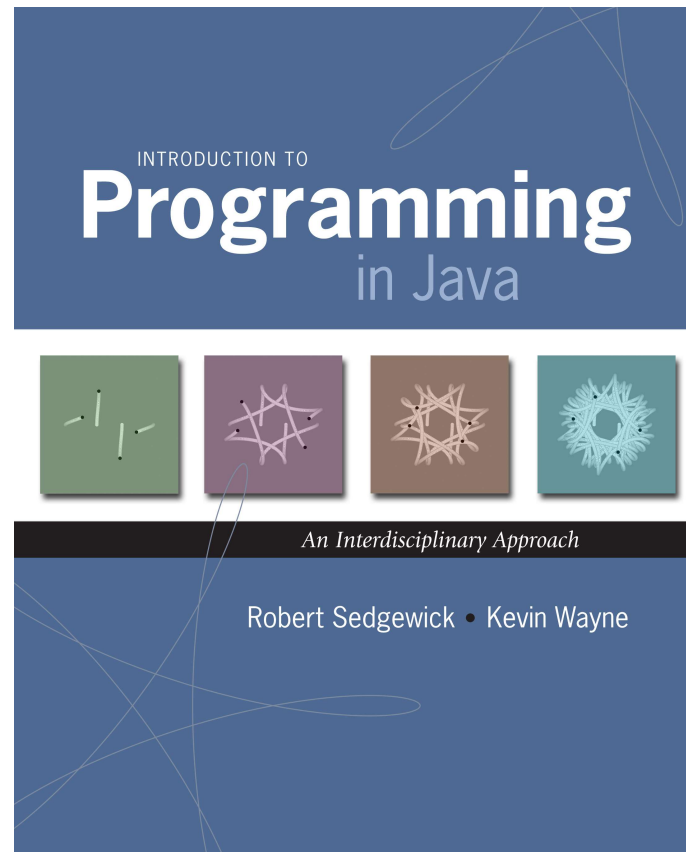
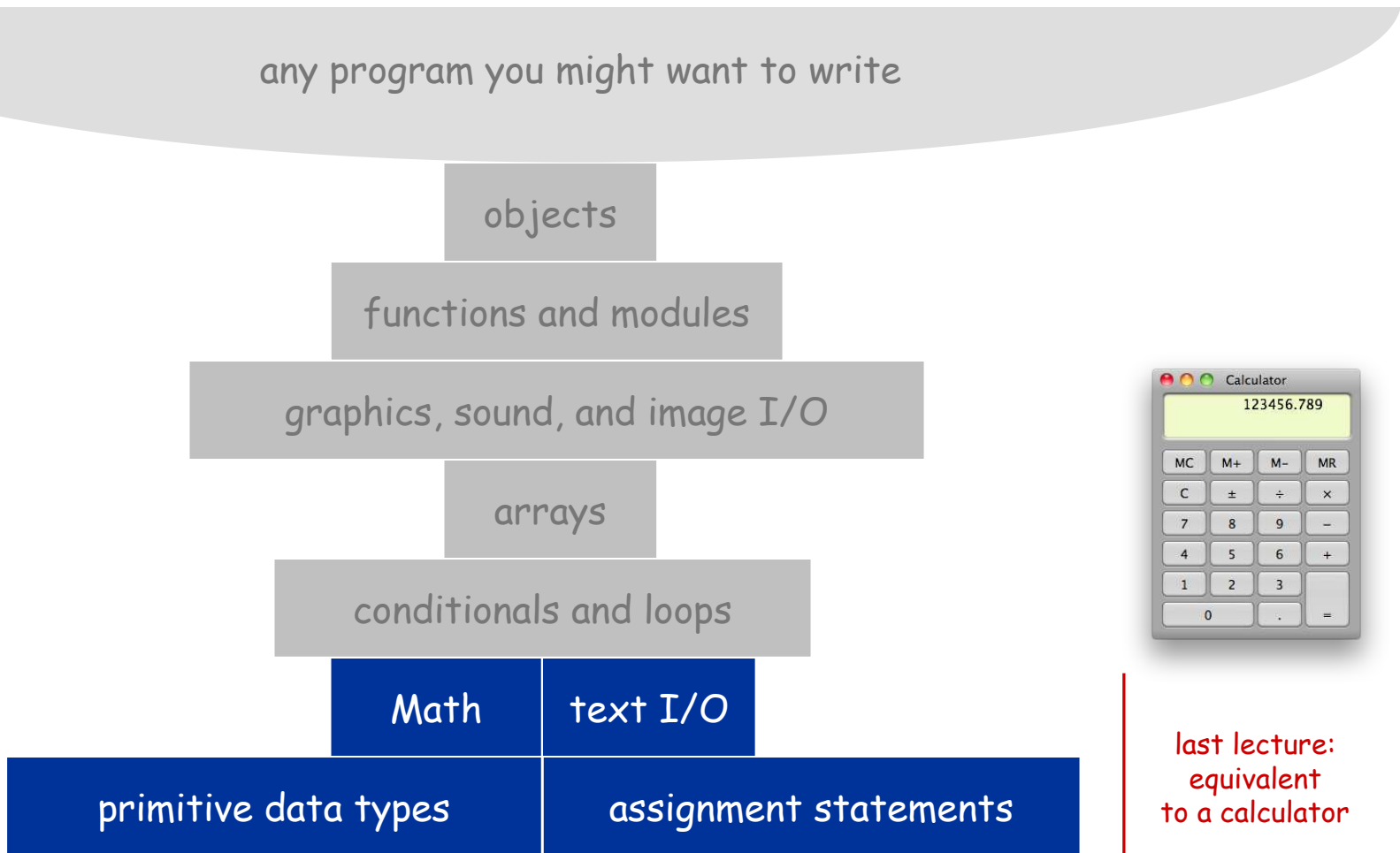


