Arrays and Functions

Dr. 何明昕, He Mingxin, Max

program06 @ yeah.net

Email Subject：(AE|A2|A3) + *(Last 4 digits of ID) + Name*: *TOPIC*

Sakai: CS102A in 2018A

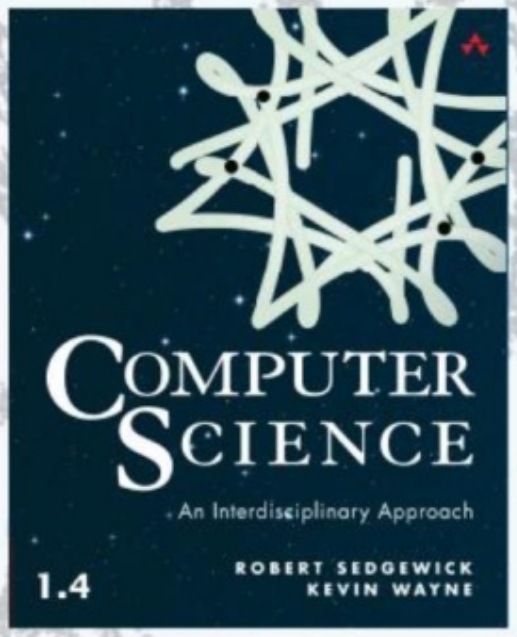
计算机程序设计基础

Introduction to Computer Programming

COMPUTER SCIENCE

SEDGEWICK / WAYNE

PART I: PROGRAMMING IN JAVA



http://introcs.cs.princeton.edu

Arrays

COMPUTER SCIENCE

SEDGEWICK / WAYNE

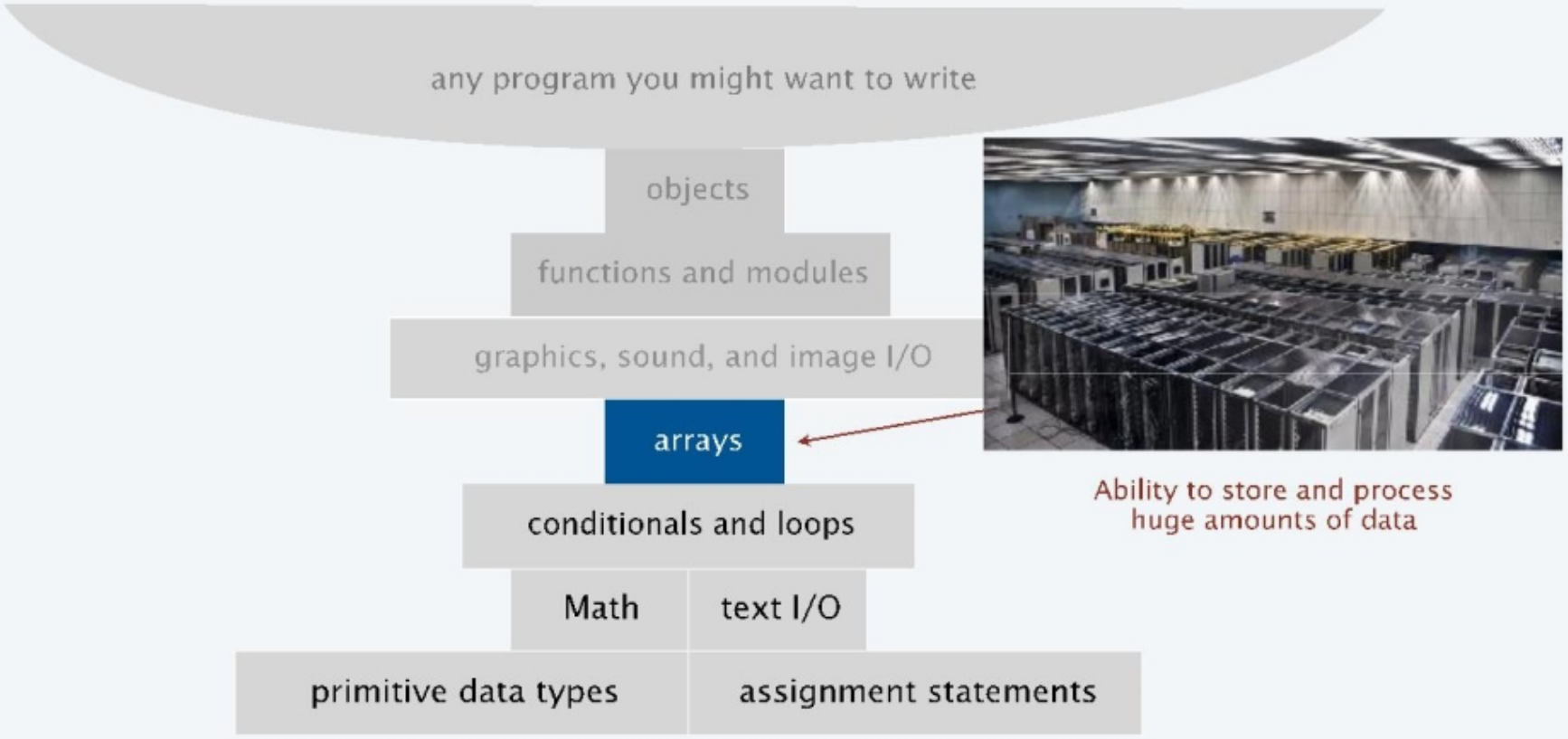
PART I: PROGRAMMING IN JAVA

Arrays

- Basic concepts
- Typical array-processing code

CS.3.A.Arrays.Basics

Basic building blocks for programming



Your first data structure

A data structure is an arrangement of data that enables efficient processing by a program.

An array is an *indexed* sequence of values of the same type.

Examples.

- 52 playing cards in a deck.
- 100 thousand students in an online class.
- 1 billion pixels in a digital image.
- 4 billion nucleotides in a DNA strand.
- 73 billion Google queries per year.
- 86 billion neurons in the brain.
- 50 trillion cells in the human body.
- 6.02×10^{23} particles in a mole.

index	*value*
0	2♥
1	6♠
2	A♦
3	A♥
...	
49	3♣
50	K♣
51	4♠

Main purpose. Facilitate storage and manipulation of data.

Processing many values of the same type

10 values, without arrays

```
double a0 = 0.0;
double a1 = 0.0;
double a2 = 0.0;
double a3 = 0.0;
double a4 = 0.0;
double a5 = 0.0;
double a6 = 0.0;
double a7 = 0.0;
double a8 = 0.0;
double a9 = 0.0;
...
a4 = 3.0;
...
a8 = 8.0;
...
double x = a4 + a8;
```

tedious and error-prone code

10 values, with an array

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

an easy alternative

1 million values, with an array

```
double[] a;
a = new double[1000000];
...
a[234567] = 3.0;
...
a[876543] = 8.0;
...
double x = a[234567] + a[876543];
```

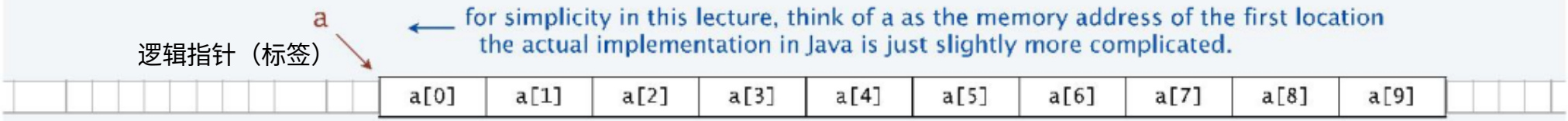
scales to handle huge amounts of data

Memory representation of an array

An array is an indexed sequence of values of the same type.

A computer's memory is *also* an indexed sequence of memory locations. ← stay tuned for many details

- Each primitive type value occupies a fixed number of locations.
- *Array values are stored in contiguous locations.*



Critical concepts

- Indices start at 0.
- Given `i`, the operation of accessing the value `a[i]` is extremely efficient.
- The assignment `b = a` makes the names `b` and `a` refer to the same array.

it does *not* copy the array, as with primitive types (stay tuned for details)

Java language support for arrays

Basic support

operation	*typical code*
Declare an array	`double[] a;`
Create an array of a given length	`a = new double[1000];`
Refer to an array entry by index	`a[i] = b[j] + c[k];`
Refer to the length of an array	`a.length;`

Initialization options

operation	*typical code*
Default initialization to 0 for numeric types	`a = new double[1000];`
Declare, create and initialize in one statement	`double[] a = new double[1000];`
Initialize to literal values	`double[] x = { 0.3, 0.6, 0.1 };`

no need to use a loop like

```
for (int i = 0; i < 1000; i++
  a[i] = 0.0;
```

BUT cost of creating an array is proportional to its length.

Copying an array

To copy an array, create a new array , then copy all the values.

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

Important note: The code b = a does *not* copy an array (it makes b and a refer to the same array).

```
double[] b = new double[a.length];
b = a;
```

Copying an array

To copy an array, create a new array , then copy all the values.

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

Important note: The code b = a does *not* copy an array (it makes b and a refer to the same array).

```
double[] b = new double[a.length];
b = a;
```

Programming with arrays: typical examples

For brevity, N is `a.length` and `b.length` in all this code.

