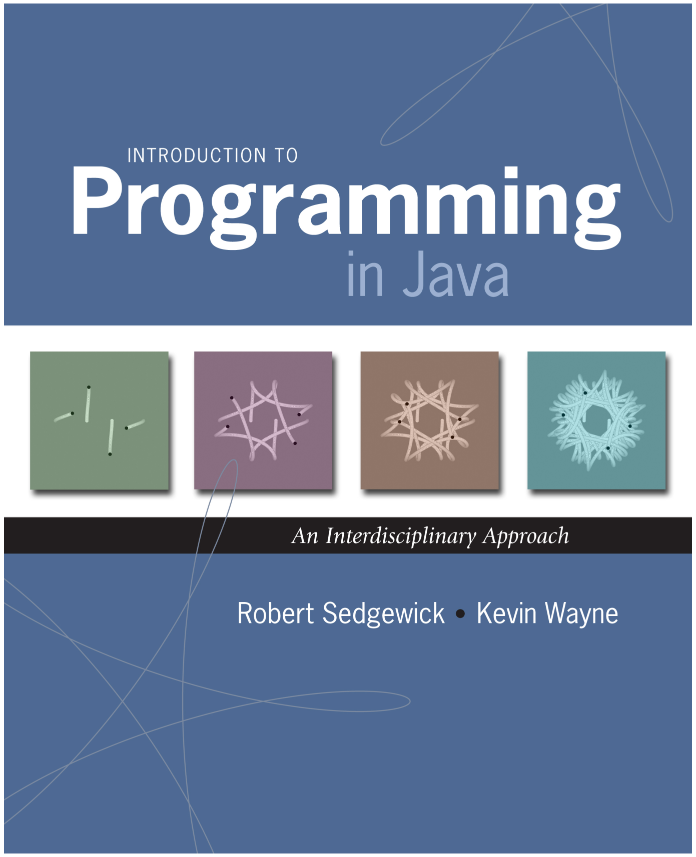
1.3 Conditionals and Loops

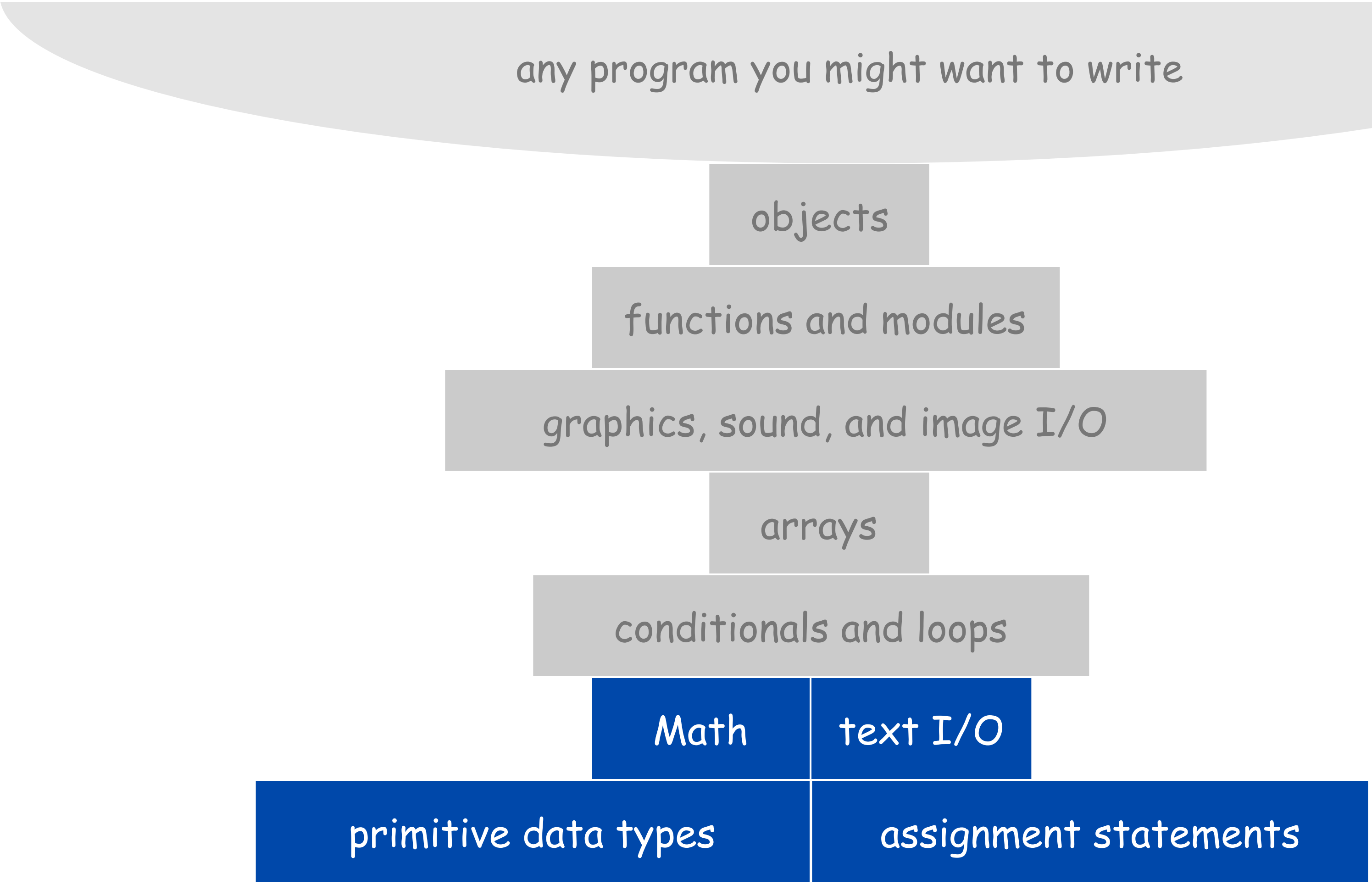


INTRODUCTION TO
Programming
in Java

An Interdisciplinary Approach

Robert Sedgewick • Kevin Wayne

A Foundation for Programming



any program you might want to write

objects

functions and modules

graphics, sound, and image I/O

arrays

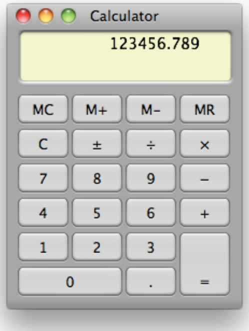
conditionals and loops

Math

