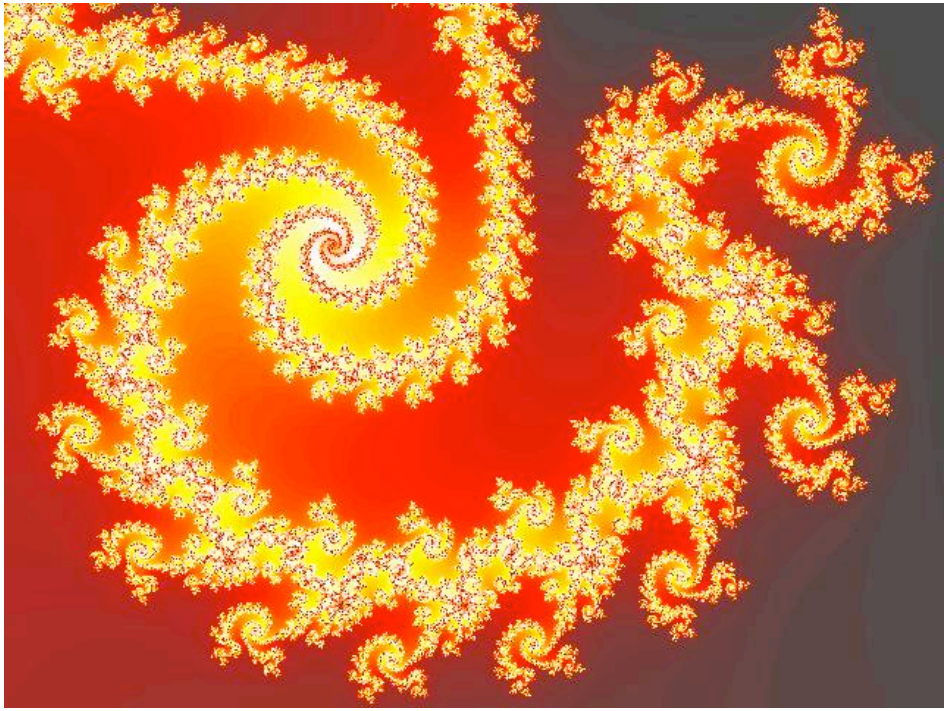
```
-1.5 0 .002
```

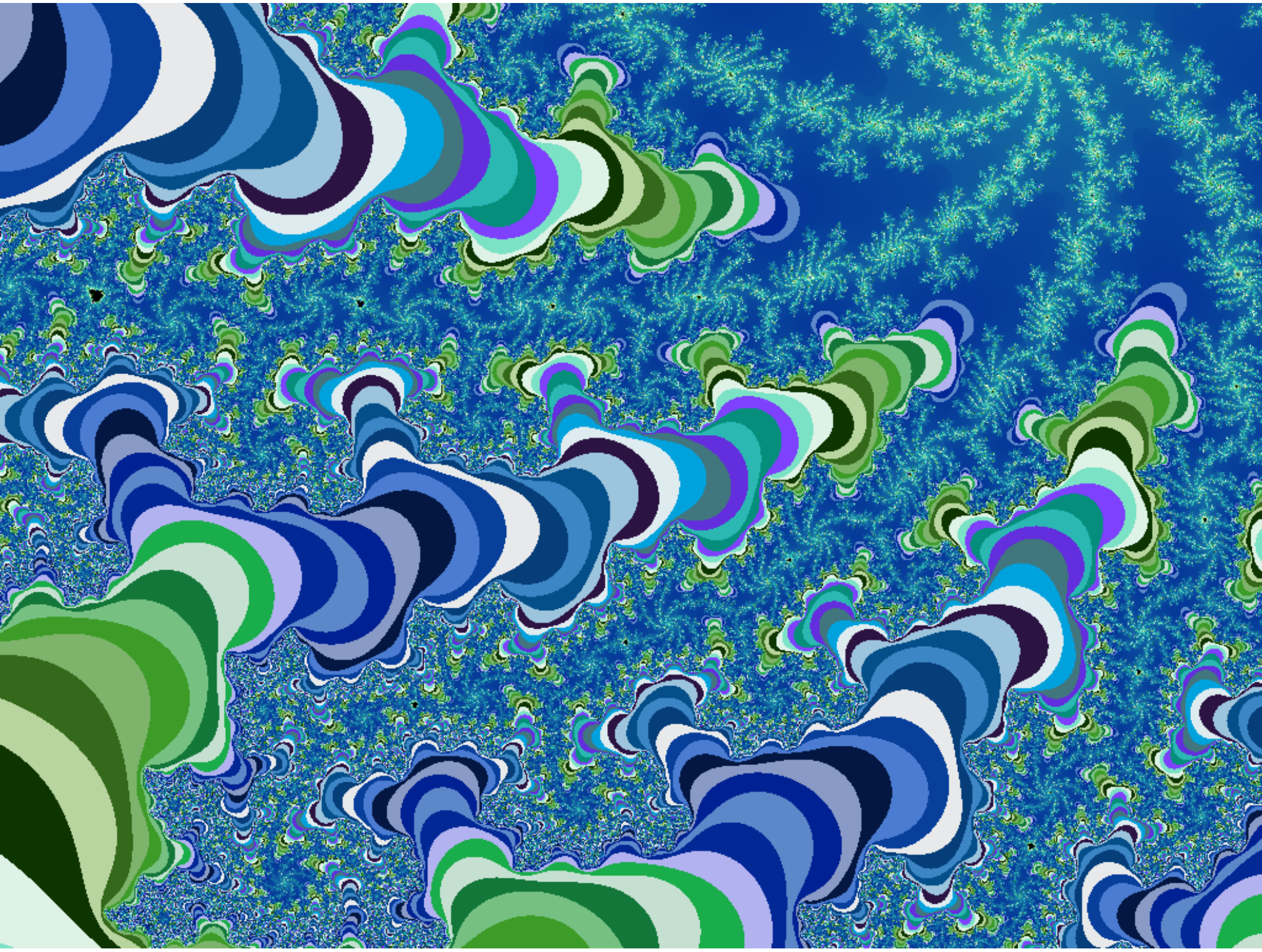












Applications of Data Types

Data type. Set of values and collection of operations on those values.

Simulating the physical world.

- Java objects model real-world objects.
- Not always easy to make model reflect reality.
- Ex: charged particle, molecule, COS 126 student, ...

Extending the Java language.

- Java doesn't have a data type for every possible application.
- Data types enable us to add our own abstractions.
- Ex: complex, vector, polynomial, matrix, ...