Five-Minute Review

1. What is a *class hierarchy*?
2. Which graphical *coordinate system* is used by Java (and most other languages)?
3. Why is a *collage* a good methapher for GObjects?
4. What is a *CFG*? What is it useful for?
5. What is the *language* defined by a CFG?

Five-Minute Review

1. What is an *abstract data type*?
2. What is an *expression*? A *term*?
3. What is an *assignment*?
4. What is *precedence*? *Associativity*?
5. How are expressions evaluated?

Programming – Lecture 4

Statement Forms (Chapter 4)

- Repeat N-Times Pattern
- Repeat-Until-Sentinel Pattern
- `if/else`
- `? :`
- `switch`
- `while`, `do while`
- Loop-and-a-Half Pattern
- `for`
- Animation

Statement Types

Recall: Expressions, have type, are interested in *value*

Recall: Classes, Methods, Statements

Statements: units of run-time execution, are interested in *side effect*

Distinguish

- Expression statements
- Declaration statements
- Control flow statements
- Compound statements (blocks)

Expression Statements

The following *expressions*, terminated with semicolon, form *expression statements*

- *Assignment expressions*
  ```
  aValue = 8933.234;
  ```
- Any use of `++` or `--`
  ```
  aValue++;
  ```
- Method invocations
  ```
  System.out.println("Hello!");
  Math.signum(42);
  ```
- Object creation expressions
  ```
  new Bicycle();
  ```

Note: the book lists simple statements, defined as "expression ;", instead of listing expression/declaration statements. However, declarations are not expressions, and not all expressions (such as "17") lead to valid statements.

`if` Statement

```
if (condition) {
    statements to be executed if the condition is true
}

if (condition) {
    statements to be executed if the condition is true
} else {
    statements to be executed if the condition is false
}
```

Cascading `if` Statement

```
if (condition_1) {
        statements_1
} else if (condition_2) {
        statements_2
. . . more else/if conditions . . .
} else {
        statements_else
}
```

Dangling Else

```
if (x > 0)
  if (doprint)
    println(“x positive”);
else
  if (doprint)
     println(“x not positive”);
```