text I/O

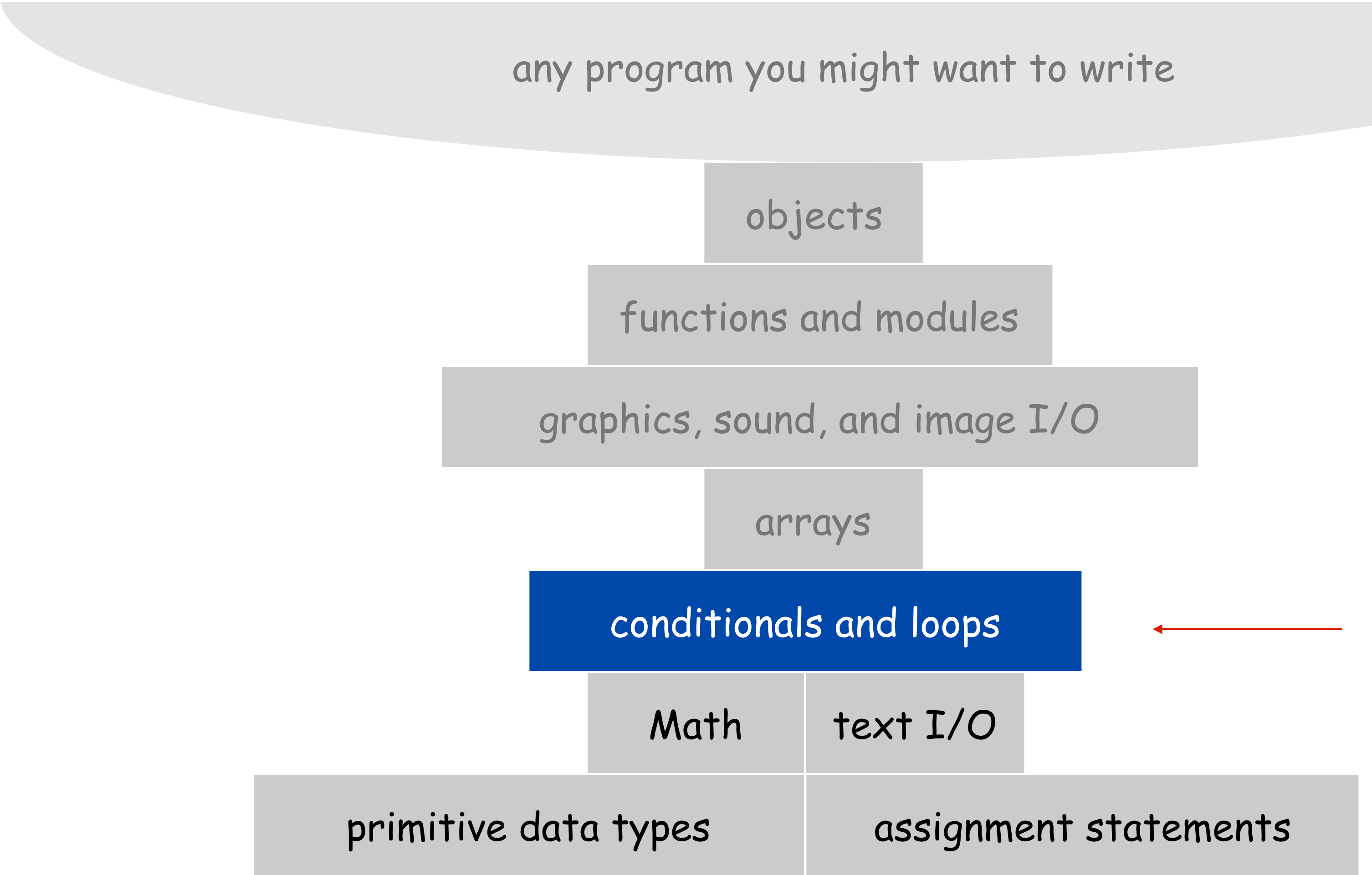
primitive data types

assignment statements



last lecture:
equivalent
to a calculator

A Foundation for Programming



any program you might want to write

objects

functions and modules

graphics, sound, and image I/O

arrays

conditionals and loops

Math

text I/O

primitive data types

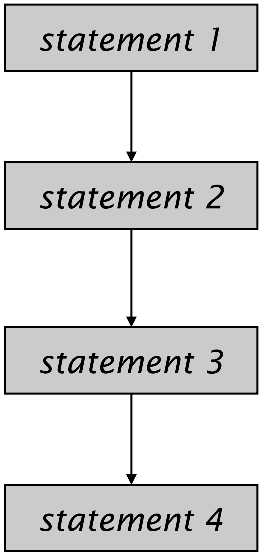
assignment statements

to infinity
and beyond!

