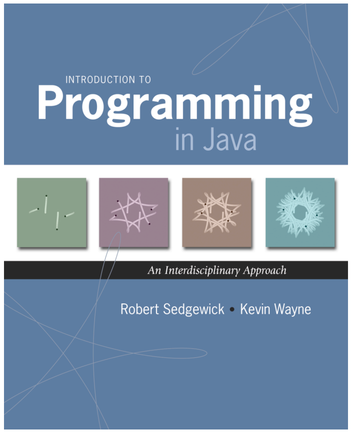
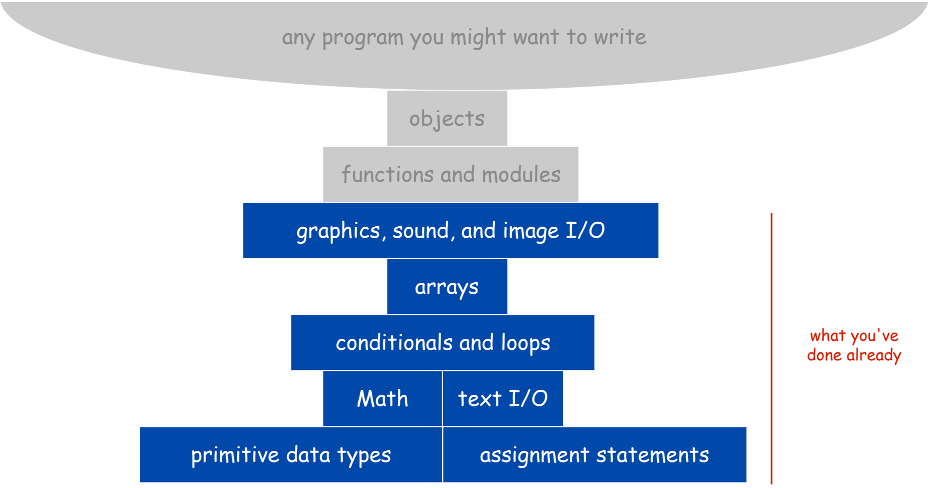
I/O (cont) and Program Development



Introduction to Programming in Java: An Interdisciplinary Approach · Robert Sedgewick and Kevin Wayne · Copyright © 2008 · February 19, 2009 10:51 AM

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

what you've done already

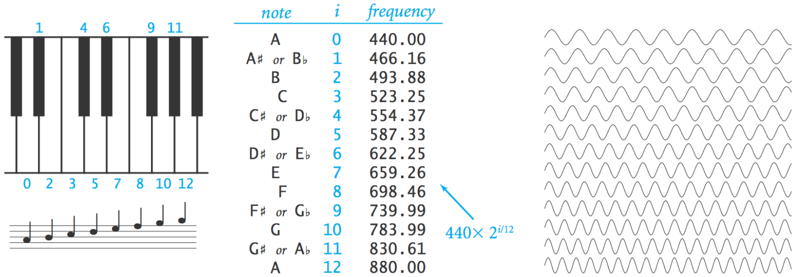
Standard Audio

Crash Course in Sound

Sound. Perception of the vibration of molecules in our eardrums.

Concert A. Sine wave, scaled to oscillate at 440Hz.

Other notes. 12 notes on chromatic scale, divided logarithmically.



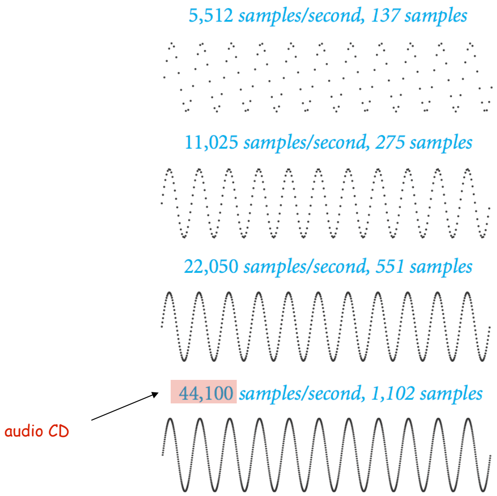
note	*i*	*frequency*
A	0	440.00
A♯ *or* B♭	1	466.16
B	2	493.88
C	3	523.25
C♯ *or* D♭	4	554.37
D	5	587.33
D♯ *or* E♭	6	622.25
E	7	659.26
F	8	698.46
F♯ *or* G♭	9	739.99
G	10	783.99
G♯ *or* A♭	11	830.61
A	12	880.00

$440 \times 2^{i/12}$

Notes, numbers, and waves

Digital Audio

Sampling. Represent curve by sampling it at regular intervals.



$$y(i) \;=\; \sin\left(\frac{2\pi \cdot i \cdot 440}{44{,}100}\right)$$

Standard Audio

Standard audio. Library for playing digital audio.