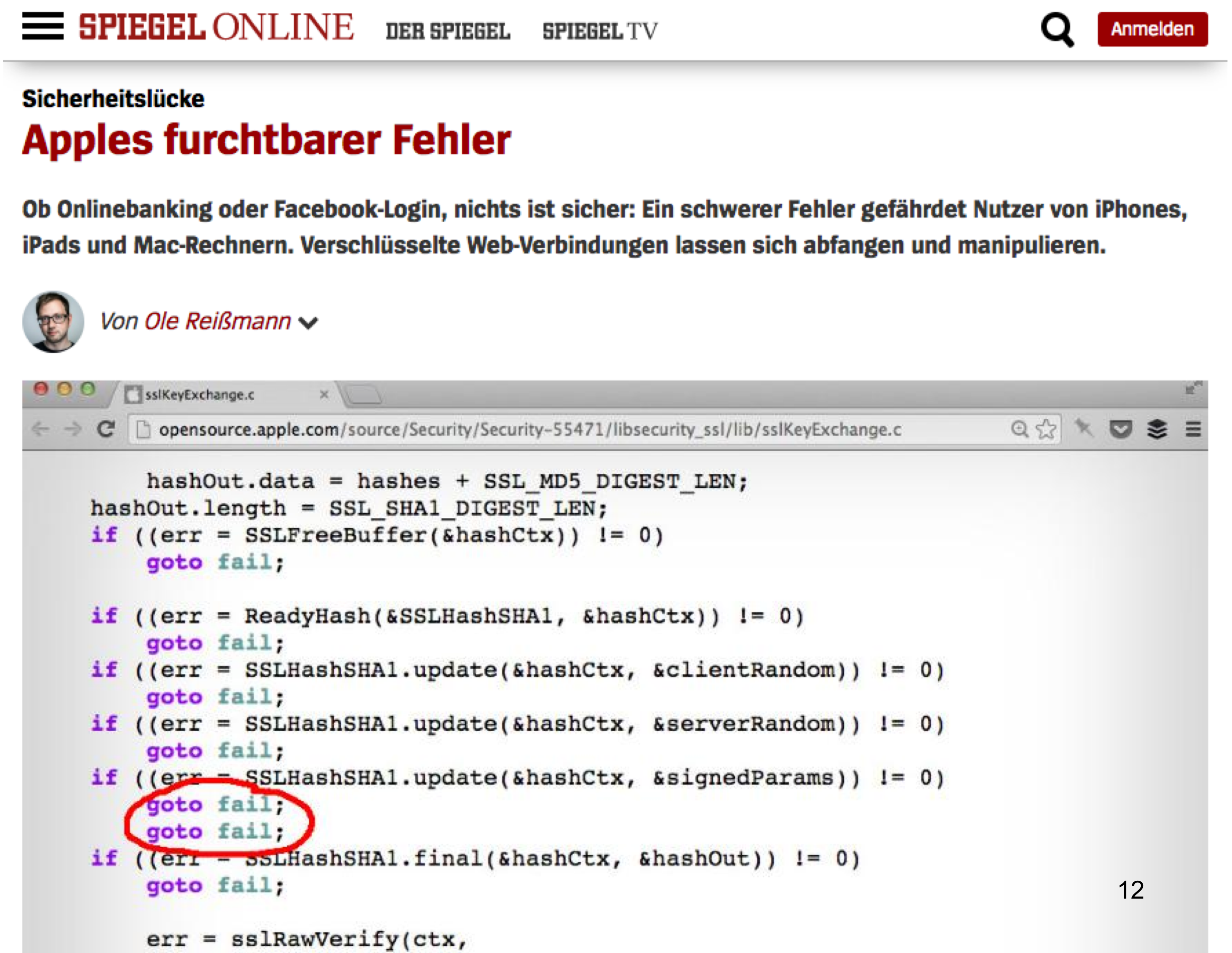
Coding Advice – Conditionals

- To avoid "dangling else":
  If an **`if`** has an **`else`** clause,
  always make **`if`** clause a block (add braces)
- In general, it is a good idea to use blocks with conditionals
- Possible exception: single statements, on same line (at least in C/C++ this is still common, in Java less so)

Bad:
```
if (x > 0)
    x++;
```

Ok:
```
if (x > 0) x++;
```

Good:
```
if (x > 0) {
    x++;
}
```



SPIEGEL ONLINE DER SPIEGEL SPIEGEL TV Anmelden

Sicherheitslücke

Apples furchtbarer Fehler

Ob Onlinebanking oder Facebook-Login, nichts ist sicher: Ein schwerer Fehler gefährdet Nutzer von iPhones, iPads und Mac-Rechnern. Verschlüsselte Web-Verbindungen lassen sich abfangen und manipulieren.

Von Ole Reißmann

sslKeyExchange.c

opensource.apple.com/source/Security/Security-55471/libsecurity_ssl/lib/sslKeyExchange.c

```
        hashOut.data = hashes + SSL_MD5_DIGEST_LEN;
    hashOut.length = SSL_SHA1_DIGEST_LEN;
    if ((err = SSLFreeBuffer(&hashCtx)) != 0)
        goto fail;

    if ((err = ReadyHash(&SSLHashSHA1, &hashCtx)) != 0)
        goto fail;
    if ((err = SSLHashSHA1.update(&hashCtx, &clientRandom)) != 0)
        goto fail;
    if ((err = SSLHashSHA1.update(&hashCtx, &serverRandom)) != 0)
        goto fail;
    if ((err = SSLHashSHA1.update(&hashCtx, &signedParams)) != 0)
        goto fail;
        goto fail;
    if ((err = SSLHashSHA1.final(&hashCtx, &hashOut)) != 0)
        goto fail;

        err = sslRawVerify(ctx,
```

`?:` Operator

condition **`?`** $expression_{\text{true}}$ **`:`** $expression_{\text{false}}$

`switch` Statement

```
switch ( expression ) {
   case v1:
      statements to be executed if expression = v1
      break;
   case v2:
      statements to be executed if expression = v2
      break;
   . . . more case clauses if needed . . .
   default:
      statements to be executed if no values match
      break;
}
```

```
public void run() {
   println("This program shows the number of days in a month.");
   int month = readInt("Enter numeric month (Jan=1): ");
   switch (month) {
      case 2:
         println("28 days (29 in leap years)");
         break;
      case 4: case 6: case 9: case 11:
         println("30 days");
         break;
      case 1: case 3: case 5: case 7: case 8: case 10: case 12:
         println("31 days");
         break;
      default:
         println("Illegal month number");
         break;
   }
}
```

Note: the Eclipse default format looks different:

- Cases not indented
- New line for each case

Coding Advice – Switch Statement

- A popular programming mistake: forgetting the **`break`** statement.
- If you really want a "non-trivial" *fall-through case*, indicate so with comment
- Always include **`default`**

```
switch (var) {
// Trivial fall-through from 1 to 2
  case 1: case 2:
    println("var is 1 or 2");
    break;
  case 3:
    println("var is 3");
    // no break here
  case 4:
    println("var is 3 or 4");
    break;
  default:
    println("var is not 1 ... 4");
    break;
}
```

Add4Integers

```
public class Add4Integers extends ConsoleProgram
{
   public void run() {
      println("This program adds four numbers.");
      int n1 = readInt("Enter n1: ");
      int n2 = readInt("Enter n2: ");
      int n3 = readInt("Enter n3: ");
      int n4 = readInt("Enter n4: ");
      int total = n1 + n2 + n3 + n4;
      println("The total is " + total + ".");
   }
}
```

Repeat-N-Times Idiom

```
for (int i = 0; i < repetitions; i++) {
    statements to be repeated
}
```

AddNIntegers

```
public class AddNIntegers extends ConsoleProgram
{
   private static final int N = 100;

   public void run() {
      println("This program adds " + N +
           " numbers.");
      int total = 0;
      for (int i = 0; i < N; i++) {
         int value = readInt(" ? ");
         total += value;
      }
      println("The total is " + total + ".");
   }
}
```

Repeat-Until-Sentinel Idiom

```
while (true) {
    prompt user and read in a value
    if (value == sentinel) {
        break;
    }
    rest of loop body
}
```

Note: some (including the author of our book) consider the braces around the **`break`** unnecessary, and in C/C++ they are commonly omitted, as in–for conciseness–most of our slide examples. However, most Java coding styles do recommend braces.