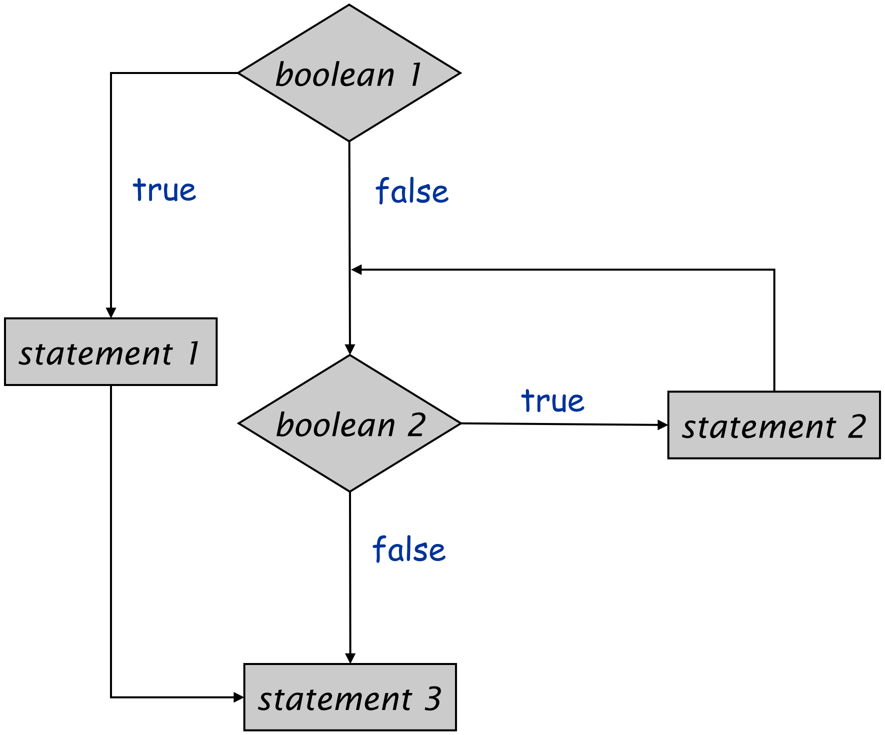
Control Flow

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

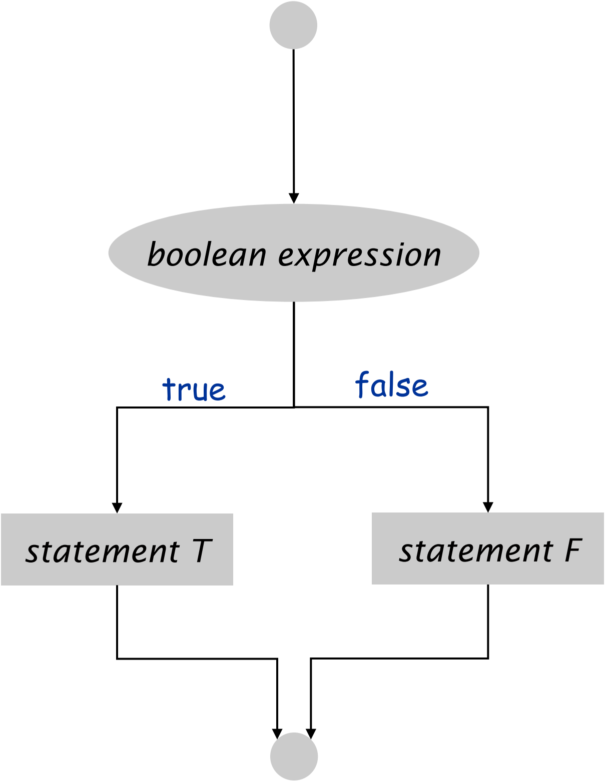
If Statement

The `if` statement. A common branching structure.

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

```
if (boolean expression) {
    statement T;
}
else {
    statement F;
}
```

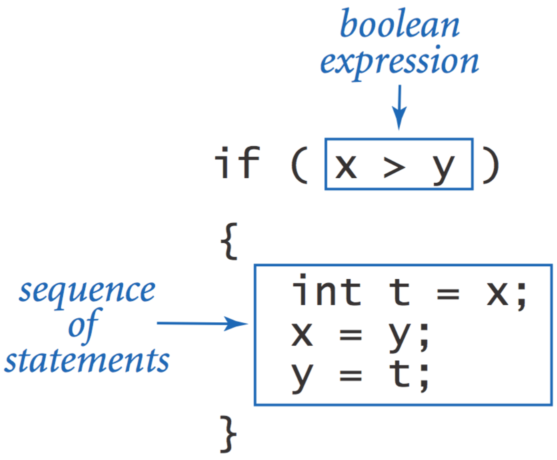
can be any sequence of statements



If Statement

The `if` statement. A common branching structure.