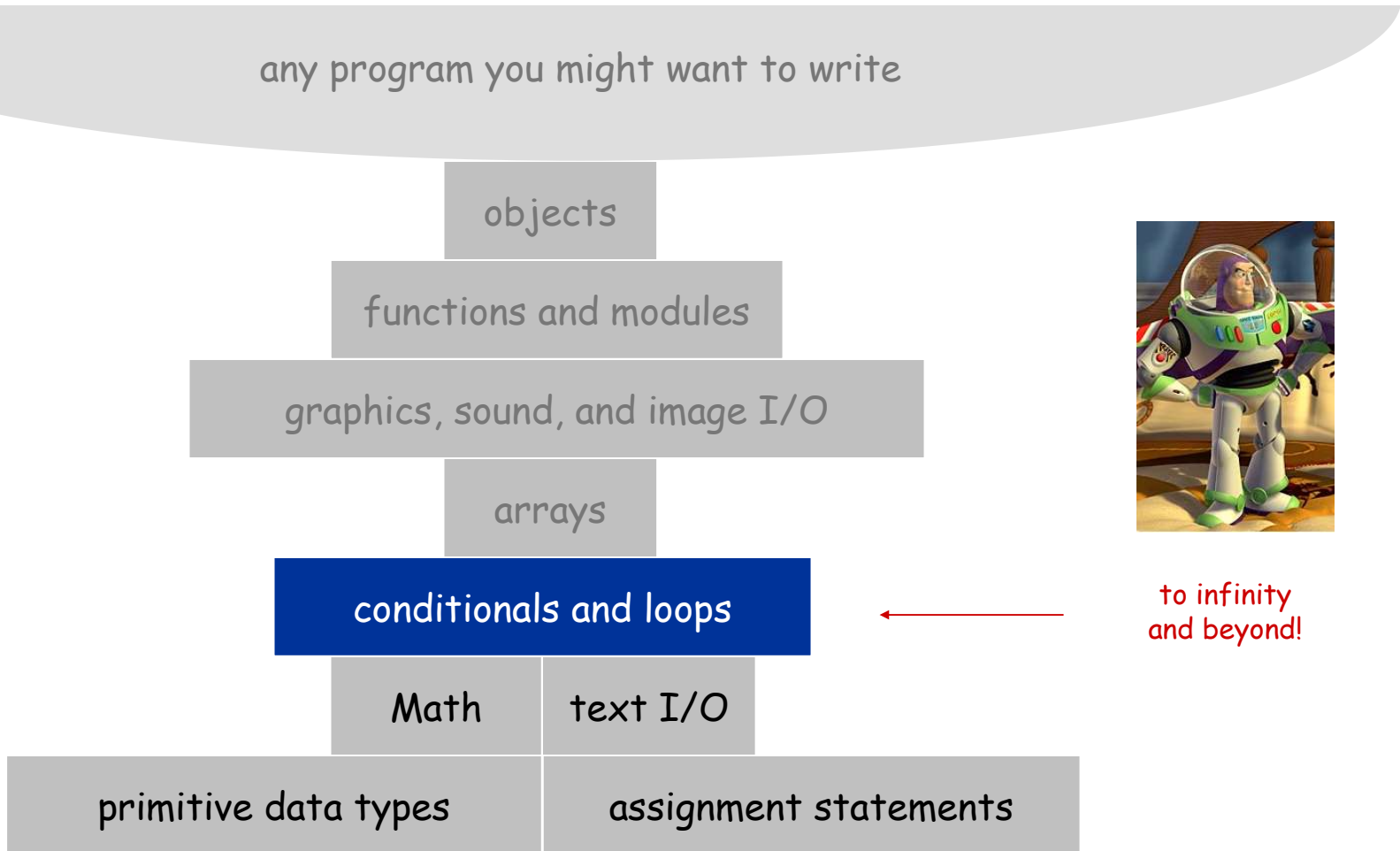
# 1.3 Conditionals and Loops



# A Foundation for Programming



# A Foundation for Programming

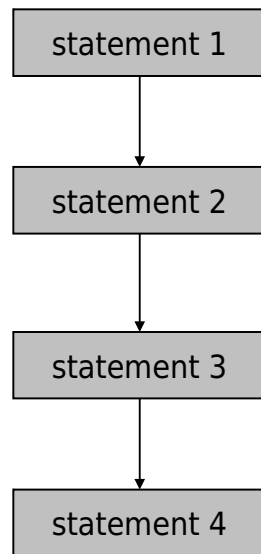


# Control Flow

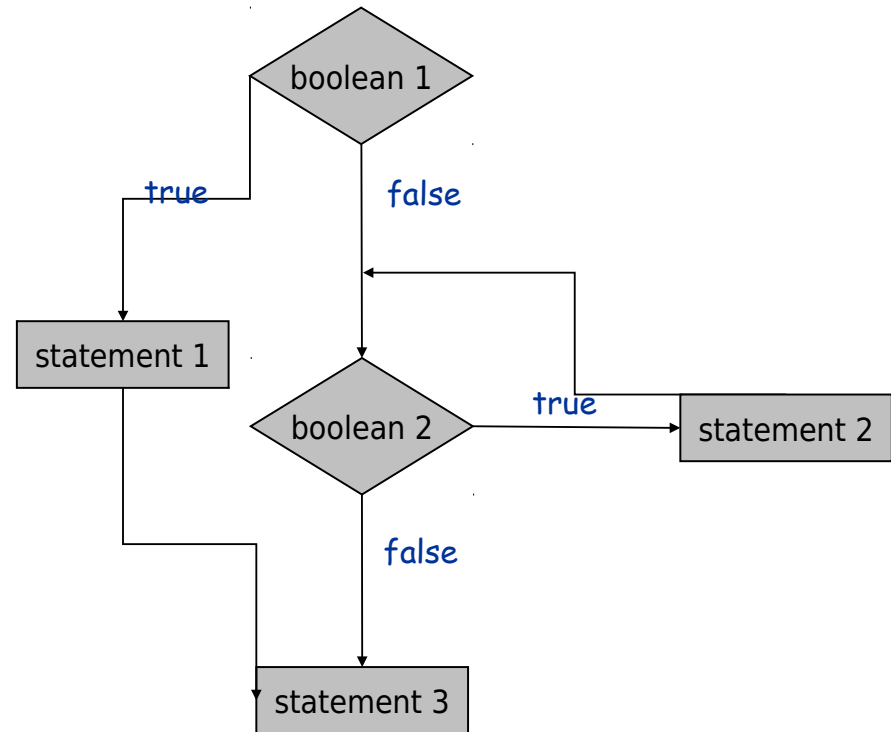