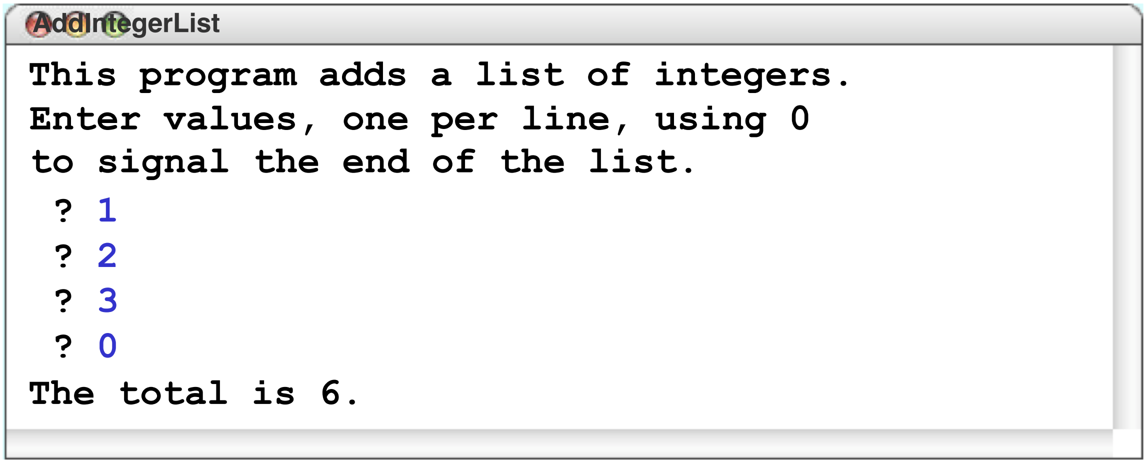
AddIntegerList

```
public class AddIntegerList extends ConsoleProgram {
   private static final int SENTINEL = 0;

   public void run() {
      println("This program adds a list of integers.");
      println("Enter values, one per line, using " +
            SENTINEL);
      println("to signal the end of the list.");
      int total = 0;
      while (true) {
         int value = readInt(" ? ");
         if (value == SENTINEL) break;
         total += value;
      }
      println("The total is " + total + ".");
   }
}
```

AddIntegerList

```
public void run() {
   println("This program adds a list of integers.");
   println("Enter values, one per line, using " + SENTINEL);
   println("to signal the end of the list.");
   int total = 0;
   while (true) {
      int value = readInt(" ? ");
      if (value == SENTINEL) break;
      total += value;
   }
   println("The total is " + total + ".");
}
```

value	total
0	6

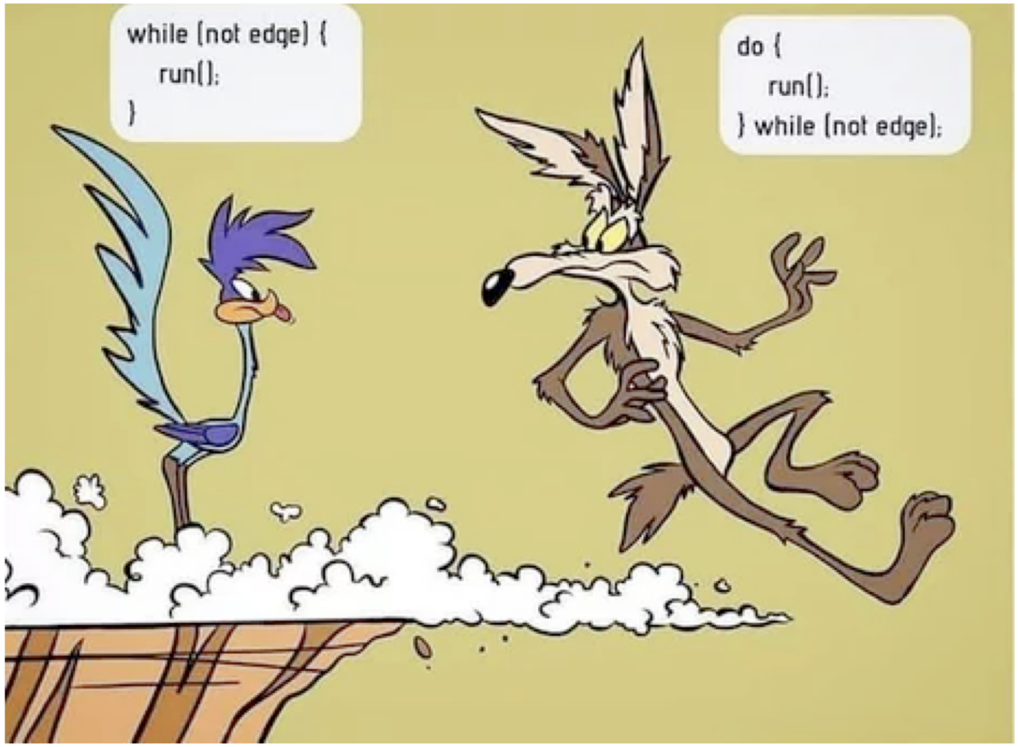


```
This program adds a list of integers.
Enter values, one per line, using 0
to signal the end of the list.
 ? 1
 ? 2
 ? 3
 ? 0
The total is 6.
```