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

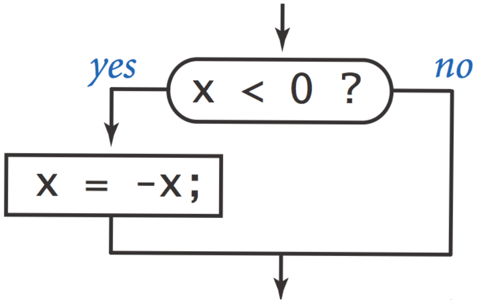


boolean expression

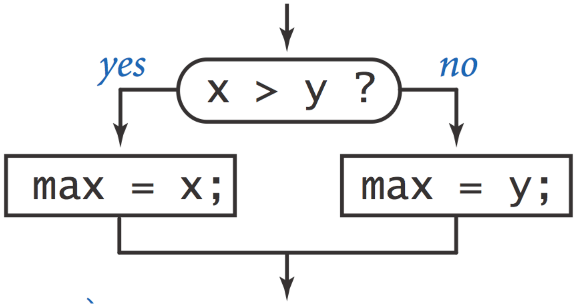
```
if ( x > y )
{
   int t = x;
   x = y;
   y = t;
}
```

sequence of statements

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



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



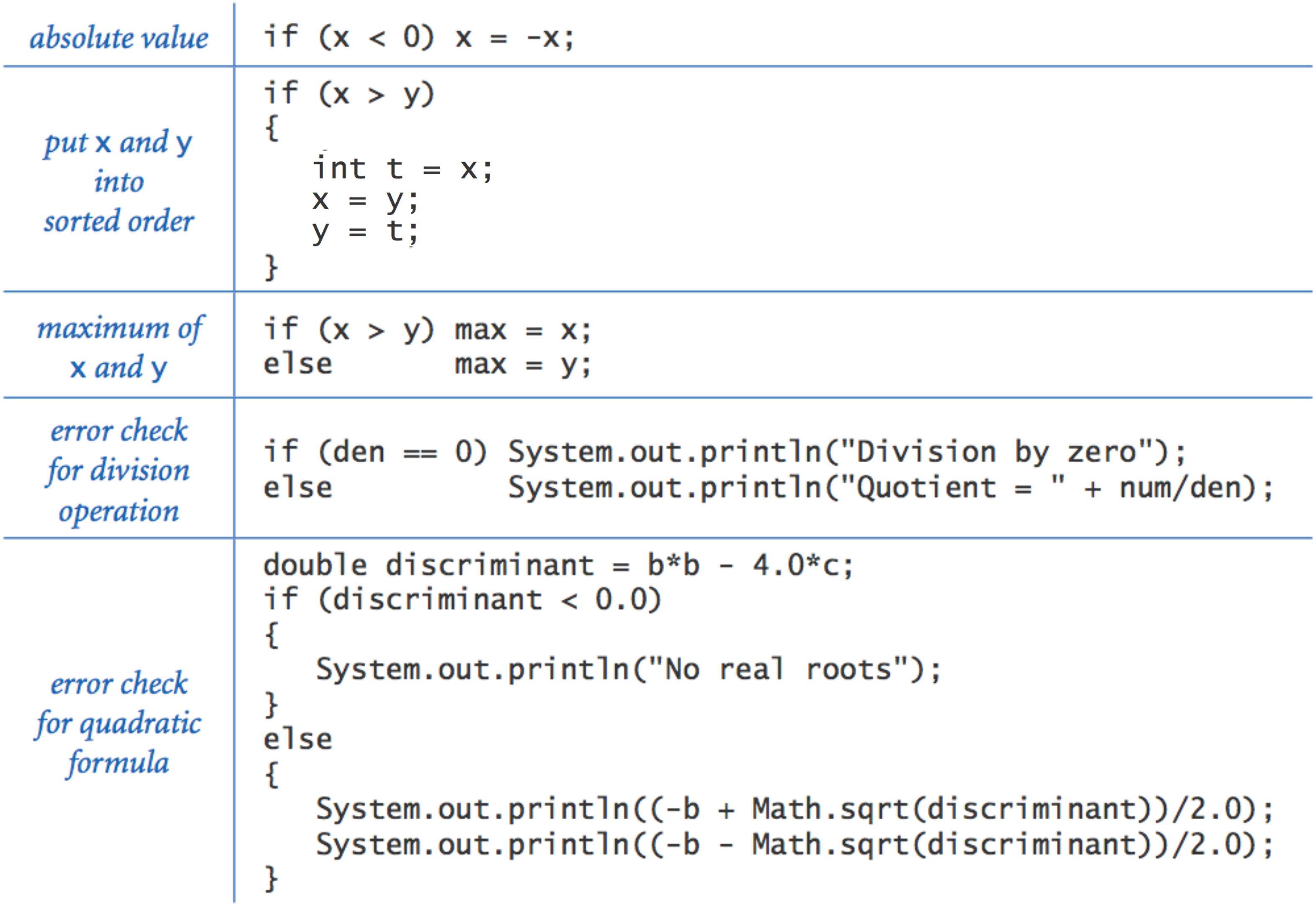
If Statement

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");
      elseMath.random() < 0.5) System.out.println("Tails");
   }
}
```

