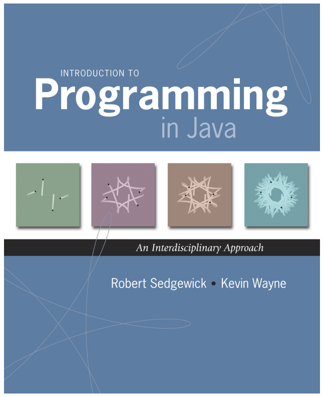
1.3 Conditionals and Loops

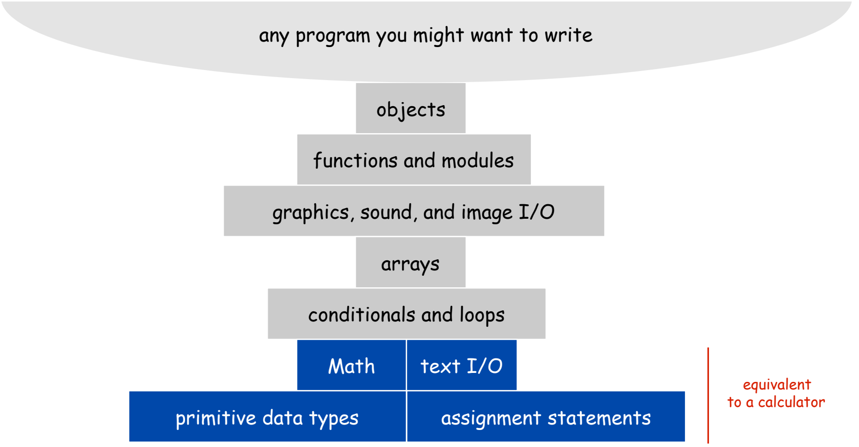


INTRODUCTION TO
Programming
in Java

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

equivalent to a calculator

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");
   }
}
```

If Statement Examples

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

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

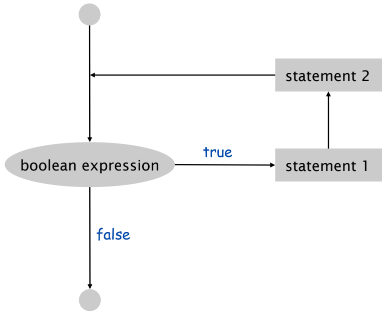
The `while` loop. A common repetition structure.

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

loop continuation condition

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

loop body



While Loops: Powers of Two

Ex. Print first `n` powers of 2.

- Increment `i` from `1` to `n`.
- Double `v` each time.

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

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

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

`n = 6`

Click for demo

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

While Loops: Square Root

Q. How might we implement `Math.sqrt()` ?

A. To compute the square root of c:

- Initialize t_0 = c.
- Repeat until t_i = c / t_i, up to desired precision:
  set t_{i+1} to be the average of t_i and c / t_i.

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

error tolerance

```
% java Sqrt 2.0
1.414213562373095
```

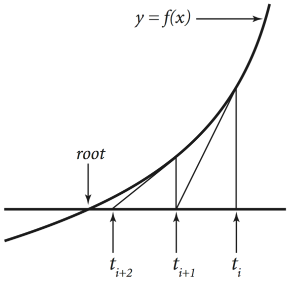
15 decimal digits of accuracy in 5 iterations

Newton-Raphson Method

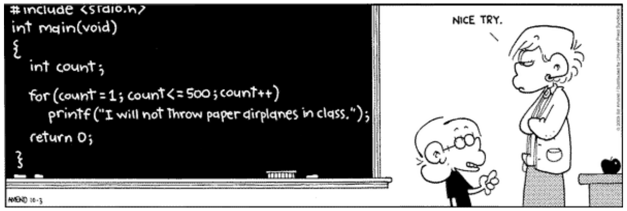
Square root method explained.

- Goal: find root of 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) = x² - c to compute √c



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;
}
NICE TRY.
AMEND 10-3

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

For Loops

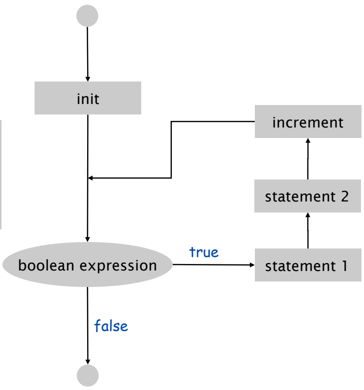
The `for` loop. Another common repetition structure.

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

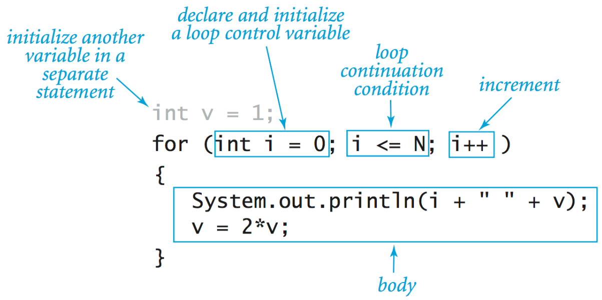
loop continuation condition

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

body



Anatomy of a For Loop



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

initialize another variable in a separate statement
declare and initialize a loop control variable
loop continuation condition
increment
body

Q. What does it print?
A.

For Loops: Subdivisions of a Ruler

Create subdivision of a ruler.

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

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

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

For Loops: Subdivisions of a Ruler

```
% java Ruler 1
 1

% java Ruler 2
 1 2 1

% java Ruler 3
 1 2 1 3 1 2 1

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

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

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

Observation. Loops can produce a huge amount of output!