skip simulation

`while`, `do while` Statements

while (*condition*) {
 statements to be repeated
}

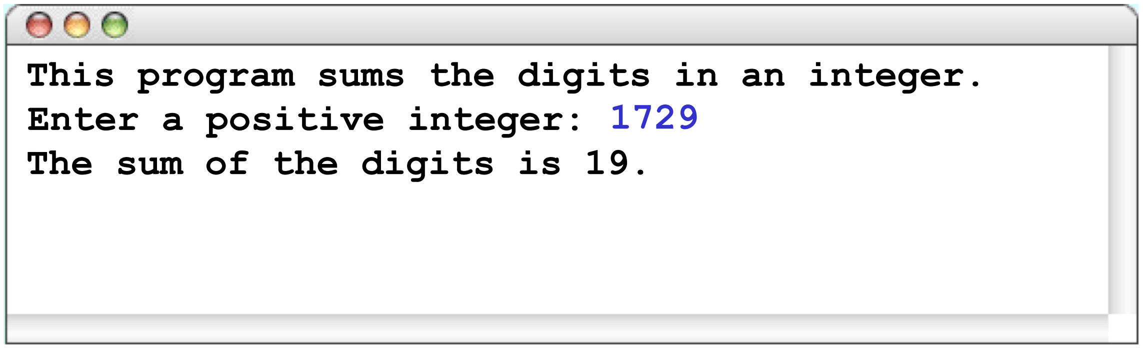


do {
 statements to be repeated
} **while** (*condition*)

DigitSum

```
public void run() {
   println("This program sums the digits in an integer.");
   int n = readInt("Enter a positive integer: ");
   int dsum = 0;
   while ( n > 0 ) {
      dsum += n % 10;
      n /= 10;
   }
   println("The sum of the digits is " + dsum);
}
```

n	dsum
0	19



```
This program sums the digits in an integer.
Enter a positive integer: 1729
The sum of the digits is 19.
```

skip simulation

Loop-and-a-Half Pattern

```
while (true) {
    computation necessary to make the test
    if (test for completion) break;
    computation for the rest of the loop cycle
}
```

Repeat-Until-Sentinel Revisited

```
while (true) {
    prompt user and read in a value
    if (value == sentinel) break;
    rest of loop body
}
```

Break Considered Harmful?