```
% java Flip
Heads

% java Flip
Heads

% java Flip
Tails

% java Flip
Heads
```

If Statement Examples



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

put x *and* y *into sorted order*	`if (x > y)`
	`{`
	`   int t = x;`
	`   x = y;`
	`   y = t;`
	`}`

maximum of x *and* y	`if (x > y) max = x;`
	`else       max = y;`

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

error check for quadratic formula	`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);`
	`}`

The While Loop

While Loop

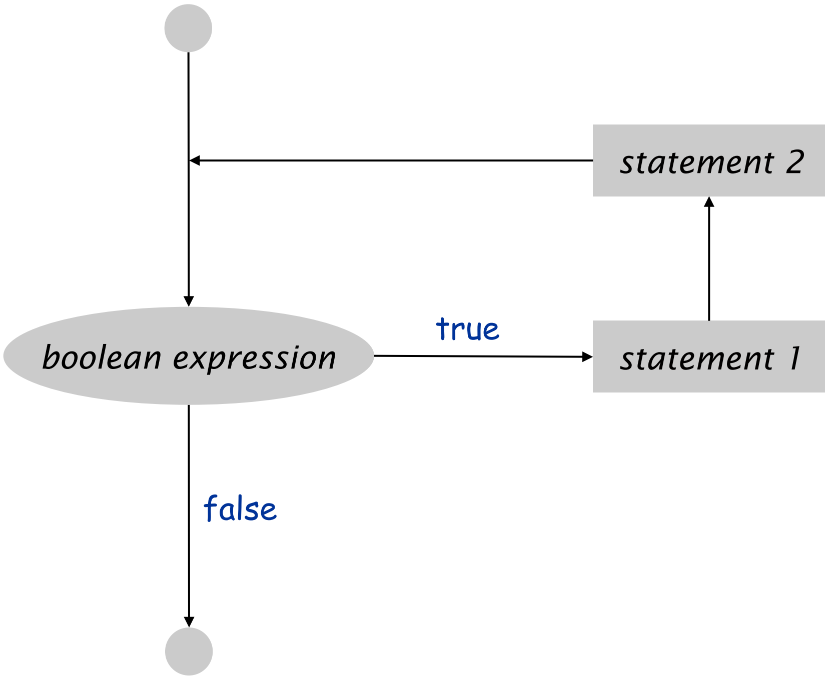
The `while` loop. A common repetition structure.

- Evaluate a `boolean` expression.
- If `true`, execute some statements.
- Repeat.

loop continuation condition

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

loop body



While Loop: 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(i + " " + 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

```
0 1
1 2
2 4
3 8
4 16
5 32
6 64
```

N = 6

Click for demo

Powers of Two

```
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(i + " " + v);
         i = i + 1;
         v = 2 * v;
      }
   }
}
```

print i and ith power of two

```
% java PowersOfTwo 3
0 1
1 2
2 4
3 8

% java PowersOfTwo 6
0 1
1 2
2 4
3 8
4 16
5 32
6 64
```

While Loop Challenge

Q. Anything wrong with the following code for printing powers of 2?

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

While Loops: Square Root

Goal. Implement `Math.sqrt()`.

```
% java Sqrt 2.0
1.414213562373095
```

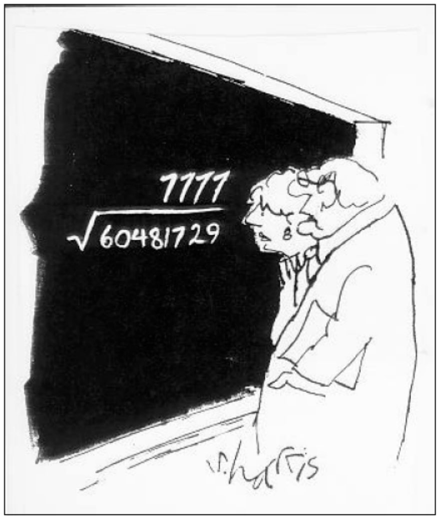
15 decimal digits of accuracy in 5 iterations

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.

$$
\begin{array}{lcccl}
t_0 & & & = & 2.0 \\
t_1 & = & \frac{1}{2}(t_0 + \frac{2}{t_0}) & = & 1.5 \\
t_2 & = & \frac{1}{2}(t_1 + \frac{2}{t_1}) & = & 1.4166666666666665 \\
t_3 & = & \frac{1}{2}(t_2 + \frac{2}{t_2}) & = & 1.4142156862745097 \\
t_4 & = & \frac{1}{2}(t_3 + \frac{2}{t_3}) & = & 1.4142135623746899 \\
t_5 & = & \frac{1}{2}(t_4 + \frac{2}{t_4}) & = & 1.414213562373095
\end{array}
$$

computing the square root of 2



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

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

While Loops: Square Root

Goal. Implement `Math.sqrt()`.

```
% java Sqrt 2.0
1.414213562373095
```

15 decimal digits of accuracy in 5 iterations

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 epsilon = 1e-15;
      double c = Double.parseDouble(args[0]);
      double t = c;
      while (Math.abs(t - c/t) > t*epsilon) {
         t = (c/t + t) / 2.0;
      }
      System.out.println(t);
   }
}
```

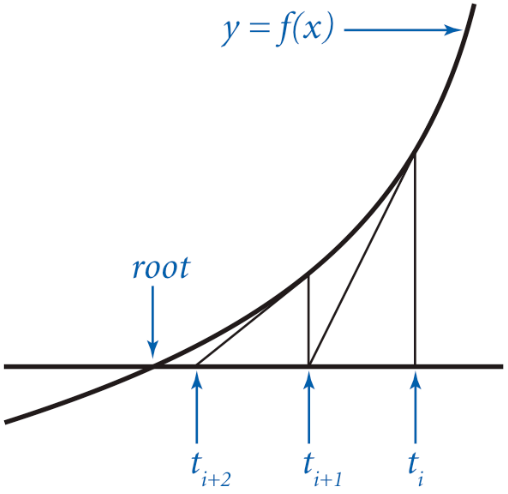
relative error tolerance

Newton-Raphson Method

Square root method explained.