Access command-line args in system array

```
int stake  = Integer.parseInt(args[0]);
int goal   = Integer.parseInt(args[1]);
int trials = Integer.parseInt(args[2]);
```

Copy to another array

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

Create an array with N random values

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

Print array values, one per line

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

Compute the average of array values

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

Find the maximum of array values

```
double max = a[0];
for (int i = 1; i < N; i++)
   if (a[i] > max) max = a[i];
```

Pop quiz 1 on arrays

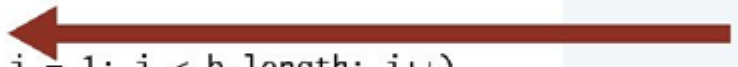
Q. What does the following code print?

```
public class PQarray1
{
    public static void main(String[] args)
    {
        int[] a = new int[6];
        int[] b = new int[a.length];

        b = a;
        for (int i = 1; i < b.length; i++)
            b[i] = i;

        for (int i = 0; i < a.length; i++)
            System.out.print(a[i] + " ");
        System.out.println();

        for (int i = 0; i < b.length; i++)
            System.out.print(b[i] + " ");
        System.out.println();
    }
}
```

Pop quiz 1 on arrays

Q. What does the following code print?

```
public class PQarray1
{
    public static void main(String[] args)
    {
        int[] a = new int[6];
        int[] b = new int[a.length];

        b = a;
        for (int i = 1; i < b.length; i++)
            b[i] = i;

        for (int i = 0; i < a.length; i++)
            System.out.print(a[i] + " ");
        System.out.println();

        for (int i = 0; i < b.length; i++)
            System.out.print(b[i] + " ");
        System.out.println();
    }
}
```



After this, b and a refer to the same array

A.

```
% java PQarray1
0 1 2 3 4 5
0 1 2 3 4 5
```

Programming with arrays: typical bugs

Array index out of bounds

```
double[] a = new double[10];
for (int i = 1; i <= 10; i++)
   a[i] = Math.random();
```

No a[10] (and a[0] unused)

Uninitialized array

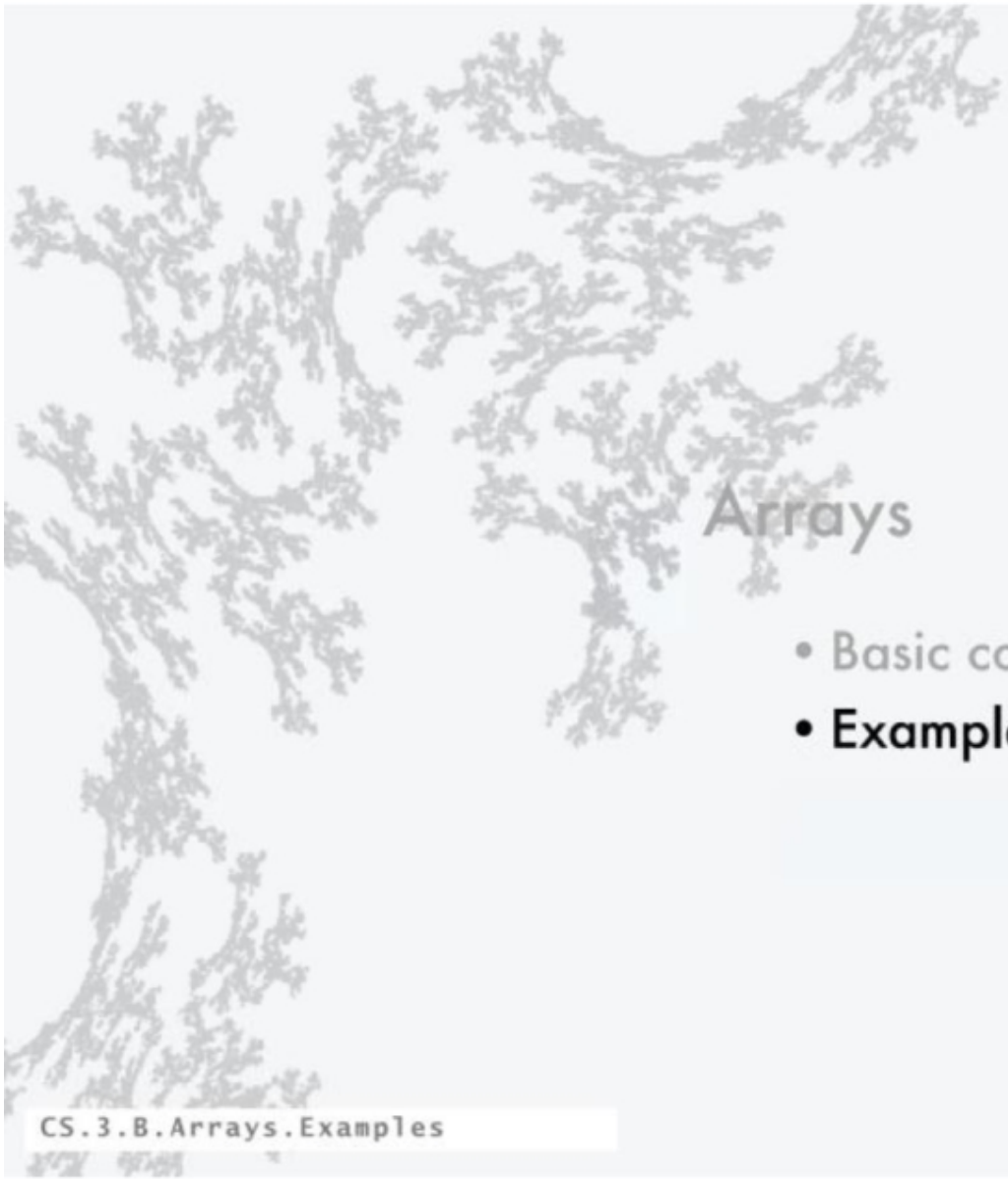
```
double[] a;
for (int i = 0; i < 9; i++)
   a[i] = Math.random();
```

Never created the array

Undeclared variable

```
a = new double[10];
for (int i = 0; i < 10; i++)
   a[i] = Math.random();
```

What type of data does a refer to?



COMPUTER SCIENCE
SEDGEWICK / WAYNE
PART I: PROGRAMMING IN JAVA

Arrays

- Basic concepts
- **Examples of array-processing code**

Example of array use: create a deck of cards

Define three arrays

- Ranks.
- Suits.
- Full deck.

```
String[] rank = {"2", "3", "4", "5", "6", "7", "8", "9", "10", "J", "Q", "K", "A" };
String[] suit = { "♣", "♦", "♥", "♠" };
String[] deck = new String[52];
```

Use nested for loops to put all the cards in the deck.

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

better style to use rank.length and suit.length
clearer in lecture to use 4 and 13

	j			
	0	1	2	3
suit	♣	♦	♥	♠

	i												
	0	1	2	3	4	5	6	7	8	9	10	11	12
rank	2	3	4	5	6	7	8	9	10	J	Q	K	A

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	...
deck	2♣	3♣	4♣	5♣	6♣	7♣	8♣	9♣	10♣	J♣	Q♣	K♣	A♣	2♦	3♦	4♦	5♦	6♦	7♦	8♦	9♦	...

Main page
Contents
Featured content
Current events
Random article
Donate to Wikipedia
Wikipedia store

Interaction

Help
About Wikipedia
Community portal
Recent changes
Contact page

Tools

What links here
Related changes
Upload file
Special pages

Playing cards in Unicode

From Wikipedia, the free encyclopedia

Unicode is a computing industry standard for the handling of fonts and symbols. Within it is a set of images depicting playing cards, and another depicting the French card suits.

Contents [show]

Card suits [edit]

The Miscellaneous Symbols block contains the following, at U+2660–2667:

U+2660	U+2665	U+2666	U+2663
♠	♥	♦	♣
Black Spade Suit	Black Heart Suit	Black Diamond Suit	Black Club Suit
&spades;	&hearts;	&diams;	&clubs;
U+2664	**U+2661**	**U+2662**	**U+2667**
♤	♡	♢	♧
White Spade Suit	White Heart Suit	White Diamond Suit	White Club Suit

U+1F0A3	U+1F0B3	U+1F0C3	U+1F0D3
🂣	🂳	🃃	🃓
Three of Spades	Three of Hearts	Three of Diamonds	Three of Clubs

Example of array use: create a deck of cards

```
public class Deck
{
    public static void main(String[] args)
    {
        String[] rank = {"2", "3", "4", "5", "6", "7", "8", "9", "10", "J", "Q", "K", "A" };
        String[] suit = { "♣", "♦", "♥", "♠" };

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

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

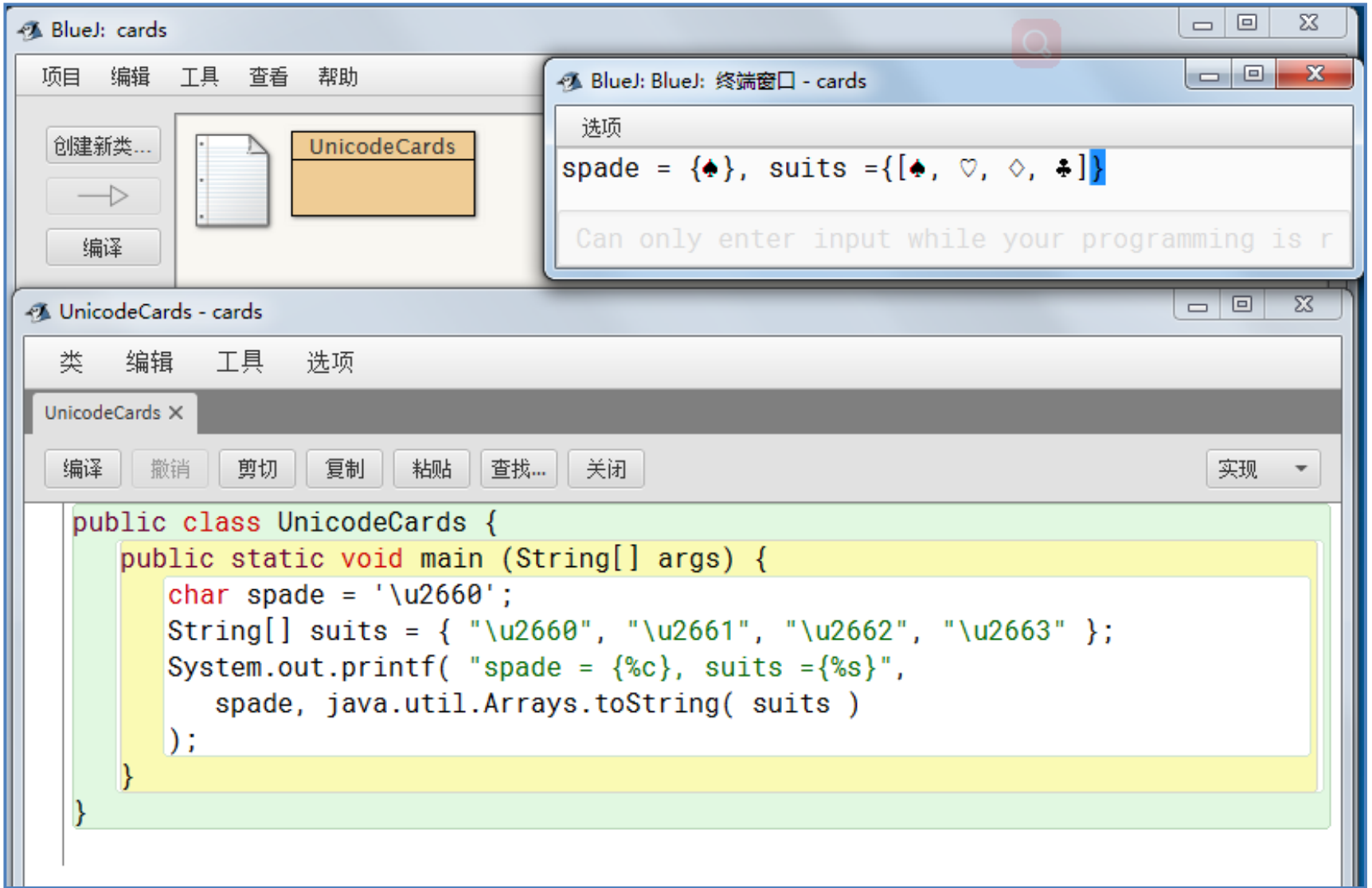
no color in Unicode;
artistic license for lecture

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