Nesting

Nesting

Very Handy. **if**, **while** and **for** statements can appear inside each other.

```
for (i = 0; i < N; i++){
   for (j = i; __ ; __ ) {
      statement;
      if (something) x = 3;
      else
         while (something else) {
            statement;
            for ( __ ; __ ; __ )
               while (  ) . . .
         }
   }
}
```

Etcetera! Nesting legal and useful!

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

```
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;
```

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

Be careful when nesting if-else statements (see Q+A p. 75).

Nested If Statement Challenge

Q. Anything wrong with the following for income tax calculation?

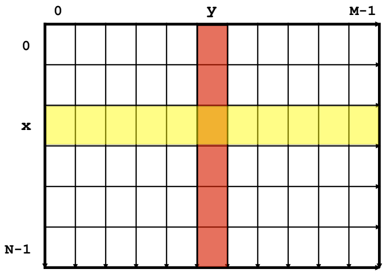
Income	Rate
0 - 47,450	22%
47,450 - 114,650	25%
114,650 - 174,700	28%
174,700 - 311,950	33%
311,950 -	35%

```
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

Nested for loops

Ex. Visit each location in a two-dimensional table.



```
for (x = 0; x < N; x++)
    for (y = 0; y < M; y++)
            Do something at entry (x,y);
```

Monte Carlo Simulation

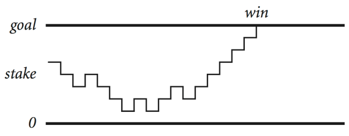
Gambler's Ruin

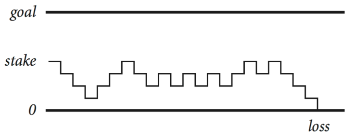
Gambler's ruin. Gambler starts with $stake and places $1 fair bets until going broke or reaching $goal.

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

Digression: Simulation and Analysis

stake goal trials

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

% java Gambler 5 25 1000
203 wins of 1000

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

after a substantial wait....

Fact. Probability of winning = stake ÷ goal.
Fact. Expected number of bets = stake × desired gain.
Ex. 20% chance of turning $500 into $2500,
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 plan of attack.

Debugging a Program

Factor. Given an integer N, compute its prime factorization.

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

i	N	*output*	i	N	*output*	i	N	*output*
2	3757208	2 2 2	9	67093		16	397	
3	469651		10	67093		17	397	
4	469651		11	67093		18	397	
5	469651		12	67093		19	397	
6	469651		13	67093	13 13	20	397	
7	469651	7	14	397				397
8	67093		15	397				

3757208/8

Application. Break RSA cryptosystem.

Debugging a Program: Syntax Errors

Syntax error. Illegal Java program.

- Compiler error messages help locate problem.
- Eventually, a file named `Factors.class`.

```
public class Factors1 {
   public static void main(String[] args) {
      long N = Long.parseLong(args[0])
      for (i = 0; i < N; i++) {
         while (N % i == 0)
            System.out.print(i + " ")
            N = N / i
      }
   }
}
```

Check if i is a factor.

As long as i is a factor, divide it out.

Compile-time error

Debugging a Program: Semantic Errors

Semantic error. Legal but wrong Java program.

- Use "`System.out.println`" method to identify problem.

```
public class Factors2 {
   public static void main(String[] args) {
      long N = Long.parseLong(args[0]);
      for (long i = 2; i < N; i++) {
         while (N % i == 0)
            System.out.print(i + " ");
            N = N / i;
      }
   }
}
```

no output (17) or infinite loop (49)

Check if i is a factor.

As long as i is a factor, divide it out.

Run-time error

Debugging a Program: Performance Errors

Performance error. Correct program but too slow.

- Use profiling to discover bottleneck.
- Devise better algorithm.

```
public class Factors3 {
   public static void main(String[] args) {
      long N = Long.parseLong(args[0]);
      for (long i = 2; i <= N; i++) {
         while (N % i == 0) {
            System.out.print(i + " ");
            N = N / i;
         }
      }
   }            too slow for large N (999,999,937)
}
```

Check if i is a factor.

As long as i is a factor, divide it out.

Performance error

Debugging a Program: Success

Fact. If N has a factor, it has one less than or equal to its square root.
Impact. Many fewer iterations of `for` loop.

```
public class Factors {
   public static void main(String[] args) {
      long N = Long.parseLong(args[0]);

      for (long i = 2; i*i <= N; i++) {
         while (N % i == 0) {
            System.out.print(i + " ");
            N = N / i;
         }
      }

      if (N > 1) System.out.println(N);
      else       System.out.println();
   }
}
```

Check if i is a factor.

As long as i is a factor, divide it out.

Corner case: biggest factor occurs once.

Program Development: Analysis

Q. How large an integer can I factor?

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

% java Factors 9201111169755555703
9201111169755555703
```

after a few minutes of computing....

largest factor

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

† estimated

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

Programming in Java

Programming in Java. [a slightly more realistic view]

1. Create the program.

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

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

4. Test it on a range of inputs.
   Program is unbearably slow for some.
   Back to step 1 to fix your errors of performance (if possible).

5. Enjoy the satisfaction of a working program!

Debugging a Program

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

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

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

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