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

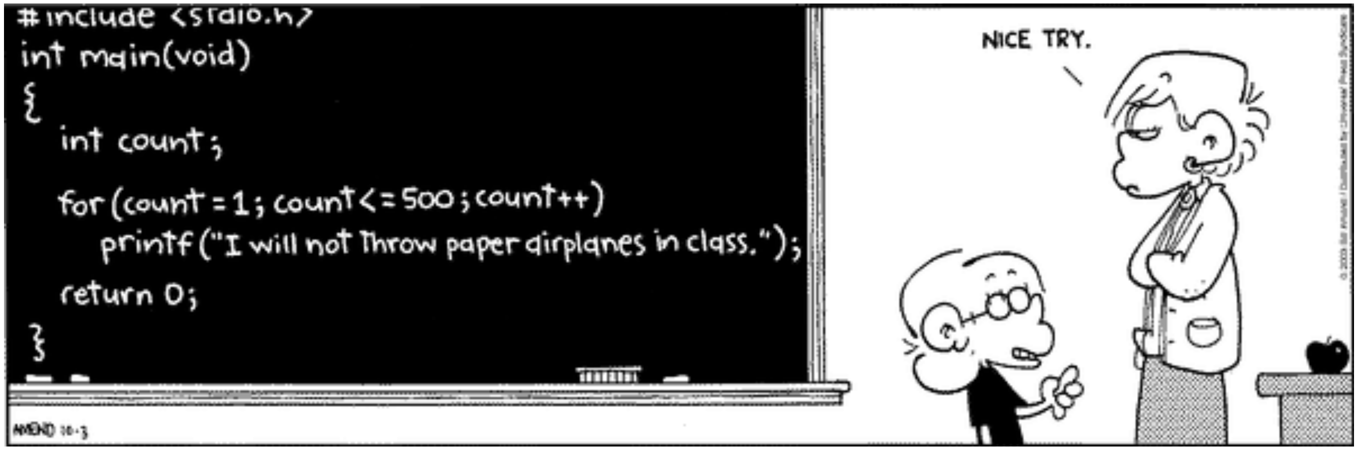
f(x) = x^2 - c to compute √c



$$t_{i+1} = t_i - \frac{f(t_i)}{f'(t_i)}$$

Technical conditions. f(x) must be smooth; t_0 must be good estimate.

The For Loop



```
#include <stdio.h>
int main(void)
{
   int count;

   for (count = 1; count <= 500; count++)
      printf("I will not throw paper airplanes in class.");
   return 0;
}
```

Copyright 2004, FoxTrot by Bill Amend

www.ucomics.com/foxtrot/2003/10/03

For Loops

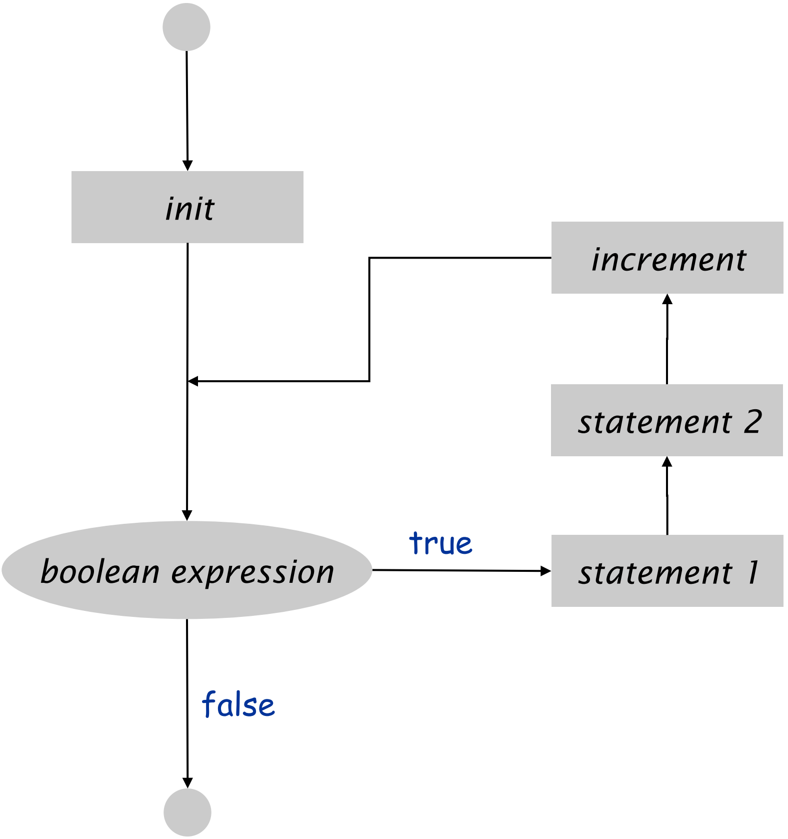
The `for` loop. Another common repetition structure.

- Execute initialization statement.
- Evaluate a `boolean` expression.
- If `true`, execute some statements.
- And then the increment statement.
- Repeat.