```
H:\course\2018-2019A\JavaUI\work\WarmUp03_code>javac UnicodeCards.java

H:\course\2018-2019A\JavaUI\work\WarmUp03_code>java UnicodeCards
spade = {?}, suits ={[?, ?, ?, ?]}
H:\course\2018-2019A\JavaUI\work\WarmUp03_code>java UnicodeCards > cards.txt

H:\course\2018-2019A\JavaUI\work\WarmUp03_code>
```

cards.txt

```
1 spade = {?}, suits ={[?, ?, ?, ?]}
```



Pop quiz 2 on arrays

Q. What happens if the order of the for loops in Deck is switched?

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

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

Pop quiz 2 on arrays

Q. What happens if the order of the for loops in Deck is switched?

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

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

A. The array is filled in a different order, but the output is the same.

	j			
	0	1	2	3
suit	♣	♦	♥	♠

	i												
	0	1	2	3	4	5	6	7	8	9	10	11	12
rank	2	3	4	5	6	7	8	9	10	J	Q	K	A

	0	1	2	...	12	13	14	15	...	25	26	27	28	...	38	39	40	41	...	51
deck	2♣	3♣	4♣	...	A♣	2♦	3♦	4♦	...	A♦	2♥	3♥	4♥	...	A♥	2♠	3♠	4♠	...	A♠

NOTE: Error on page 92 in 3rd printing of text (see errata list on booksite).

Pop quiz 3 on arrays

Q. Change Deck to put the cards in rank order in the array.

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

Pop quiz 3 on arrays

Q. Change Deck to put the cards in rank order in the array.

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

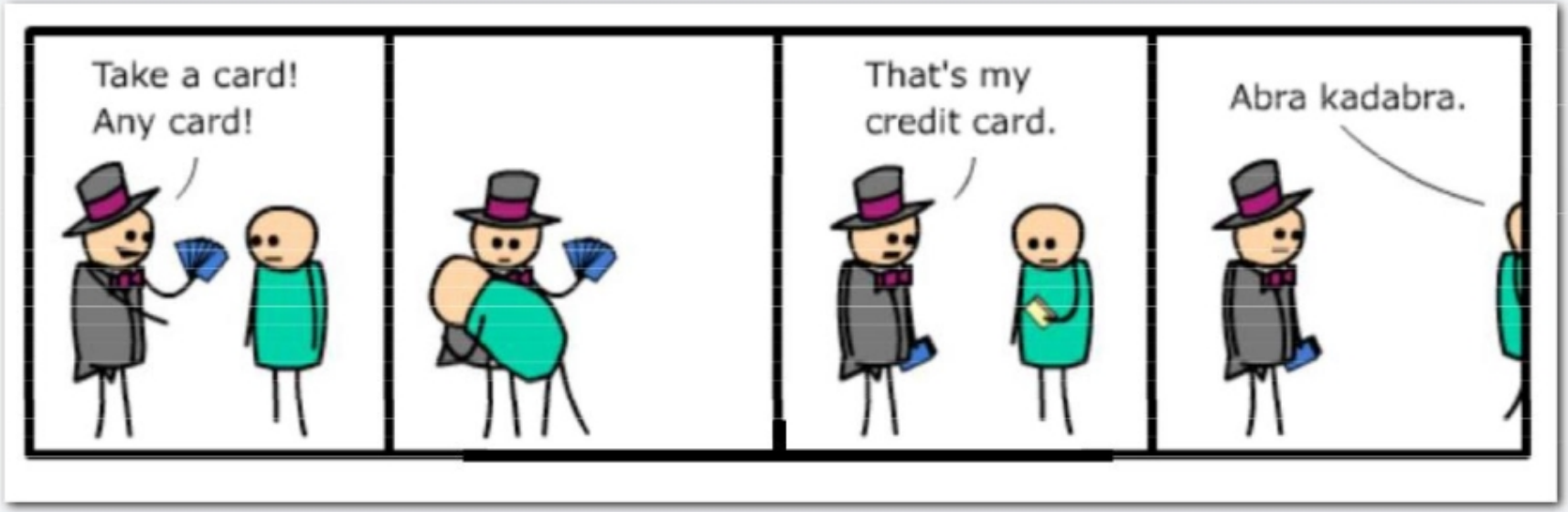
A.

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

	j			
	0	1	2	3
suit	♣	♦	♥	♠

	i												
	0	1	2	3	4	5	6	7	8	9	10	11	12
rank	2	3	4	5	6	7	8	9	10	J	Q	K	A

	0	1	2	3	4	5	6	7	8	9	10	11	12	...
deck	2♣	2♦	2♥	2♠	3♣	3♦	3♥	3♠	4♣	4♦	4♥	4♠	5♣	...


Take a card!
Any card!
That's my
credit card.
Abra kadabra.

Array application: take a card, any card

Problem: Print a random sequence of *N* cards.

Algorithm

Take *N* from the command line and do the following *N* times

- Calculate a random index *r* between 0 and 51.
- Print `deck[r]`.

Implementation: Add this code instead of printing deck in `Deck`.

```
for (int i = 0; i < N; i++)
{
   int r = (int) (Math.random() * 52);
   System.out.println(deck[r]);
}
```

each value between 0 and 51 equally likely

Note: Same method is effective for printing a random sequence from any data collection.

Array application: random sequence of cards

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

      String[] rank = {"2", "3", "4", "5", "6", "7", "8", "9","10", "J", "Q", "K", "A" };
      String[] suit = { "♣", "♦", "♥", "♠" };

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

      for (int i = 0; i < N; i++)
      {
         int r = (int) (Math.random() * 52);
         System.out.print(deck[r] + " ");
      }
      System.out.println();
   }
}
```

```
% java DrawCards 10
6♥ K♦ 10♠ 8♦ 9♦ 9♥ 6♦ 10♠ 3♣ 5♦

% java DrawCards 10
2♦ A♠ 5♣ A♣ 10♣ Q♦ K♣ K♠ A♣ A♦

% java DrawCards 10
6♠ 10♦ 4♥ A♣ K♥ Q♠ K♠ 7♣ 5♦ Q♠

% java DrawCards 10
A♣ J♣ 5♥ K♥ Q♣ 5♥ 9♦ 9♣ 6♠ K♥
```

appears twice

Note: Sample is *with* replacement (same card can appear multiple times).

Array application: shuffle and deal from a deck of cards

Problem: Print *N* random cards from a deck.

Algorithm: Shuffle the deck, then deal.

- Consider each card index *i* from 0 to 51.
  - Calculate a random index *r* between *i* and 51.
  - Exchange `deck[i]` with `deck[r]`
- Print the first *N* cards in the deck.

Implementation

```
for (int i = 0; i < 52; i++)
{
   int r = i + (int) (Math.random() * (52-i));
   String t = deck[r];
   deck[r] = deck[i];
   deck[i] = t;
}
for (int i = 0; i < N; i++)
    System.out.print(deck[i]);
System.out.println();
```

each value between *i* and 51 equally likely

Array application: shuffle a deck of 10 cards (trace)

```
for (int i = 0; i < 10; i++)
{
   int r = i + (int) (Math.random() * (10-i));
   String t = deck[r];
   deck[r] = deck[i];
   deck[i] = t;
}
```

Q. Why does this method work?

- Uses only exchanges, so the deck after the shuffle has the same cards as before.
- $N-i$ equally likely values for `deck[i]`.
- Therefore $N \times (N-1) \times (N-1) \ldots \times 2 \times 1 = N!$ equally likely values for `deck[]`.

Initial order is immaterial.

i	r	deck									
		0	1	2	3	4	5	6	7	8	9
		2♣	3♣	4♣	5♣	6♣	7♣	8♣	9♣	10♣	J♣
0	7	9♣	3♣	4♣	5♣	6♣	7♣	8♣	2♣	10♣	J♣
1	3	9♣	5♣	4♣	3♣	6♣	7♣	8♣	2♣	10♣	J♣
2	9	9♣	5♣	J♣	3♣	6♣	7♣	8♣	2♣	10♣	4♣
3	9	9♣	5♣	J♣	4♣	6♣	7♣	8♣	2♣	10♣	3♣
4	6	9♣	5♣	J♣	4♣	8♣	7♣	6♣	2♣	10♣	3♣
5	9	9♣	5♣	J♣	4♣	8♣	3♣	6♣	2♣	10♣	7♣
6	8	9♣	5♣	J♣	4♣	8♣	3♣	10♣	2♣	6♣	7♣
7	9	9♣	5♣	J♣	4♣	8♣	3♣	10♣	7♣	6♣	2♣
8	8	9♣	5♣	J♣	4♣	8♣	3♣	10♣	7♣	6♣	2♣
9	9	9♣	5♣	J♣	4♣	8♣	3♣	10♣	7♣	6♣	2♣

Note: Same method is effective for randomly rearranging any type of data.

Array application: shuffle and deal from a deck of cards

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

       String[] rank = {"2", "3", "4", "5", "6", "7", "8", "9", "10", "J", "Q", "K", "A" };
       String[] suit = { "♣", "♦", "♥", "♠" };

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

       for (int i = 0; i < 52; i++)
       {
          int r = i + (int) (Math.random() * (52-i));
          String t = deck[r];
          deck[r] = deck[i];
          deck[i] = t;
       }

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

random poker hand

```
% java DealCards 5
9♣ Q♥ 6♥ 4♦ 2♠
```

random bridge hand

```
% java DealCards 13
3♠ 4♥ 10♦ 6♥ 6♦ 2♠ 9♣ 8♠ A♠ 3♥ 9♠ 5♠ Q♥
```

Coupon collector

Coupon collector problem

- M different types of coupons.
- Collector acquires random coupons, one at a time, each type equally likely.