Control flow.

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.

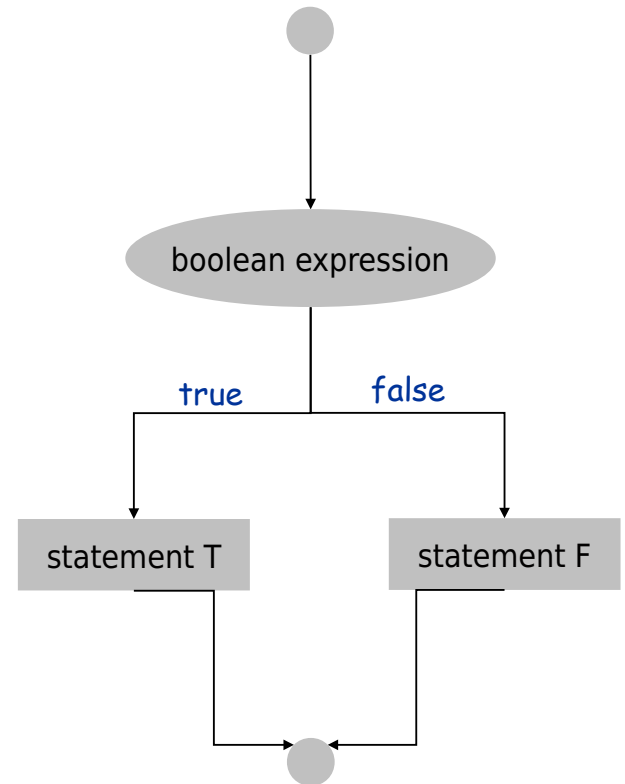
Evaluate a `boolean` expression.

If `true`, execute some statements.

If `false`, execute other statements.