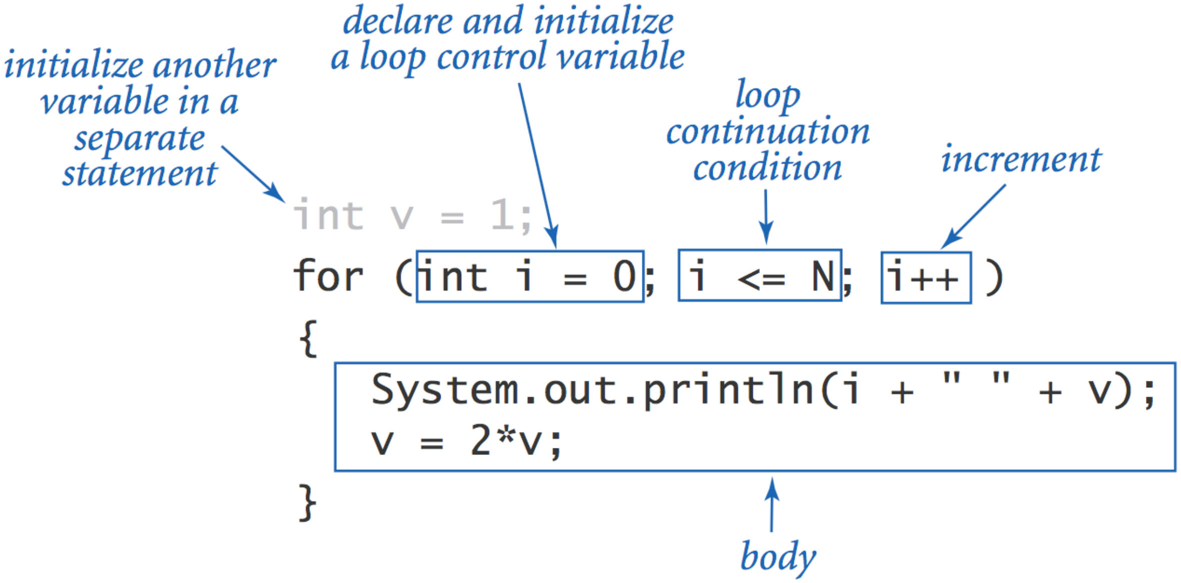
loop continuation condition

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

body



Anatomy of a For Loop



initialize another variable in a separate statement

declare and initialize a loop control variable

loop continuation condition

increment

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

body

Q. What does it print?

A.

For Loops: Subdivisions of a Ruler

Create subdivision of a ruler.

- Initialize `ruler` to `" "`.
- For each value `i` from `1` to `N`:
  sandwich two copies of `ruler` on either side of `i`.

```
public class RulerN {
   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);
   }
}
```

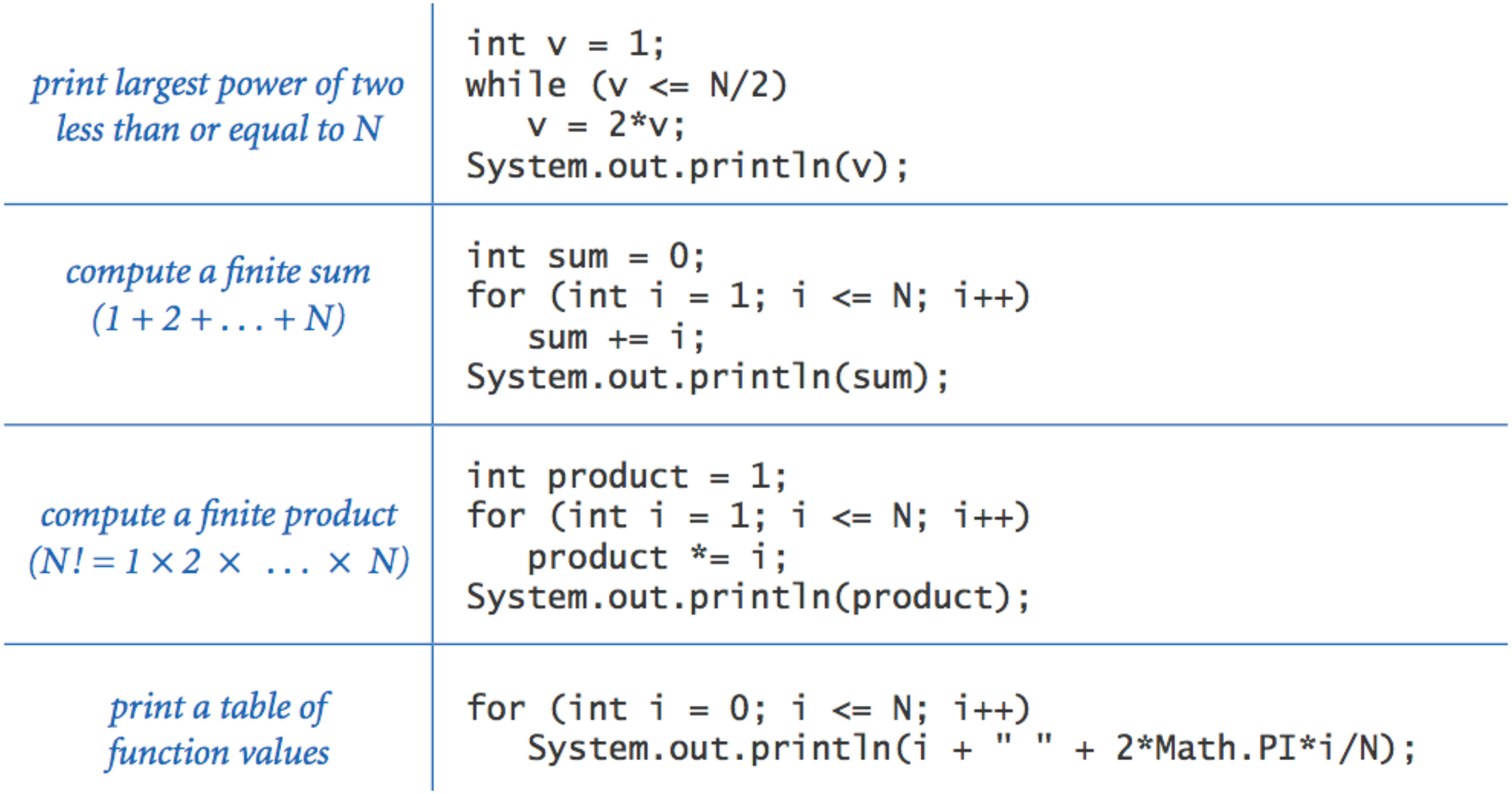
i	ruler
	" "
1	" **1** "
2	" 1 **2** 1 "
3	" 1 2 1 **3** 1 2 1 "