Q. What is the expected number of coupons needed to acquire a full collection?

Example: Collect all ranks in a random sequence of cards ($M =$

Sequence

9♣	5♠	8♥	10♦	2♠	A♠	10♥	Q♦	3♠	9♥	5♦	9♣	7♦	2♦	8♣	6♣	Q♥	K♣	10♥	A♦	4♦	J♥

Collection

2	3	4	5	6	7	8	9	10	J	Q	K	A
2♠	3♠	4♦	5♠	6♣	7♦	8♥	9♣	10♦	J♥	Q♦	K♣	A♠
2♦			5♦			8♣	9♥	10♥		Q♥		A♦
							9♣	10♥				

22 cards needed to complete collection

Array application: coupon collector

Coupon collector simulation

- Generate random `int` values between 0 and $M-1$.
- Count number used to generate each value at least once.

Key to the implementation

- Create a boolean array of length M. (Initially all `false` by default.)
- When r generated, check the rth value in the array.
  - If `true`, ignore it (not new).
  - If `false`, count it as new distinct value (and set rth entry to `true`)

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

      boolean[] found = new boolean[M];
      while (distinct < M)
      {
         int r = (int) (Math.random() * M);
         cards++;
         if (!found[r])
         {
            distinct++;
            found[r] = true;
         }
      }

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

```
% java Coupon 13
46

% java Coupon 13
22

% java Coupon 13
54

% java Coupon 13
27
```

Array application: coupon collector (trace for M = 6)

```
boolean[] found = new boolean[M];
while (distinct < M)
{
   int r = (int) (Math.random() * M);
   cards++;
   if (!found[r])
   {
      distinct++;
      found[r] = true;
   }
}
```

r	found						distinct	cards
	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

Simulation, randomness, and analysis (revisited)

Coupon collector problem

- M different types of coupons.
- Collector acquires random coupons, one at a time, each type equally likely.

Q. What is the expected number of coupons needed to acquire a full collection?

Pierre-Simon Laplace
1749-1827

A. (known via mathematical analysis for centuries) About $M \ln M + .57721M$.

type	*M*	*expected wait*
playing card suits	4	8
playing card ranks	13	41
baseball cards	1200	9201
Magic™ cards	12534	125508

```
% java Coupon 4
11

% java Coupon 13
38

% java Coupon 1200
8789

% java Coupon 12534
125671
```

Remarks

- Computer simulation can help validate mathematical analysis.
- Computer simulation can also validate software behavior.

Example: Is `Math.random()` simulating randomness?

Simulation, randomness, and analysis (revisited)

Once simulation is debugged, experimental evidence is easy to obtain.

Gambler's ruin simulation, previous lecture

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

      int wins   = 0;
      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++;
      }
      System.out.println(wins + " wins of " + trials);
   }
}
```

Analogous code for coupon collector, this lecture

```
public class CouponCollector
{
    public static void main(String[] args)
    {
        int M = Integer.parseInt(args[0]);
        int trials = Integer.parseInt(args[1]);
        int cards = 0;
        boolean[] found;

        for (int i = 0; i < trials; i++)
        {
           int distinct = 0;
           found = new boolean[M];
           while (distinct < M)
           {
              int r = (int) (Math.random() * M);
              cards++;
              if (!found[r])
              {
                 distinct++;
                 found[r] = true;
              }
           }
        }
        System.out.println(cards/trials);
    }
}
```

Simulation, randomness, and analysis (revisited)

Coupon collector problem

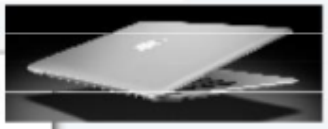
- M different types of coupons.
- Collector acquires random coupons, one at a time, each type equally likely.

Q. What is the expected number of coupons needed to acquire a full collection?

Predicted by mathematical analysis

type	M	$M \ln M + .57721M$
playing card suits	4	8
playing card ranks	13	41
playing cards	52	236
baseball cards	1200	9201
magic cards	12534	125508

Observed by computer simulation



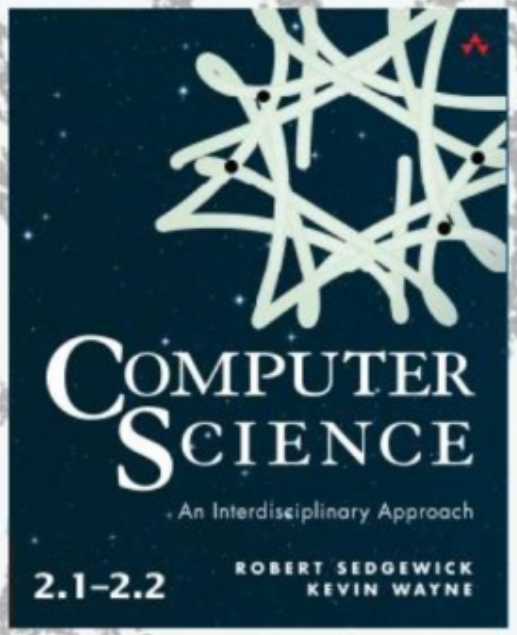
```
% java CouponCollector 4 1000000
8