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

← can be any sequence  
of statements



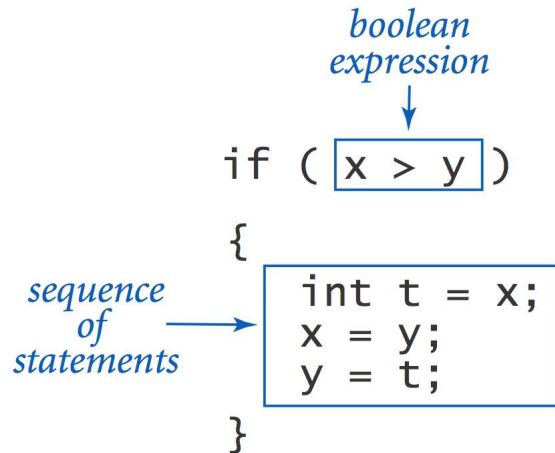
# If Statement

The `if` statement. A common branching structure.

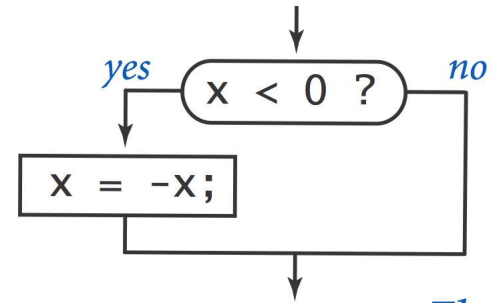
Evaluate a `boolean` expression.

If `true`, execute some statements.

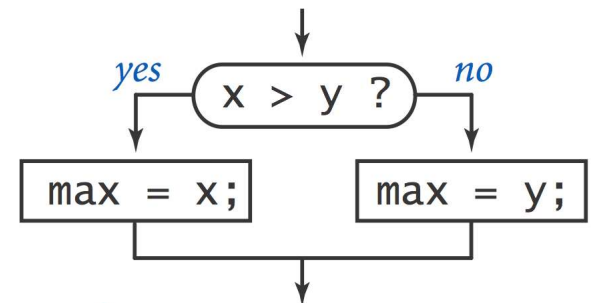
If `false`, execute other statements.



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



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



# If Statement

Ex. Take different action depending on value of variable.

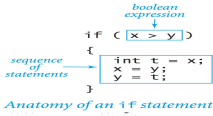
```
public class Flip {  
    public static void main(String[] args) {  
        if (Math.random() < 0.5) System.out.println("Heads");  
        else System.out.println("Tails");  
    }  
}
```



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



# If Statement Examples

<i>absolute value</i>	<code>if (x &lt; 0) x = -x;</code>
<i>put x and y into sorted order</i>	<pre> if (x &gt; y) {     x = y;     y = x; } </pre>  <p><i>Anatomy of an if statement</i></p>
<i>maximum of x and y</i>	<code>if (x &gt; y) max = x; else max = y;</code>
<i>error check for division operation</i>	<code>if (den == 0) System.out.println("Division by zero"); else System.out.println("Quotient = " + num/den);</code>
<i>error check for quadratic formula</i>	<code>double discriminant = b*b - 4.0*c; if (discriminant &lt; 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); }</code>

# The While Loop

---



# While Loop

The `while` loop. A common repetition structure.

Evaluate a `boolean` expression.

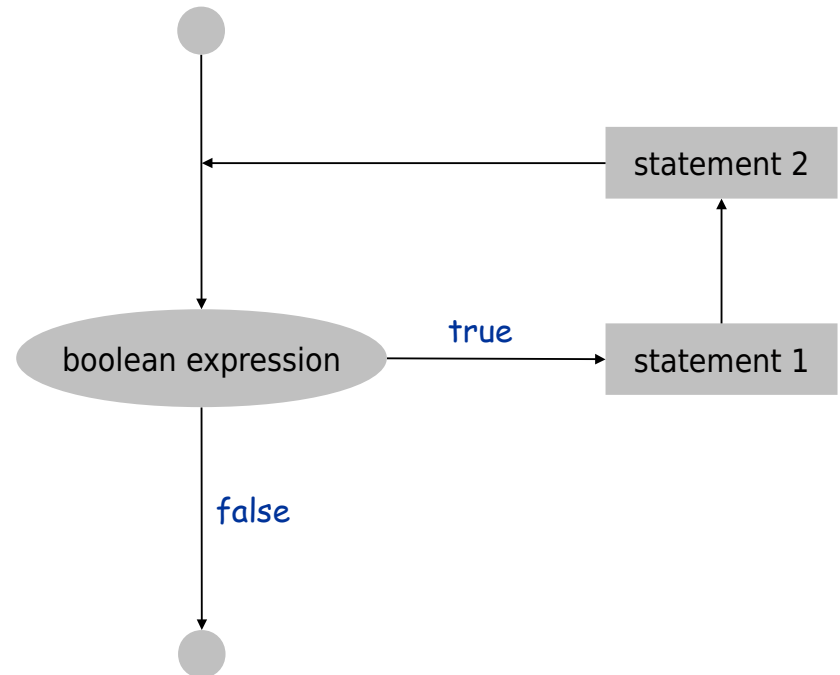
If `true`, execute some statements.

Repeat.