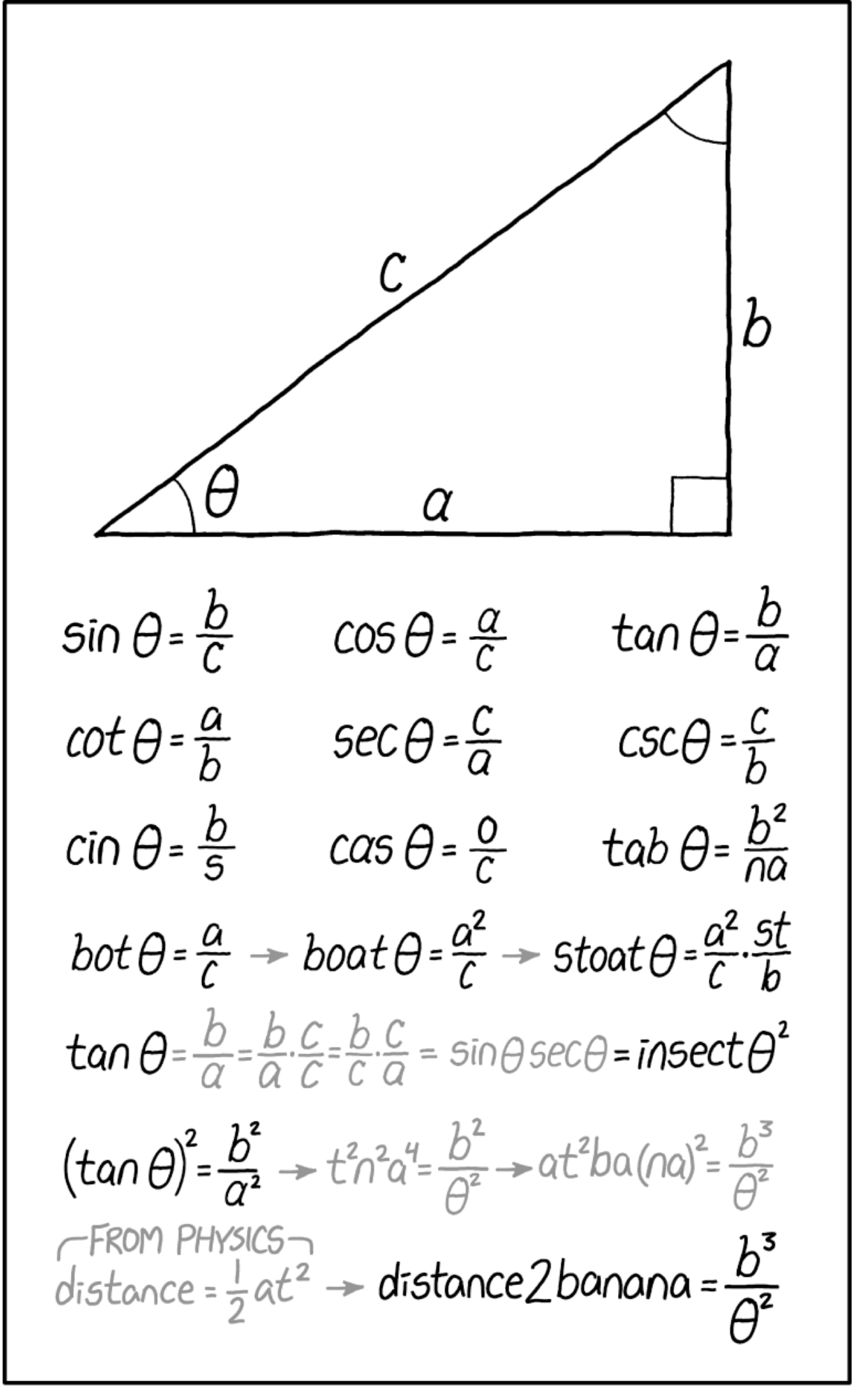
- **`break`** is disciplined cousin of C's **`goto`**
- Most seem to agree that **`goto`** should be avoided
  - Dijkstra, "Go To Statement Considered Harmful", CACM 11(3), 1968
  - Moore et al., "''' GOTO Considered Harmful' Considered Harmful" Considered Harmful?''", CACM 30(5), 1987
- Most seem to agree that **`break`** (and patterns that use it) are ok, but even that has its criticisms, e.g., difficult formalization

`for` Statement

```
for ( init ; test ; step ) {
      statements to be repeated
}
```

Equivalent to:

```
{
   init ;
   while ( test ) {
      statements to be repeated
      step ;
   }
}
```



c
b
θ
a
$\sin\theta = \frac{b}{c}$ $\cos\theta = \frac{a}{c}$ $\tan\theta = \frac{b}{a}$
$\cot\theta = \frac{a}{b}$ $\sec\theta = \frac{c}{a}$ $\csc\theta = \frac{c}{b}$
$\text{cin}\,\theta = \frac{b}{s}$ $\text{cas}\,\theta = \frac{0}{c}$ $\text{tab}\,\theta = \frac{b^2}{na}$
$\text{bot}\,\theta = \frac{a}{c} \rightarrow \text{boat}\,\theta = \frac{a^2}{c} \rightarrow \text{stoat}\,\theta = \frac{a^2}{c}\cdot\frac{st}{b}$
$\tan\theta = \frac{b}{a} = \frac{b}{a}\cdot\frac{c}{c} = \frac{b}{c}\cdot\frac{c}{a} = \sin\theta\sec\theta = \text{insect}\,\theta^2$
$(\tan\theta)^2 = \frac{b^2}{a^2} \rightarrow t^2n^2a^4 = \frac{b^2}{\theta^2} \rightarrow at^2ba(na)^2 = \frac{b^3}{\theta^2}$
FROM PHYSICS
$\text{distance} = \frac{1}{2}at^2 \rightarrow \text{distance}2\text{banana} = \frac{b^3}{\theta^2}$

KEY TRIGONOMETRIC IDENTITIES

Please visit pingo.upb.de/643250

https://xkcd.com/2070

Exercise

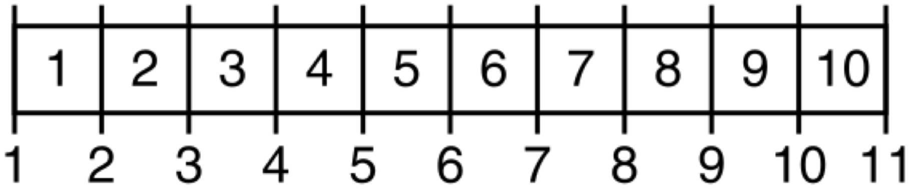
```
for (int i = 1; i <= 10; i++)

for (int i = 0; i < N; i++)

for (int n = 99; n >= 1; n -= 2)

for (int x = 1; x <= 1024; x *= 2)
```

Fence-Post Problem

In the araeostylos [style of temple] it is only necessary to preserve, in a peripteral building, twice the number of intercolumniations on the flanks that there are in front, so that the length may be twice the breadth. Those who use twice the number of columns for the length, appear to ***err****, because they thus make one intercolumniation more than should be used.*

Vitruvius, *On Architecture*

If you build a straight fence 30 meters long with posts spaced 3 meters apart, how many posts do you need?