% java CouponCollector 13 1000000
41

% java CouponCollector 52 100000
236

% java CouponCollector 1200 10000
9176

% java CouponCollector 12534 1000
125920
```

Hypothesis. Centuries-old analysis is correct and `Math.random()` simulates randomness.

COMPUTER SCIENCE

SEDGEWICK / WAYNE

PART I: PROGRAMMING IN JAVA



http://introcs.cs.princeton.edu

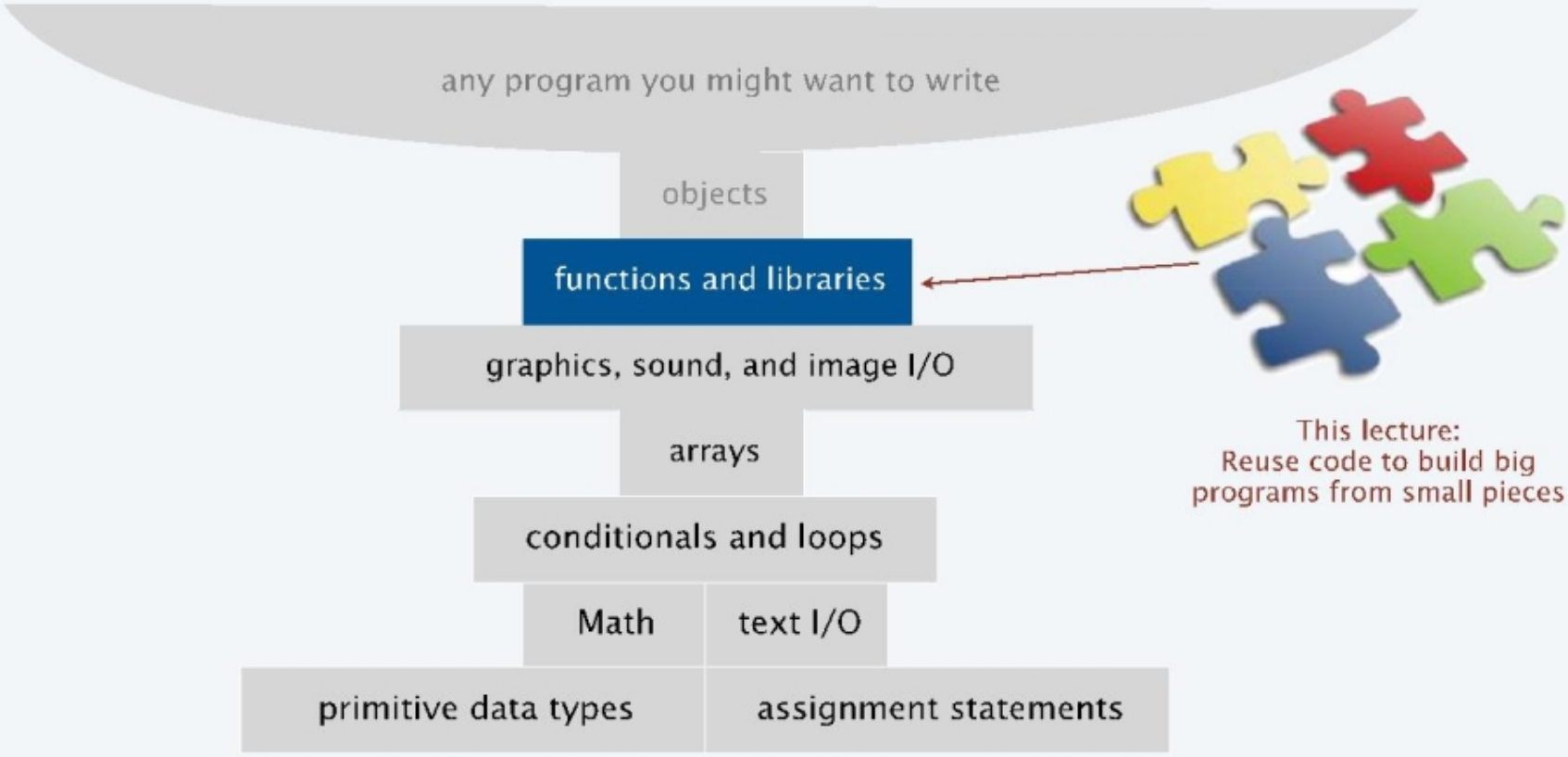
Functions and Libraries

COMPUTER SCIENCE
SEDGEWICK / WAYNE
PART I: PROGRAMMING IN JAVA

5. Functions and Libraries

- **Basic concepts**
- Application: Gaussian distribution
- Modular programming and libraries

Context: basic building blocks for programming



Functions, libraries, and modules

Modular programming

- Organize programs as independent modules that do a job together.
- Why? Easier to share and reuse code to build bigger programs.

Facts of life

- Support of modular programming has been a holy grail for decades.
- Ideas can conflict and get highly technical in the real world.

Def. A library is a set of functions.

↑ for purposes of this lecture

Def. A module is a `.java` file.

↑ for purposes of this course

For now. Libraries and modules are the *same thing*: `.java` files containing sets of functions.

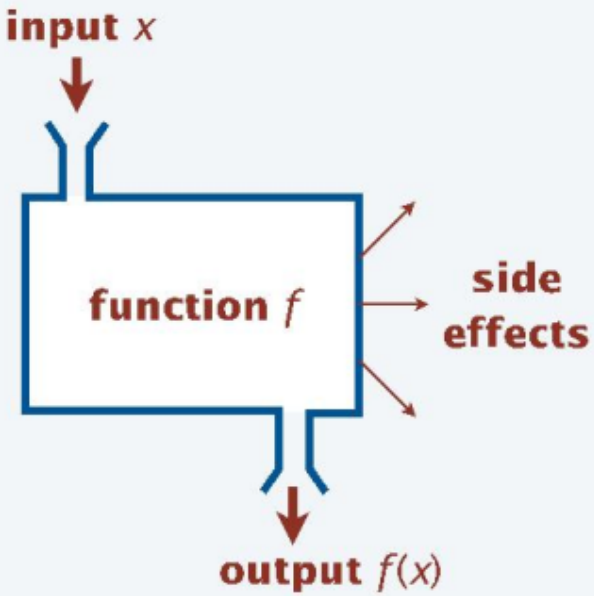
Later. Modules implement *data types* (stay tuned).

Functions (static methods)

Java *function* ("aka *static method*")

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

Java functions are *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.

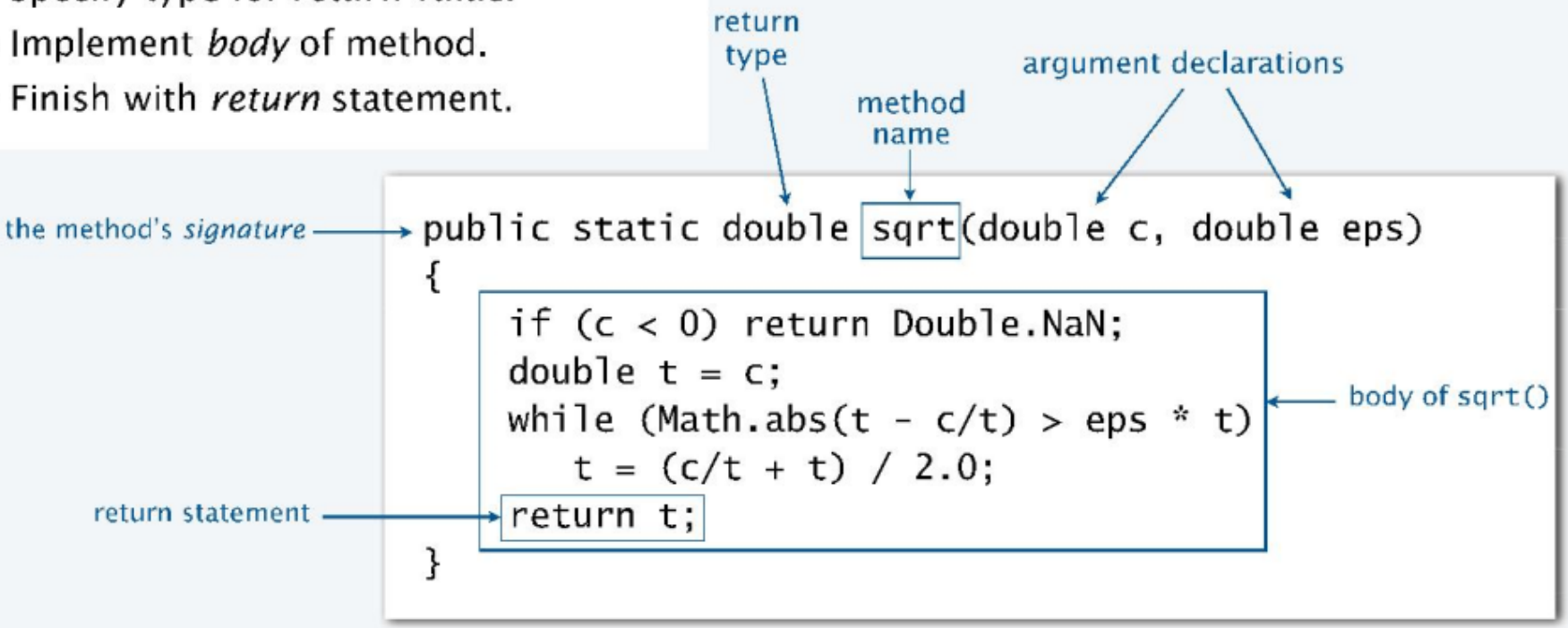
Examples seen so far

- 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 static method

To implement a function (static method)

- Create a *name*.
- Declare type and name of *argument*(*s*).
- Specify type for *return value*.
- Implement *body* of method.
- Finish with *return* statement.



Anatomy of a Java library

A library is a set of functions.

Note: We are using our sqrt() from Lecture 2 here to illustrate the basics with a familiar function.

Our focus is on control flow here. See Lecture 2 for technical details.

You can use Math.sqrt().

sqrt() method

module named Newton.java

main() method

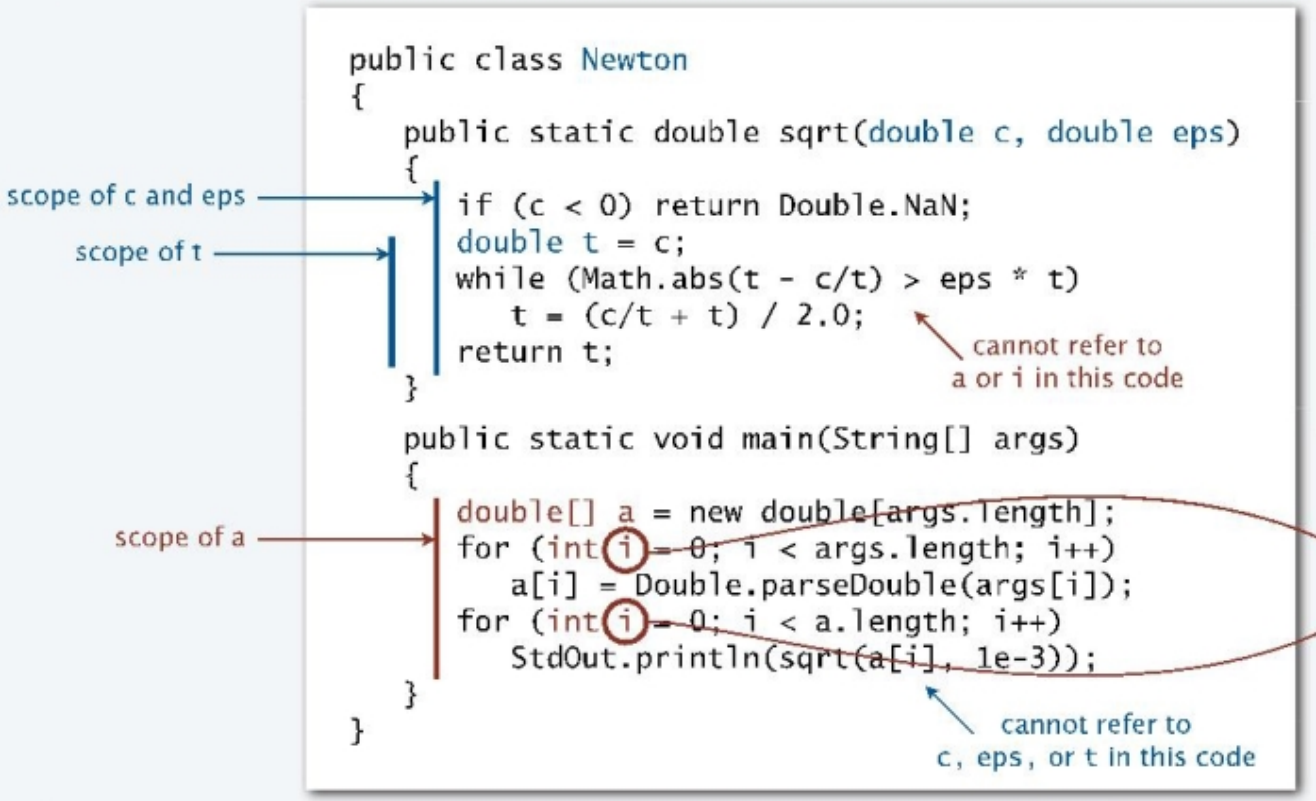
```
public class Newton
{
   public static double sqrt(double c, double eps)
   {
      if (c < 0) return Double.NaN;
      double t = c;
      while (Math.abs(t - c/t) > eps * t)
         t = (c/t + t) / 2.0;
      return t;
   }

   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++)
         StdOut.println(sqrt(a[i], 1e-3));
   }
}
```

library/module name

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

Scope

Def. The scope of a variable is the code that can refer to it by name.



In a Java library, a variable's scope is the code following its declaration, in the same block.

Best practice. Declare variables so as to *limit* their scope.

Flow of control

```
public class Newton
{
   public static double sqrt(double c, double eps)
   {
      if (c < 0) return Double.NaN;
      double t = c;
      while (Math.abs(t - c/t) > eps * 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], 1e-3);
         StdOut.println(x);
      }
   }
}
```

Summary of flow control for a function call

- Control transfers to the function code.
- Argument variables are declared and initialized with the given values.
- Function code is executed.
- Control transfers back to the calling code (with return value assigned in place of the function name in the calling code).

"pass by value"
(other methods used in other systems)

Note. OS calls `main()` on `java` command

Function call flow of control trace

```
public class Newton
{
   public static double sqrt(double c, double eps)
   {
      if (c < 0) return Double.NaN;
      double t = c;
      while (Math.abs(t - c/t) > eps * 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], 1e-3);
         StdOut.println(x);
      }
   }
}
```

c	t
3.0	3.0
	2.0
	1.75
	1.732

i	a[i]	x
0	1.0	1.000
1	2.0	1.414
2	3.0	1.732
3		

```
% java Newton 1 2 3
1.000
1.414
1.732
```

Pop quiz 1a on functions

Q. What happens when you compile and run the following code?

```
public class PQfunctions1a
{
   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));
   }
}
```

Pop quiz 1a on functions

Q. What happens when you compile and run the following code?

```
public class PQfunctions1a
{
   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));
   }
}
```

A. Takes *N* from the command line, then prints cubes of integers from 1 to *N*

```
% javac PQfunctions1a.java
% java PQfunctions1a 6
1 1
2 8
3 27
4 64
5 125
6 216
```

Pop quiz 1b on functions

Q. What happens when you compile and run the following code?