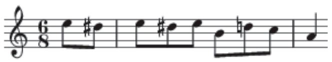
public class StdAudio	
void play(String file)	*play the given .wav file*
void play(double[] a)	*play the given sound wave*
void play(double x)	*play sample for 1/44100 second*
void save(String file, double[] a)	*save to a .wav file*
void double[] read(String file)	*read from a .wav file*

Play That Tune

Goal. Read in pitches and durations from standard input, and play using standard audio.

```
% more elise.txt
7 .125
6 .125
7 .125
6 .125
7 .125
2 .125
5 .125
3 .125
0 .25
```

```
% java PlayThatTune < elise.txt
```



Warmup: Musical Tone

Musical tone. Create a music tone of a given frequency and duration.

```
public class Tone {
   public static void main(String[] args) {
      int sps = 44100;
      double hz       = Double.parseDouble(args[0]);
      double duration = Double.parseDouble(args[1]);
      int N = (int) (sps * duration);
      double[] a = new double[N+1];
      for (int i = 0; i <= N; i++)
         a[i] = Math.sin(2 * Math.PI * i * hz / sps);
      StdAudio.play(a);
   }
}
```

$$y(i) \;=\; \sin\left(\frac{2\pi \cdot i \cdot hz}{44{,}100}\right)$$

```
% java Note 440 1.5
[ concert A for 1.5 seconds]
```

Play That Tune

Goal. Read in pitches and durations from standard input, and play using standard audio.

```
public class PlayThatTune {
   public static void main(String[] args) {
      int sps = 44100;
      while (!StdIn.isEmpty()) {
         int pitch = StdIn.readInt();
         double duration = StdIn.readDouble();
         double hz = 440 * Math.pow(2, pitch / 12.0);
         int N = (int) (sps * duration);
         double[] a = new double[N+1];
            for (int i = 0; i <= N; i++)
               a[i] = Math.sin(2 * Math.PI * i * hz / sps);
            StdAudio.play(a);
         }
      }
   }
}
```

Program Development

Ada Lovelace

Admiral Grace Murray Hopper

95% of Program Development

Program development. Creating a program and putting it to good use.
Def. A bug is a mistake in a computer program.

Programming is primarily a process of finding and fixing bugs.

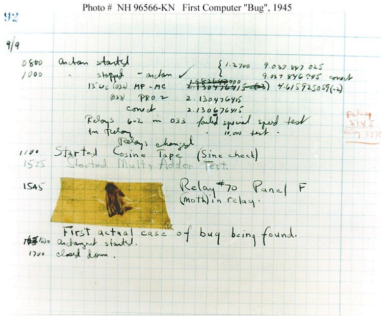


Photo # NH 96566-KN First Computer "Bug", 1945

92

9/9

0800 Anctan started
1000 " stopped - anctan ✓ {1.2700 9.037 847 025 / 9.037 846 995 conect
13" uc (032) MP - MC ~~2.130476415~~ 4.615925059(-2)
(033) PRO 2 2.130476415
conect 2.130676415
Relays 6-2 in 033 failed special speed test
In relay " 11.000 test .
Relays changed
1100 Started Cosine Tape (Sine check)
1525 Started Mult + Adder Test.
1545 Relay #70 Panel F
(moth) in relay.
First actual case of bug being found.
1630 antangent started.
1700 closed down.

Good news. Can use computer to test program.
Bad news. Cannot use computer to automatically find all bugs.

95% of Program Development

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

> *" If I had eight hours to chop down a tree, I would spend six hours sharpening an axe. "* — Abraham Lincoln

Debugging Example

Factor. Given an integer N > 1, compute its prime factorization.

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

$98 = 2 \times 7^2$

$17 = 17$

$11,111,111,111,111,111 = 2,071,723 \times 5,363,222,357$

Application. Break RSA cryptosystem (factor 200-digit numbers).

Debugging Example

Factor. Given an integer N > 1, compute its prime factorization.

Brute-force algorithm. For each putative factor i = 2, 3, 4, …, check if N is a multiple of i, and if so, divide it out.

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

Debugging: 95% of Program Development

Programming. A process of finding and fixing mistakes.