For Loops: Subdivisions of a Ruler

```
% java RulerN 1
 1

% java RulerN 2
 1 2 1

% java RulerN 3
 1 2 1 3 1 2 1

% java RulerN 4
 1 2 1 3 1 2 1 4 1 2 1 3 1 2 1

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

Observation. Loops can produce a huge amount of output!

Loop Examples



print largest power of two less than or equal to N	`int v = 1;`
	`while (v <= N/2)`
	`   v = 2*v;`
	`System.out.println(v);`

compute a finite sum $(1 + 2 + \ldots + N)$	`int sum = 0;`
	`for (int i = 1; i <= N; i++)`
	`   sum += i;`
	`System.out.println(sum);`

compute a finite product $(N! = 1 \times 2 \times \ldots \times N)$	`int product = 1;`
	`for (int i = 1; i <= N; i++)`
	`   product *= i;`
	`System.out.println(product);`

print a table of function values	`for (int i = 0; i <= N; i++)`
	`   System.out.println(i + " " + 2*Math.PI*i/N);`

Nesting

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

```
double rate;
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 if (income < 311950) rate = 0.35;
```

graduated income tax calculation

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 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 on p. 75.]

Nested If Statement Challenge

Q. What's wrong with the following for income tax calculation?

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

```
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;
```

wrong graduated income tax calculation

Monte Carlo Simulation

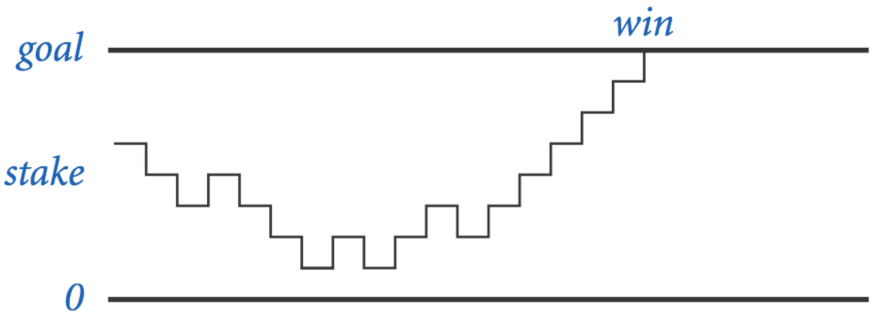
Gambler's Ruin

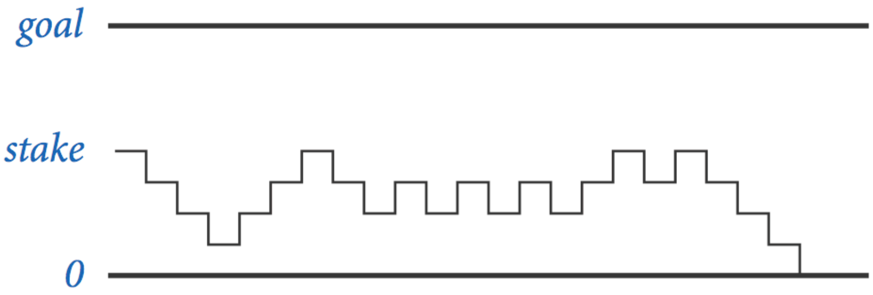
Gambler's ruin. Gambler starts with \$stake and places \$1 fair bets until going broke or reaching \$goal.

- 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.





Gambler's Ruin

```java
public class Gambler {
   public static void main(String[] args) {
      int stake  = Integer.parseInt(args[0]);
      int goal   = Integer.parseInt(args[1]);
      int T      = Integer.parseInt(args[2]);
      int wins   = 0;
      // repeat experiment T times
      for (int t = 0; t < T; t++) {
         // do one gambler's ruin experiment
         int cash = stake;
         while (cash > 0 && cash < goal) {
            // flip coin and update
            if (Math.random() < 0.5) cash++;
            else                     cash--;
         }
         if (cash == goal) wins++;
      }
      System.out.println(wins + " wins of " + T);
   }
}
```

Digression: Simulation and Analysis

stake goal T

```
% 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. [see ORF 309] Probability of winning = stake ÷ goal.

Fact. [see ORF 309] Expected number of bets = stake × desired gain.

Ex. 20% chance of turning $500 into $2500,
but expect to make one million $1 bets.

500/2500 = 20%

500 * (2500 - 500) = 1 million

Remark. Both facts can be proved mathematically; for more complex scenarios, computer simulation is often the best (only) plan of attack.

Control Flow Summary

Control flow.

- Sequence of statements that are actually executed in a program.
- Conditionals and loops: enable 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`