```
public class PQfunctions1b
{
   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));
   }
}
```

Pop quiz 1b on functions

Q. What happens when you compile and run the following code?

```
public class PQfunctions1b
{
   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));
   }
}
```

A. Won't compile. Argument variable i is declared and initialized for function block, so the name cannot be reused.

```
% javac PQfunctions1b.java
PQfunctions1b.java:5: i is already defined in cube(int)
      int i = i * i * i;
          ^
1 error
```

Pop quiz 1c on functions

Q. What happens when you compile and run the following code?

```
public class PQfunctions1c
{
   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));
   }
}
```

Pop quiz 1c on functions

Q. What happens when you compile and run the following code?

```
public class PQ6_1c
{
   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));
   }
}
```

A. Won't compile. Need return statement.

```
% javac PQfunctions1c.java
PQfunctions1c.java:6: missing return statement
   }
   ^
1 error
```

Pop quiz 1d on functions

Q. What happens when you compile and run the following code?

```
public class PQfunctions1d
{
   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));
   }
}
```

Pop quiz 1d on functions

Q. What happens when you compile and run the following code?

```
public class PQfunctions1d
{
   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));
   }
}
```

A. Works. The `i` in `cube()` is

- Declared and initialized as an argument.
- Different from the `i` in `main()`.

BUT changing values of function arguments is sufficiently confusing to be deemed bad style for this course.

```
% javac PQfunctions1d.java
% java PQfunctions1d 6
1 1
2 8
3 27
4 64
5 125
6 216
```

Pop quiz 1e on functions

Q. What happens when you compile and run the following code?

```
public class PQfunctions1e
{
   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));
   }
}
```

Pop quiz 1e on functions

Q. What happens when you compile and run the following code?

```
public class PQfunctions1e
{
   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));
   }
}
```

A. Works fine. Preferred (compact) code.

```
% javac PQfunctions1e.java
% java PQfunctions1e 6
1 1
2 8
3 27
4 64
5 125
6 216
```

COMPUTER SCIENCE
SEDGEWICK / WAYNE
PART I: PROGRAMMING IN JAVA

5. Functions and Libraries

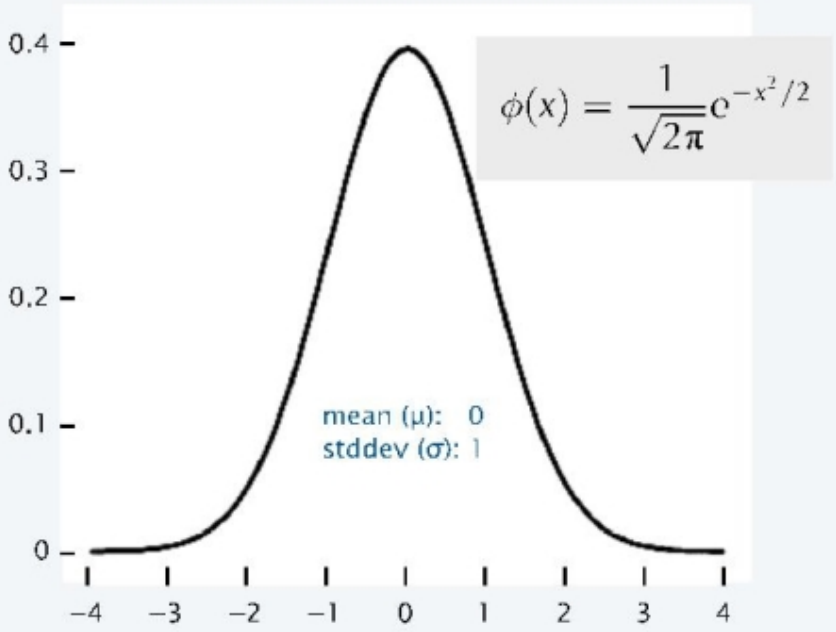
- Basic concepts
- **Application: Gaussian distribution**
- Modular programming and libraries

Gaussian distribution

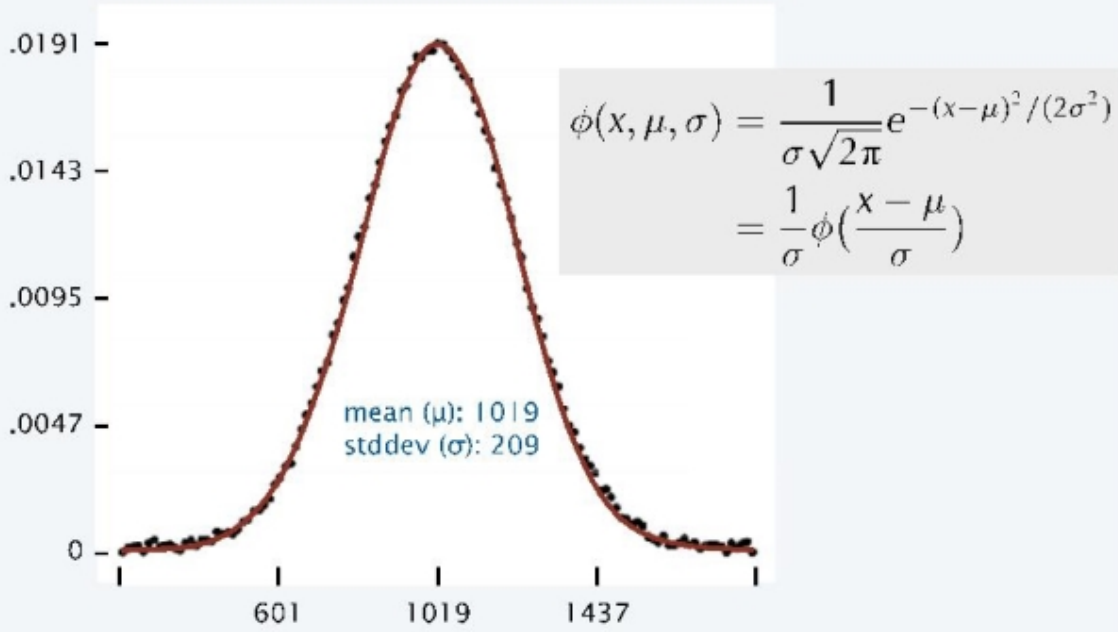
Gaussian distribution.

- A mathematical model used successfully for centuries.
- "Bell curve" fits experimental observations in many contexts.

Gaussian probability density function (pdf)

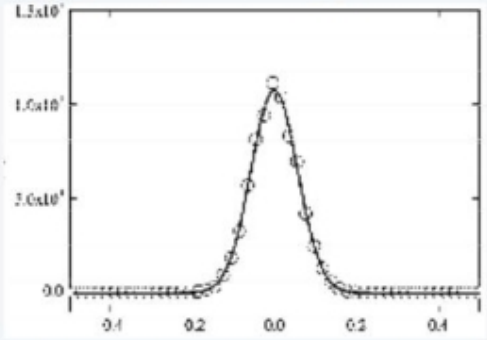


Example: SAT scores in 20xx (verbal + math)



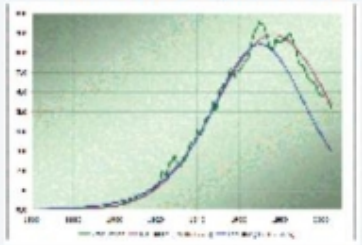
Gaussian distribution in the wild

Polystyrene particles in glycerol

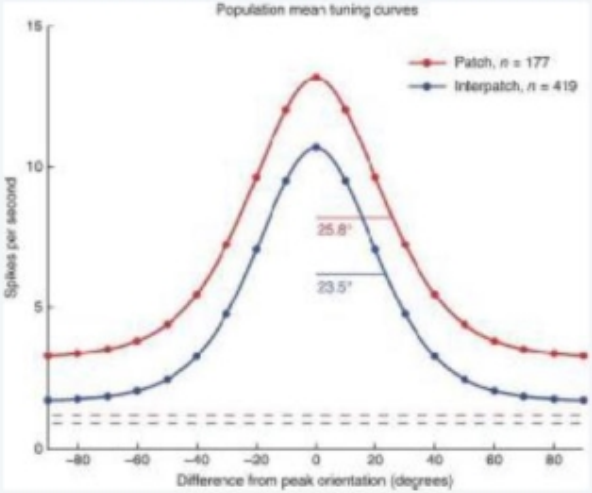


German money

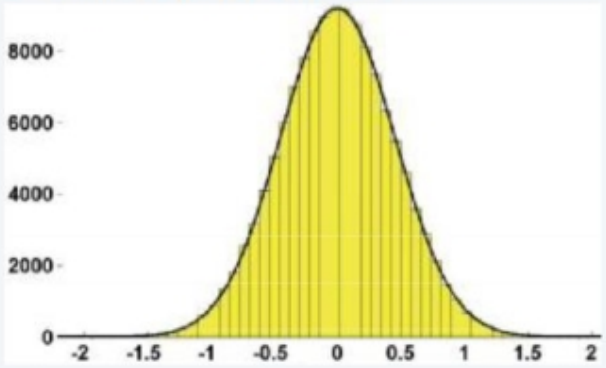
Predicted US oil production



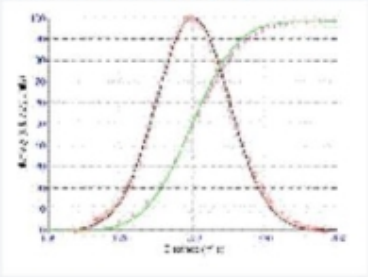
Cytochrome oxidase patches
in macaque primary visual cortex



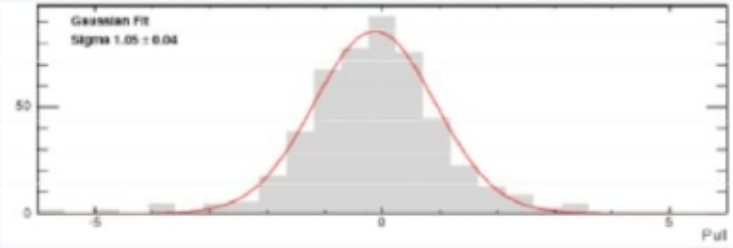
Calibration of optical tweezers



Laser beam propagation



Polarized W bosons from top-quark decay



Defining a library of functions

Q. Is the Gaussian pdf Φ implemented in Java's Math library?

A. No.

Q. Why not?

A. Maybe because it is so easy for you to do it yourself.

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

$$\phi(x, \mu, \sigma) = \frac{1}{\sigma}\phi\left(\frac{x - \mu}{\sigma}\right)$$