Coding Advice: Beware of the *Off-By-One Error*

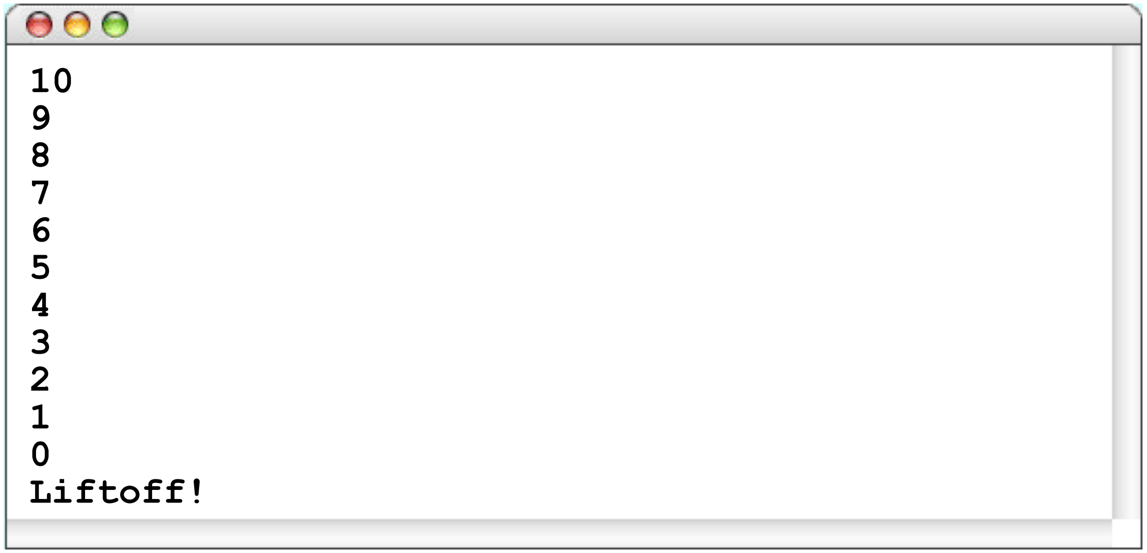
http://www.dsm.fordham.edu/~moniot/Opinions/fencepost-error-history.shtml
https://en.wikipedia.org/wiki/Off-by-one_error

Countdown

```
public void run() {
   for ( int t = 10 ; t >= 0 ; t-- ) {
      println(t);
   }
   println("Liftoff!");
}
```