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

loop continuation condition

loop body



# While Loop: Powers of Two

Ex. Print powers of 2 that are  $\leq 2N$ .

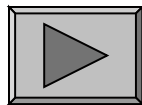
Increment  $i$  from 0 to  $N$ .

Double  $v$  each time.

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

$i$	$v$	$i \leq N$
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



Click for demo

**$N = 6$**

# 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 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;
```

A. Need curly braces around statements in while loop; otherwise it enters an infinite loop, printing "0 1".

Moment of panic. How to stop infinite loop?

# While Loops: Square Root

Goal. Implement `Math.sqrt()` .

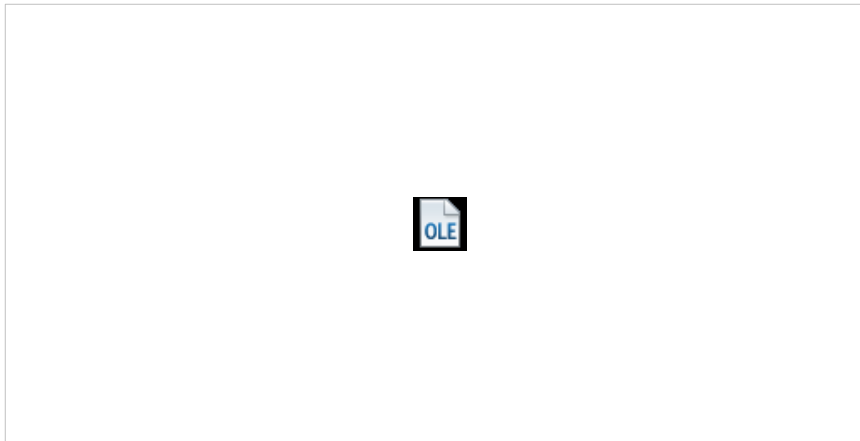
```
% java Sqrt 2.0  
1.414213562373095
```

15 decimal digits of accuracy in 5 iterations

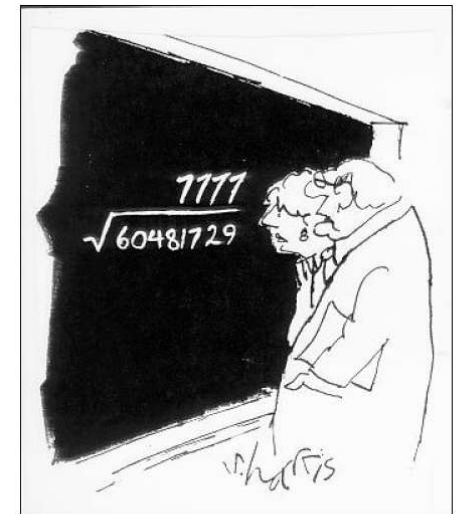
Newton-Raphson method to compute the square root of  $c$ :

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  $c / t_i$ .



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  
[www.sciencecartoonsplus.com](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  $f(x)$ .

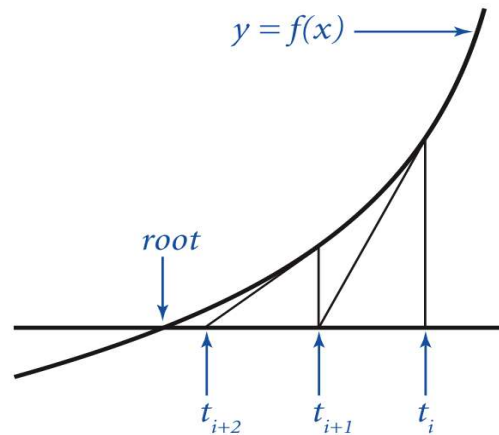
Start with estimate  $t_0$ .

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  $\sqrt{c}$

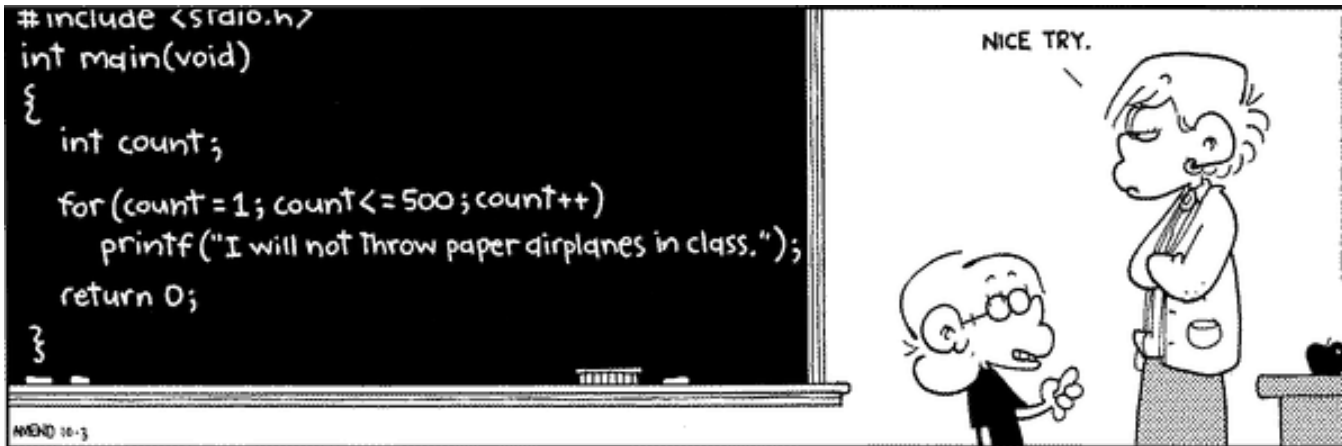


*Newton's method*

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

Technical conditions.  $f(x)$  is smooth;  $t_0$  is good estimate.

# The For Loop



Copyright 2004, FoxTrot by Bill Amend