```
public class Gaussian
{
   public static double pdf(double x)
   {
      return Math.exp(-x*x / 2) / Math.sqrt(2 * Math.PI);
   }

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

   // Stay tuned for more functions.
}
```

call a function in another module

module named Gaussian.java

functions with different signatures are different (even if names match)

Functions and libraries provide an easy way for any user to *extend* the Java system.

Gaussian cumulative distribution function

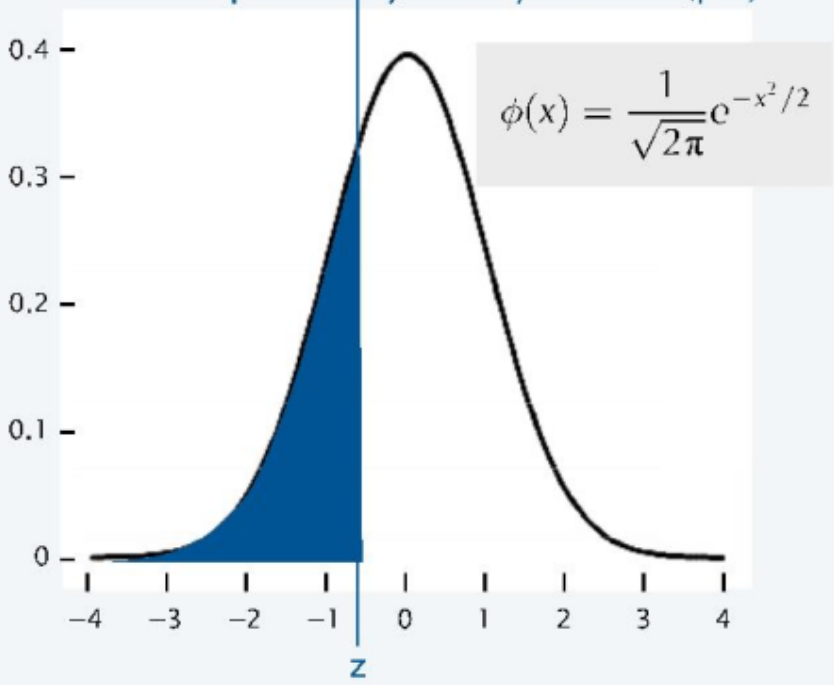
Q. What percentage of the total is less than or equal to *z*?

Q. (equivalent). What is the area under the curve to the left of *z*?

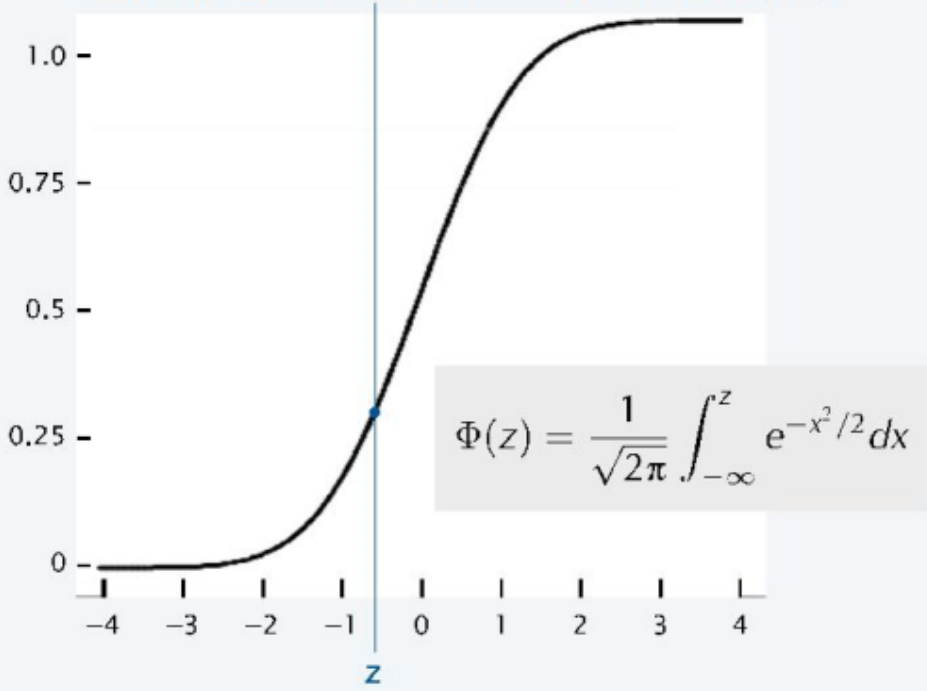
A. Gaussian *cumulative distribution function.*

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

Gaussian probability density function (pdf)



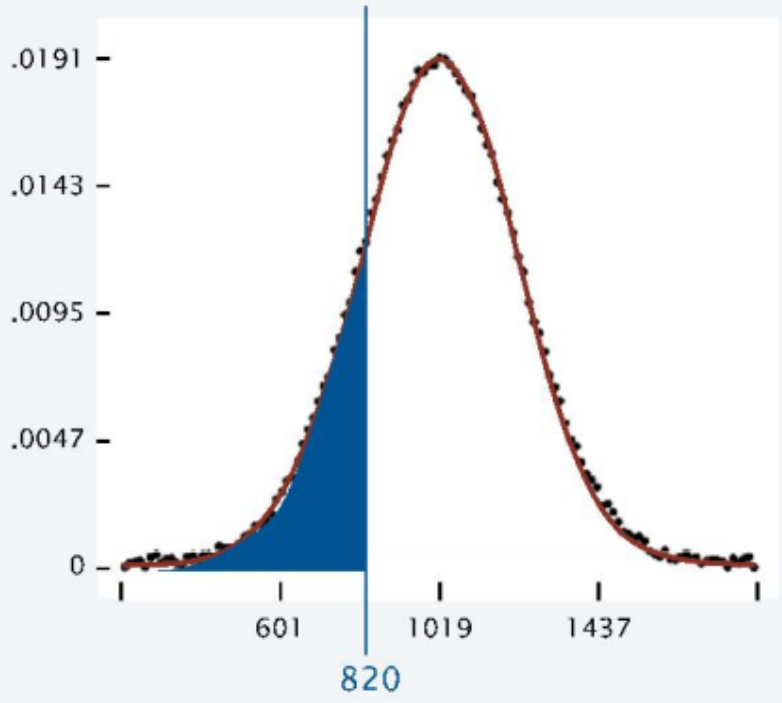
Gaussian cumulative distribution function (cdf)

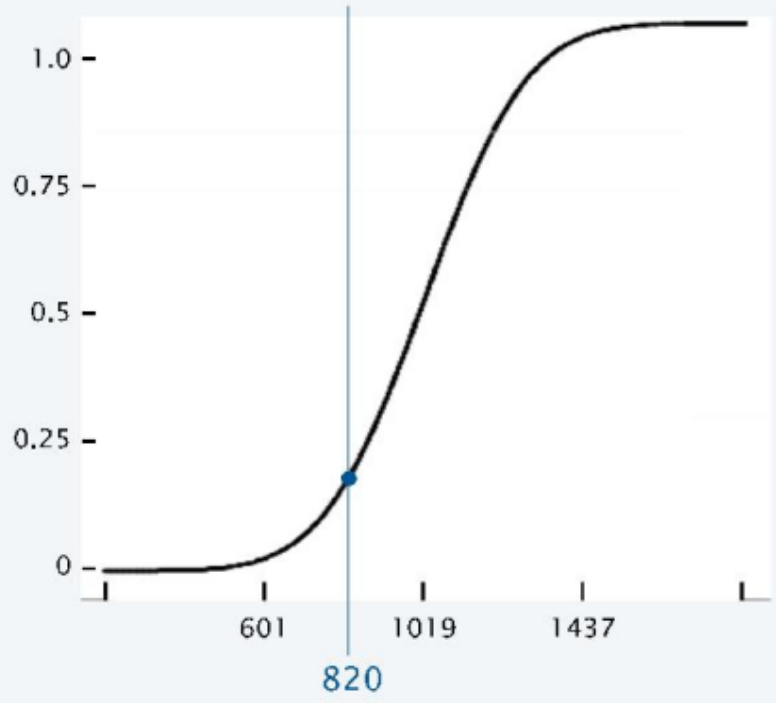


Typical application: SAT scores

Q. In 20xx NCAA required at least 820 for Division I athletes.
What fraction of test takers did not qualify??

A. About 17%, since $\Phi(820, 1019, 209) = 0.17050966869132111...$





Gaussian CDF implementation

Q. No closed form for Gaussian CDF Φ. How to implement?

A. Use Taylor series. $\Phi(z) \equiv \int_{-\infty}^{z} \phi(x)dx = \frac{1}{2} + \phi(z)\left(z + \frac{z^3}{3} + \frac{z^5}{3 \cdot 5} + \frac{z^7}{3 \cdot 5 \cdot 7} + \ldots\right)$

```
public static double cdf(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 * pdf(z);
}

public static double cdf(double z, double mu, double sigma)
{  return cdf((z - mu) / sigma);  }
```

← accurate to 15 places

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

Summary: a library for Gaussian distribution functions

Best practice

- Test all code at least once in `main()`.
- Also have it do something useful.

Q. What fraction of SAT test takers did not qualify for NCAA participation in 20xx?

```
% java Gaussian 820 1019 209
0.17050966869132111
```

Fun fact

We use `cdf()` to evaluate randomness in submitted programs.

Bottom line

YOU can build a layer of abstraction to use in any future program.