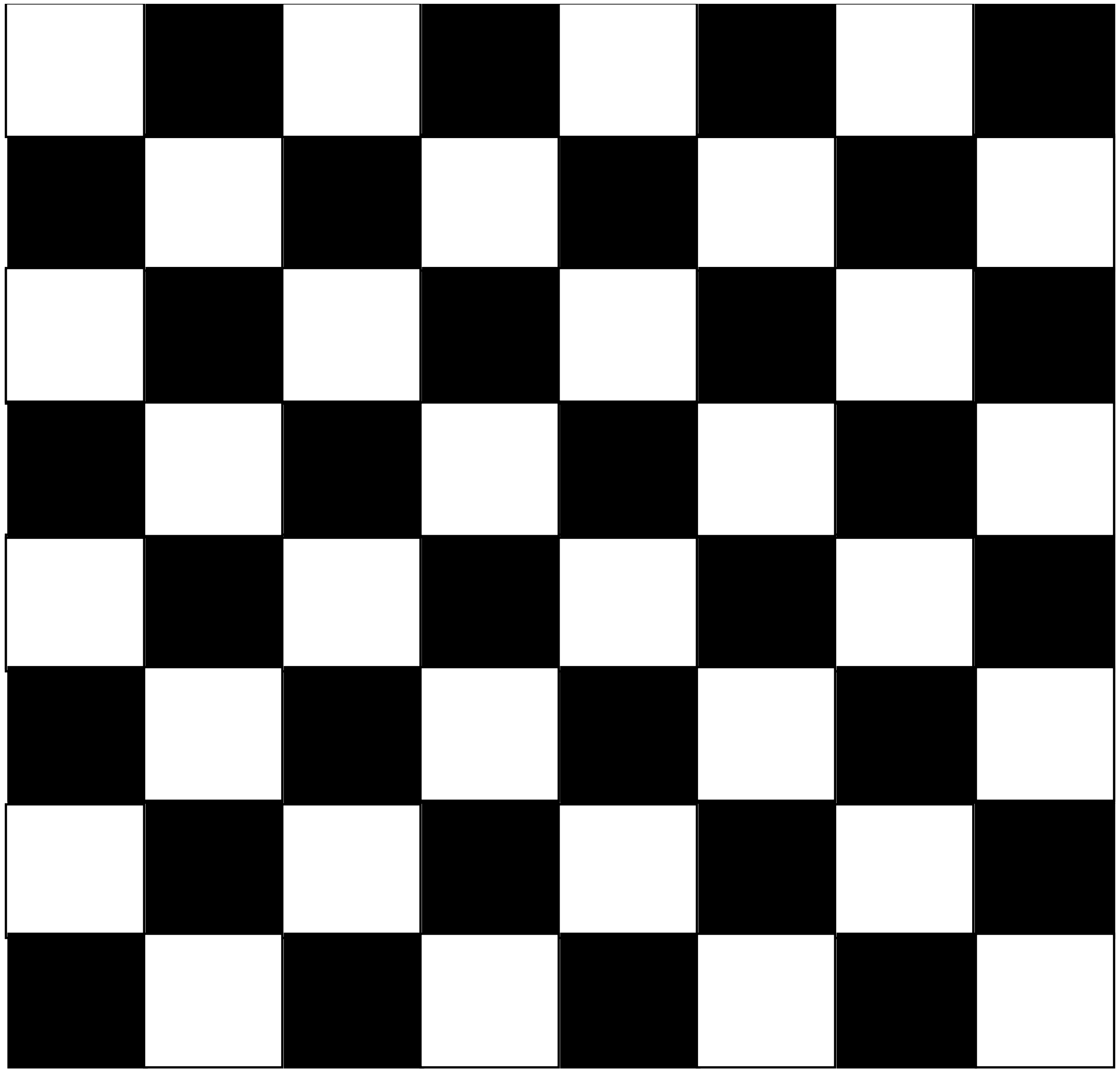
t

−1



```
10
9
8
7
6
5
4
3
2
1
0
Liftoff!
```

skip simulation



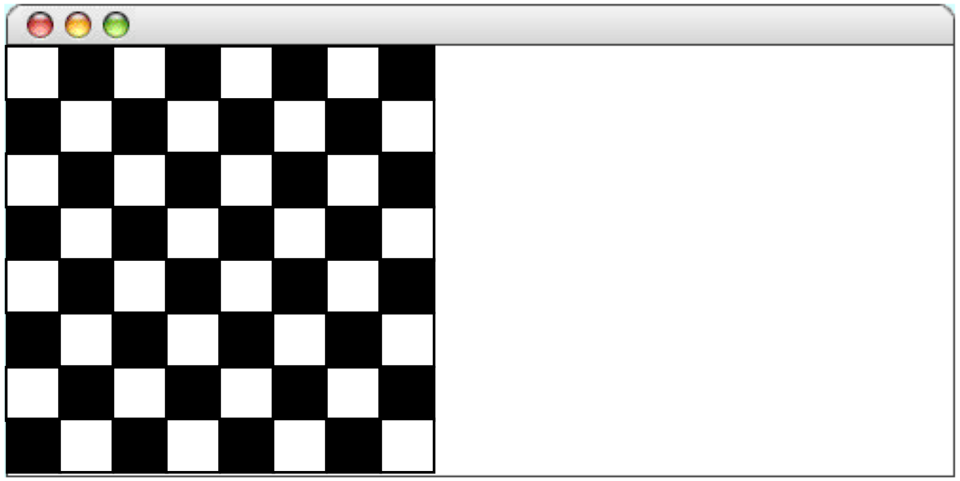
Nested `for`

```
for (int i = 0; i < N_ROWS; i++) {
    for (int j = 0; j < N_COLUMNS; j++) {
        Display the square at row i and column j.
    }
}
```

Checkerboard

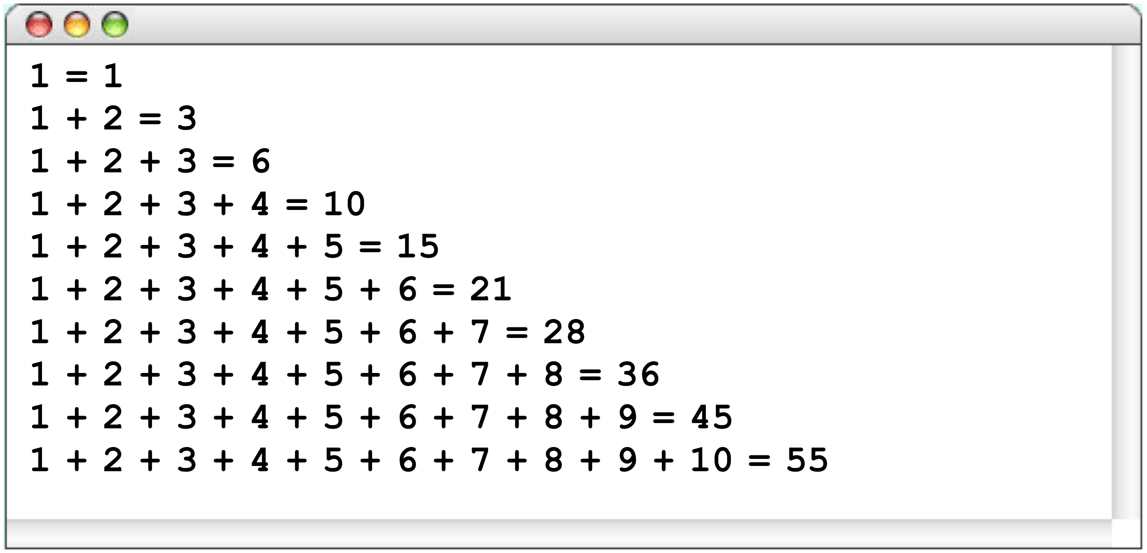
```
public void run() {
   double sqSize = (double) getHeight() / N_ROWS;
   for (int i = 0; i < N_ROWS; i++) {
      for (int j = 0; j < N_COLUMNS; j++) {
         double x = j * sqSize;
         double y = i * sqSize;
         GRect sq = new GRect(x, y, sqSize, sqSize);
         sq.setFilled((i + j) % 2 != 0);
         add(sq);
      }
   }
}
```