[www.ucomics.com/foxtrot/2003/10/03](http://www.ucomics.com/foxtrot/2003/10/03)

# For Loops

The `for` loop. Another common repetition structure.

Execute initialization statement.

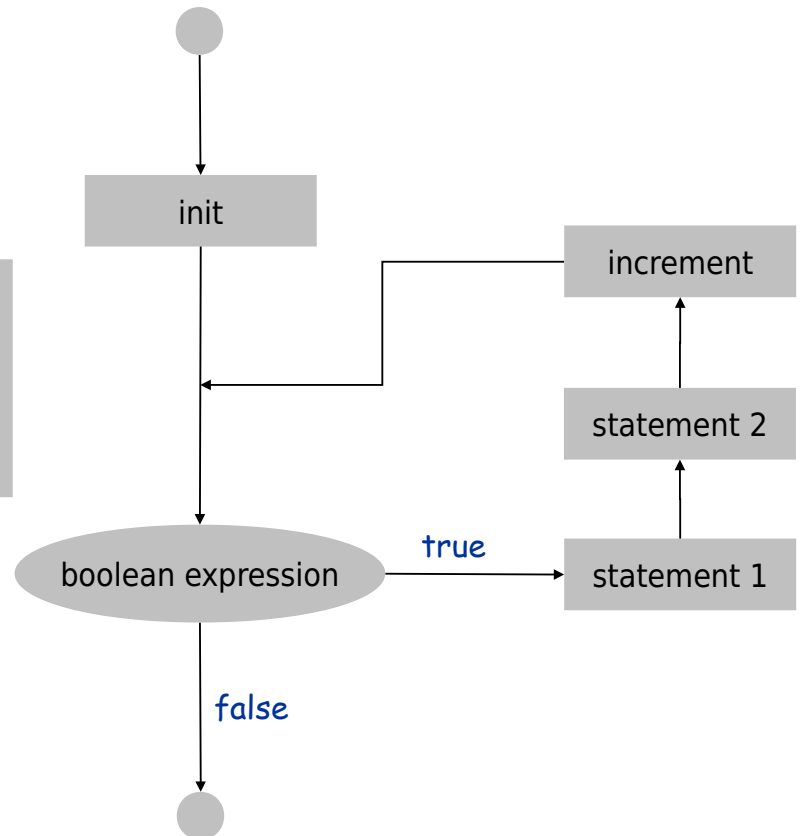
Evaluate a `boolean` expression.

If `true`, execute some statements.

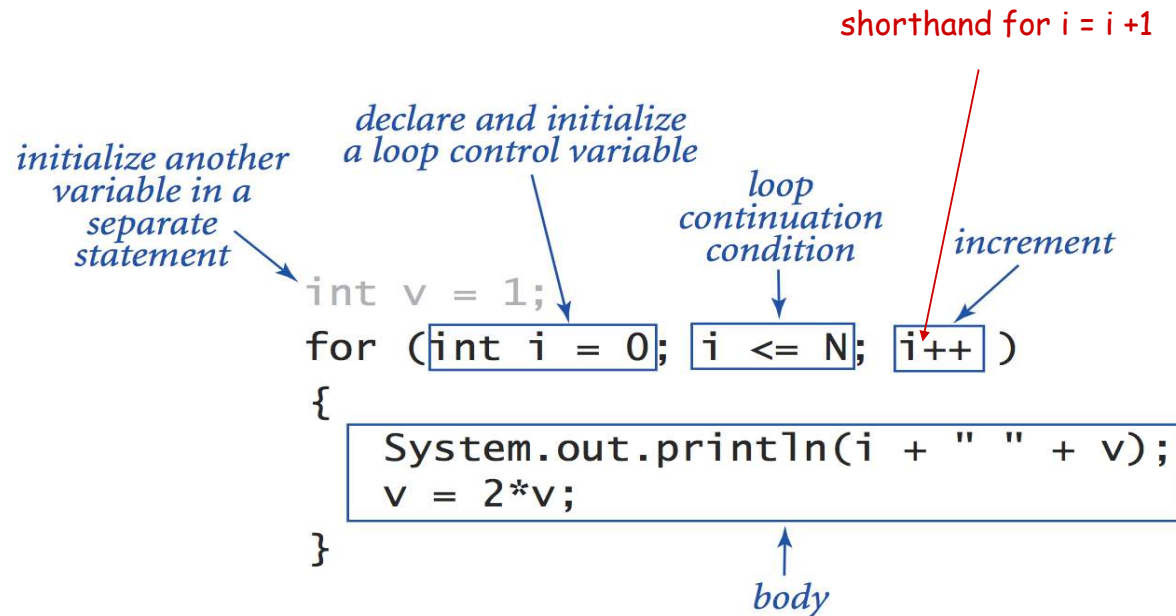
And then the `increment` statement.

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

Diagram annotations: An arrow points from "init" to the first semicolon, from "boolean expression" to the second semicolon, and from "body" to the code block.



# Anatomy of a For Loop



*Anatomy of a for loop (that prints powers of 2)*

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

<i>print largest power of two less than or equal to N</i>	<pre>int v = 1; while (v &lt;= N/2)     v = 2*v; System.out.println(v);</pre>
<i>compute a finite sum (1 + 2 + ... + N)</i>	<pre>int sum = 0; for (int i = 1; i &lt;= N; i++)     sum += i; System.out.println(sum);</pre>
<i>compute a finite product (N! = 1 × 2 × ... × N)</i>	<pre>int product = 1; for (int i = 1; i &lt;= N; i++)     product *= i; System.out.println(product);</pre>
<i>print a table of function values</i>	<pre>for (int i = 0; i &lt;= N; i++)     System.out.println(i + " " + 2*Math.PI*i/N);</pre>
<i>print the ruler function (see Program 1.2.1)</i>	<pre>String ruler = " "; for (int i = 1; i &lt;= N; i++)     ruler = ruler + i + ruler; System.out.println(ruler);</pre>



# Nesting

---



# Nesting Conditionals and Loops

**Conditionals** enable you to do one of 2n sequences of operations with n lines.