- Compiler error messages help locate syntax errors.
- Run program to find semantic and performance errors.

```
public class Factors {
   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

this program has many bugs!

Debugging: Syntax Errors

Syntax error. Illegal Java program.

- Compiler error messages help locate problem.
- Goal: no errors and a file named `Factors.class`.

```
public class Factors {
   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

      }
   }
}
```

```
% javac Factors.java
Factors.java:4: ';' expected
      for (i = 0; i < N; i++)
      ^
1 error
```

the first error

Debugging: Syntax Errors

Syntax error. Illegal Java program.

- Compiler error messages help locate problem.
- Goal: no errors and a file named `Factors.class`.

```
public class Factors {
   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
      }
   }
}
```

need to declare variable i

need terminating semicolons

syntax (compile-time) errors

Debugging: Semantic Errors

Semantic error. Legal but wrong Java program.

- Run program to identify problem.
- Add print statements if needed to produce trace.

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

```
% javac Factors.java
% java Factors
Exception in thread "main"
java.lang.ArrayIndexOutOfBoundsException: 0
        at Factors.main(Factors.java:5)
```

oops, no argument

Debugging: Semantic Errors

Semantic error. Legal but wrong Java program.

- Run program to identify problem.
- Add print statements if needed to produce trace.

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

need to start at 2 because 0 and 1 cannot be factors

```
% javac Factors.java
% java Factors 98
Exception in thread "main"
java.lang.ArithmeticException: / by zero
        at Factors.main(Factors.java:8)
```

Debugging: Semantic Errors

Semantic error. Legal but wrong Java program.

- Run program to identify problem.
- Add print statements if needed to produce trace.

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

indents do not imply braces

```
% javac Factors.java
% java Factors 13
3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3  …
```

infinite loop!

Debugging: The Beat Goes On

Success. Program factors 98 = 2 × 7^2.

- But that doesn't mean it works for all inputs.
- Add trace to find and fix (minor) problems.

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

```
% java Factors 98
2 7 7 %

% java Factors 5

% java Factors 6
2 %
```

need newline

??? no output

??? missing the 3

Debugging: The Beat Goes On

Success. Program factors 98 = 2 × 7^2.

- But that doesn't mean it works for all inputs.
- Add trace to find and fix (minor) problems.

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

```
% java Factors 5
TRACE 2 5
TRACE 3 5
TRACE 4 5

% java Factors 6
2
TRACE 2 3
```

Aha!
i loop should
go up to N

Debugging: Success?

Success. Program now seems to work.

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

```
% java Factors 5
5

% java Factors 6
2 3

% java Factors 98
2 7 7

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

Debugging: Performance Error

Performance error. Correct program, but too slow.

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

```
% java Factors 11111111
11 73 101 137

% java Factors 11111111111
21649 51329

% java Factors 11111111111111
11 239 4649 909091

% java Factors 11111111111111111
2071723 -1 -1 -1 -1 -1 -1 -1 -1
-1 -1 -1 -1 -1 -1 -1 -1 -1 …
```

Debugging: Performance Error

Performance error. Correct program, but too slow.

Solution. Improve or change underlying algorithm.

fixes performance error:
if N has a factor, it has one
less than or equal to its square root

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

```
% java Factors 98
2 7 7

% java Factors 11111111
11 73 101

% java Factors 11111111111111
11 239 4649

% java Factors 11111111111111111
2071723
```

missing last factor
(sometimes)

Debugging: Performance Error

Caveat. Optimizing your code tends to introduce bugs.
Lesson. Don't optimize until it's absolutely necessary.

need special case to print
biggest factor
(unless it occurs more than once)

```
public class Factors {
   public static void main(String[] args) {
      long N = Long.parseLong(args[0]);
      for (int i = 2; i <= N/i; i++) {
         while (N % i == 0) {
            System.out.print(i + " ");
            N = N / i;
         }
      }
      if (N > 1) System.out.println(N);
      else       System.out.println();
   }
}
```

"corner case"

```
% java Factors 11111111
11 73 101 137

% java Factors 11111111111
21649 51329

% java Factors 11111111111111
11 239 4649 909091

% java Factors 11111111111111111
2071723 5363222357
```

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 <= N/i)
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).

Debugging

Programming. A process of finding and fixing mistakes.

1. Create the program.

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

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

4. Enjoy the satisfaction of a working program!

5. Too slow? Back to step 1 to try a different algorithm.

U.S.S. Grace Murray Hopper