sqSize	i	j	x	y	sq
30.0	8	8	210.0	210.0	□



skip simulation

Exercise: Triangle Number Table


```
1 = 1
1 + 2 = 3
1 + 2 + 3 = 6
1 + 2 + 3 + 4 = 10
1 + 2 + 3 + 4 + 5 = 15
1 + 2 + 3 + 4 + 5 + 6 = 21
1 + 2 + 3 + 4 + 5 + 6 + 7 = 28
1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 = 36
1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9 = 45
1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9 + 10 = 55
```

TriangleTable

```
public class TriangleTable extends ConsoleProgram {

   private static final int MAX_VALUE = 10;

   public void run() {
      for (int n = 1; n <= MAX_VALUE; n++) {
         int total = 0;
         for (int i = 1; i <= n; i++) {
            if (i > 1) print(" + ");
            print(i);
            total += i;
         }
         println(" = " + total);
      }
   }
}
```

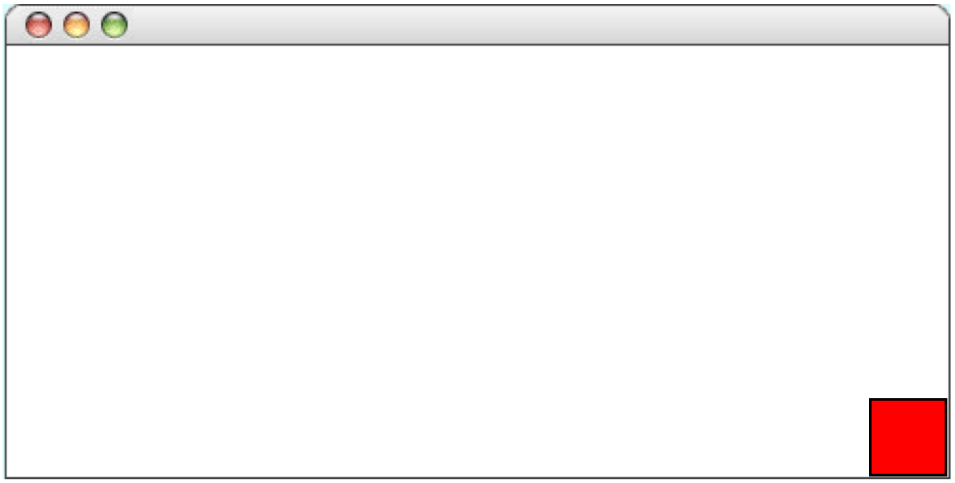
Simple Graphical Animation

```
for (int i = 0; i < N_STEPS; i++) {
    update the graphical objects by a small amount
    pause(PAUSE_TIME);
}
```

AnimatedSquare

```
public void run() {
   GRect square = new GRect(0, 0, SQUARE_SIZE, SQUARE_SIZE);
   square.setFilled(true);
   square.setFillColor(Color.RED);
   add(square);
   double dx = (getWidth() - SQUARE_SIZE) / N_STEPS;
   double dy = (getHeight() - SQUARE_SIZE) / N_STEPS;
   for (int i = 0; i < N_STEPS; i++) {
      square.move(dx, dy);
      pause(PAUSE_TIME);
   }
}
```

i	dx	dy	square
100	3.0	1.7	🟥



skip simulation

Summary

- *Expressions* (“Ausdrücke”) primarily yield a value, *statements* (“Befehle”) primarily yield a side effect
- However, most expressions (assignments, ++/--, method invocations, object creation expressions) can also be turned into *expression statements* by adding a semicolon
- Conversely, some statements (such as assignments) can also be used as expressions

Summary

- *Compound statements* (*blocks*) provide structure and scoping
- Java *control statements* include **`if/else`**, **`switch`**, **`while`**, **`for`**
- Common *idioms* are Repeat N-Times, Repeat-Until-Sentinel, and Loop-and-a-Half
- Beware of missing **`break`**s
- Beware of the *off-by-one* error