```
if (a0 > 0) System.out.print(0);  
if (a1 > 0) System.out.print(1);  
if (a2 > 0) System.out.print(2);  
if (a3 > 0) System.out.print(3);  
if (a4 > 0) System.out.print(4);  
if (a5 > 0) System.out.print(5);  
if (a6 > 0) System.out.print(6);  
if (a7 > 0) System.out.print(7);  
if (a8 > 0) System.out.print(8);  
if (a9 > 0) System.out.print(9);
```

210 = 1024 possible results, depending on input

More sophisticated programs.

Nest conditionals within conditionals.

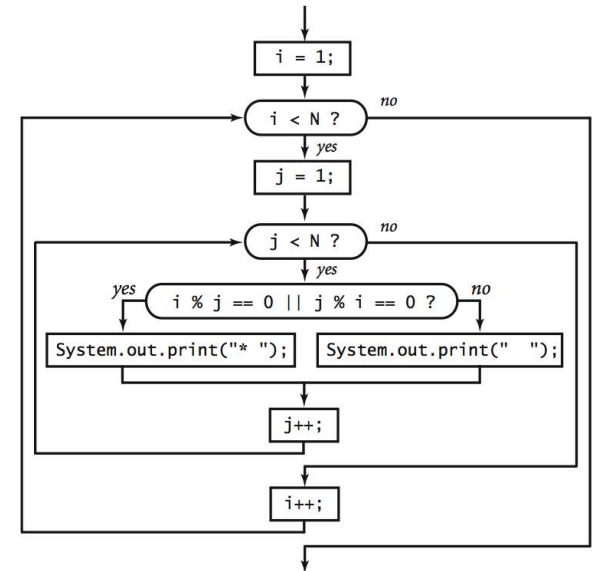
Nest loops within loops.

Nest conditionals within loops within loops.

**Loops** enable you to do an operation n times using only 2 lines of code.