```
public class Gaussian
{
   public static double pdf(double x)
   {  return Math.exp(-x*x / 2) / Math.sqrt(2 * Math.PI);  }
   public static double pdf(double x, double mu, double sigma)
   {  return pdf((x - mu) / sigma) / sigma; }
   public static double cdf(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 * pdf(z);
   }
   public static double cdf(double z, double mu, double sigma)
   {  return cdf((z - mu) / sigma);  }
   public static void main(String[] args)
   {
      double z = Double.parseDouble(args[0]);
      double mu = Double.parseDouble(args[1]);
      double sigma = Double.parseDouble(args[2]);
      StdOut.println(cdf(z, mu, sigma));
   }
}
```

Using a library

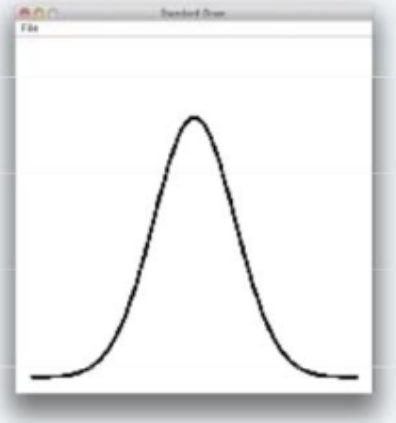
To use these methods in another program

- Put a copy of `Gaussian.java` in your working directory.
- Call `Gaussian.pdf()` or `Gaussian.cdf()` from any other module in that directory.

Learn to use your OS "classpath" mechanism if you find yourself with too many copies.

Example. Draw a plot of $\phi(x, 0, 1)$ in (–4, 4)

```
% java GaussianPlot 200
```



Libraries of functions provide an easy way for *any* user (you) to extend the Java system.

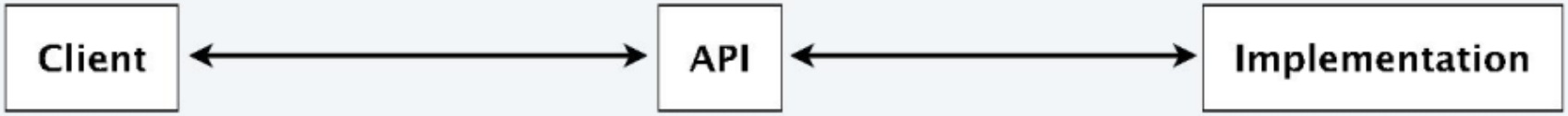
```
public class GaussianPlot
{
    public static void main(String[] args)
    {
        int N = Integer.parseInt(args[0]);
        StdDraw.setXscale(-4.0, +4.0);
        StdDraw.setYscale(0, .5);
        StdDraw.setPenRadius(0.01);
        double[] x = new double[N+1];
        double[] y = new double[N+1];
        for (int i = 0; i <= N; i++)
        {
           x[i] = -4.0 + 8.0 * i / N;
           y[i] = Gaussian.pdf(x[i]);
        }
        for (int i = 0; i < N; i++)
           StdDraw.line(x[i], y[i], x[i+1], y[i+1]);
    }
}
```

COMPUTER SCIENCE
SEDGEWICK / WAYNE
PART I: PROGRAMMING IN JAVA

5. Functions and Libraries

- Basic concepts
- Application: Gaussian distribution
- **Modular programming** and libraries

Fundamental abstractions for modular programming



Client
Module that calls a library's methods.

Applications programming interface (API)
Defines signatures, describes methods.

Implementation
Module containing library's Java code.

```
public class GaussianPlot
{
   ...
   y[i] = Gaussian.pdf(x[i]);
   ...
}
```

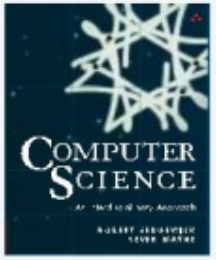
public class Gaussian	
double pdf(double x)	*Gaussian probability density function*
double cdf(double x)	*Gaussian cumulative distribution function*

```
public class Gaussian
{
   public static double pdf(double x)
   {
      double val = Math.exp(-x*x / 2);
      val /=  Math.sqrt(2 * Math.PI);
      return val
   }
   ...
}
```

Example: StdRandom library

Developed for this course, but broadly useful

- Implement methods for generating random numbers of various types.
- Available for download at booksite (and included in introcs software).



API

public class StdRandom	
int uniform(int N)	*integer between* 0 *and* N-1
double uniform(double lo, double hi)	*real between* lo *and* hi
booean bernoulli(double p)	true *with probability* p
double gaussian()	*normal with mean* 0, *stddev* 1
double gaussian(double m, double s)	*normal with mean* m, *stddev* 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.
}
```

First step in developing a library: Articulate the API!

StdRandom details

Implementation

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

   public static int uniform(int N)
   {  return (int) (Math.random() * N);  }

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

   public static double gaussian()
      /* see Exercise 1.2.27 */

   public static double gaussian(double m, double s)
   {  return mean + (stddev * gaussian());  }
   ...
}
```

You *could* implement many of these methods, but now you don't have to!

Typical client

```
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);
      }
   }
}
```

```
% java RandomPoints 10000
```

Best practices

Small modules

- Separate and classify small tasks.
- Implement a layer of abstraction.

Independent development

- Code client *before* coding implementation.
- Anticipate needs of future clients.

```
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
```

Test clients

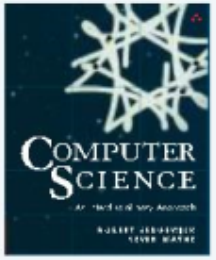
- Include `main()` test client in each module. ← run all code at least once!
- Do more extensive testing in a separate module.

Example: StdStats library

Developed for this course, but broadly useful

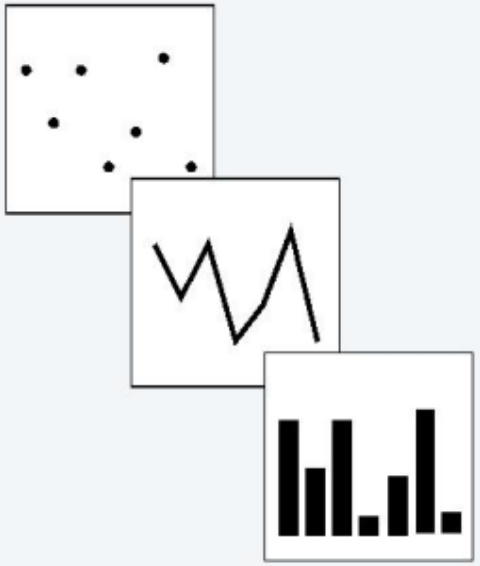
and plotting on StdDraw

- Implement methods for computing statistics on arrays of real numbers.
- Available for download at booksite (and included in introcs software).



API

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)	*plot points at* (i, a[i])
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])



Easy to implement, but easier to use!

← one reason to develop a library: clarify client code

Example of modular programming: StdStats, StdRandom, and Gaussian client

Experiment

- Flip *N* coins.
- How many heads?
- Prediction: Expect *N*/2.

Prediction (more detailed)

- Run experiment *trials* times.
- How many heads?

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

Goal. Write a program to validate predictions.

Example of modular programming: Bernoulli trials

```
public class Bernoulli
{
   public static int binomial(int N)
   // See previous slide.

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

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

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

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

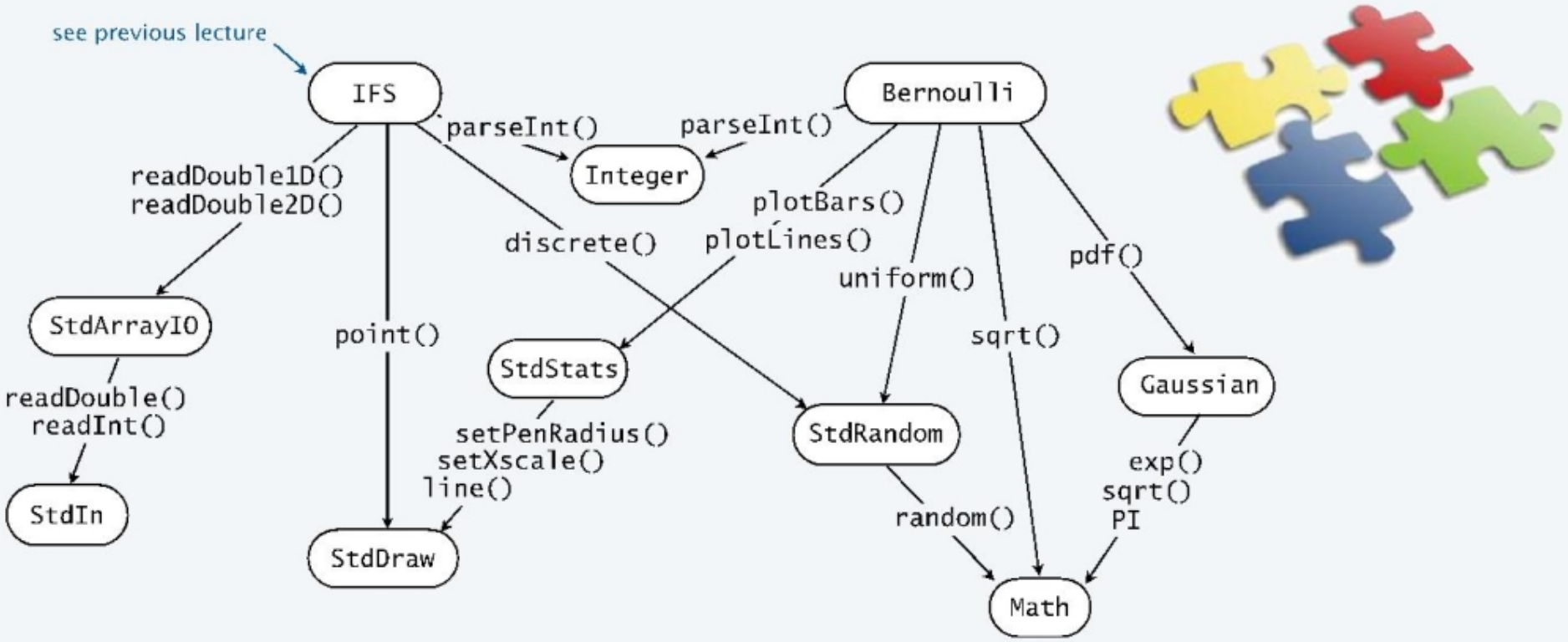
Bernoulli simulation

- Get command-line arguments (*trials* experiments of *N* flips).
- Run experiments. Keep track of frequency of occurrence of each return value.
- Normalize to between 0 and 1. Plot histogram.
- Plot theoretical curve.

```
% java Bernoulli 20 10000
```

Modular programming

enables development of complicated programs via simple independent modules.



Advantages. Code is easier to understand, debug, maintain, improve, and reuse.

Why modular programming?

Modular programming enables

- Independent development of small programs.
- Every programmer to develop and share layers of abstraction.
- Self-documenting code.

Fundamental characteristics

- Separation of client from implementation benefits all *future* clients.
- Contract between implementation and clients (API) benefits all *past* clients.

Challenges

- How to break task into independent modules?
- How to specify API?