```
double sum = 0.0;  
for (int i = 1; i <= 1024; i++)  
    sum = sum + 1.0 / i;
```

computes  $1/1 + 1/2 + \dots + 1/1024$



# 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           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  $\$goal$ .

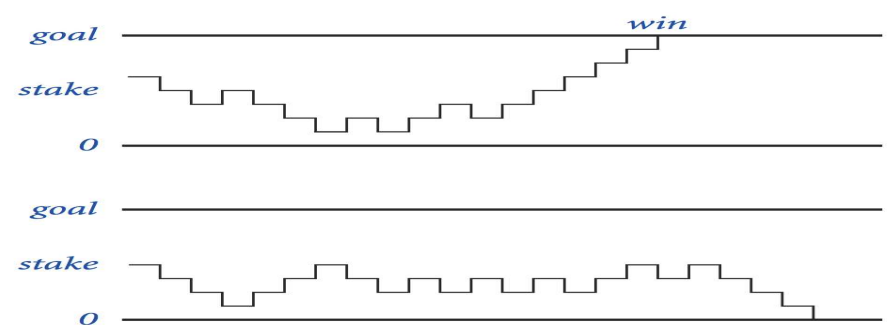
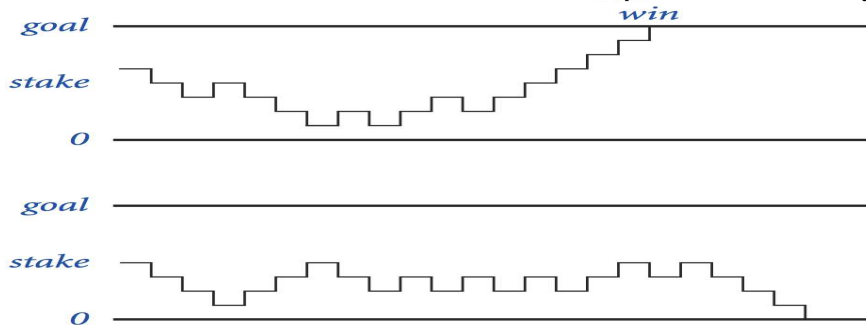
What are the chances of winning?

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

```
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.

$$500/2500 = 20\%$$

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

$$500 * (2500 - 500) = 1 \text{ million}$$

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

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	<code>if</code> <code>if-else</code>
loops	certain statements are executed repeatedly until certain conditions are met	<code>while</code> <code>for</code> <code>do-while</code>

# Extra Slides

---

# Oblivious Sorting

```
public class Sort3 {  
    public static void main(String[] args) {  
  
        int a = Integer.parseInt(args[0]);  
        int b = Integer.parseInt(args[1]);  
        int c = Integer.parseInt(args[2]);  
  
        if (b > c) { int t = b; b = c; c = t; } ← swap b and c  
        if (a > b) { int t = a; a = b; b = t; } ← swap a and b  
        if (b > c) { int t = b; b = c; c = t; } ← swap b and c  
  
        System.out.println(a + " " + b + " " + c);  
    }  
}
```

read in 3 integers  
from command-line

swap b and c

swap a and b

swap b and c

```
% java Sort3 9 8 7
```

```
7 8 9
```

```
% java Sort3 2 1 7
```

```
1 2 7
```

# Do-While Loop

The **do-while** loop. A less common repetition structure.

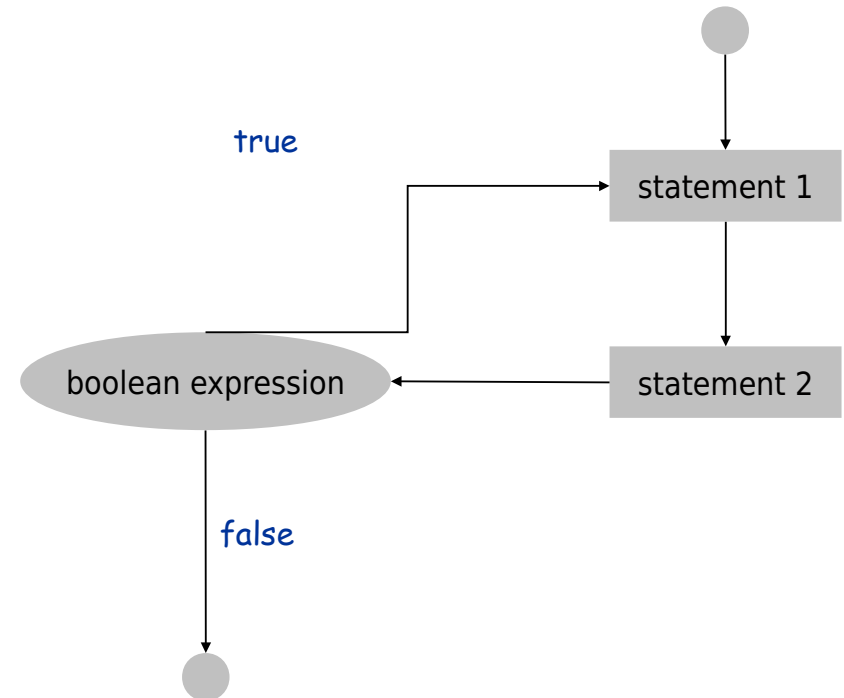
Execute sequence of statements.

Check loop-continuation condition.

Repeat.

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

do-while loop syntax



# Do-While Loop

Ex. Find a point  $(x, y)$  that is uniformly distributed in unit disc.

Pick a random point in unit square.

Check if point is also in unit disc.

Repeat.

```
do {  
    x = 2.0 * Math.random() - 1.0;  
    y = 2.0 * Math.random() - 1.0;  
} while (x*x + y*y > 1.0);
```

between -1 and 1

