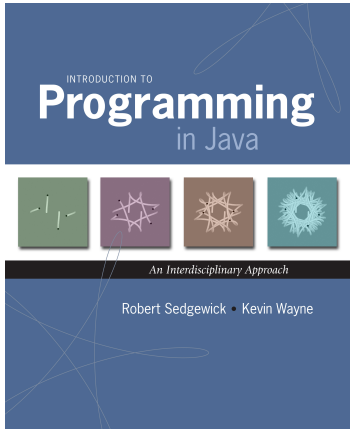
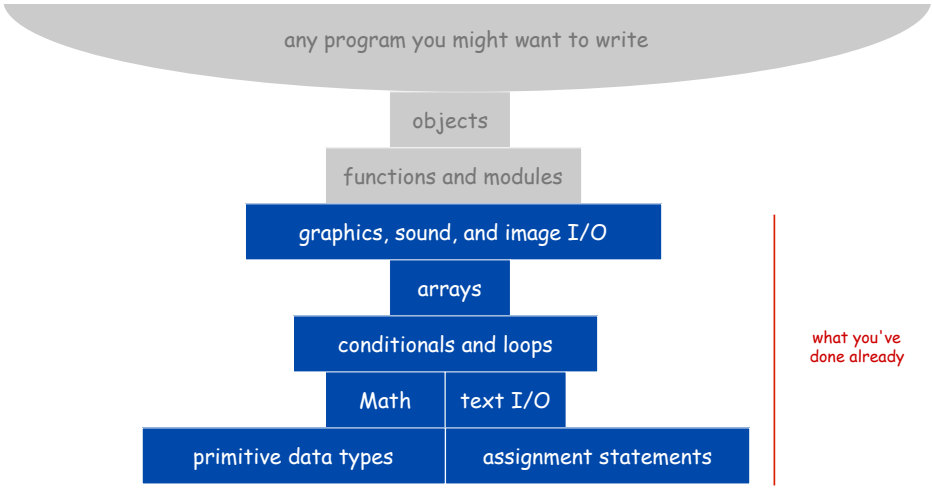
Program Development

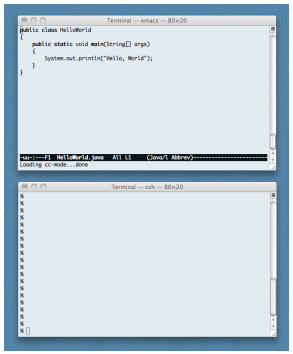




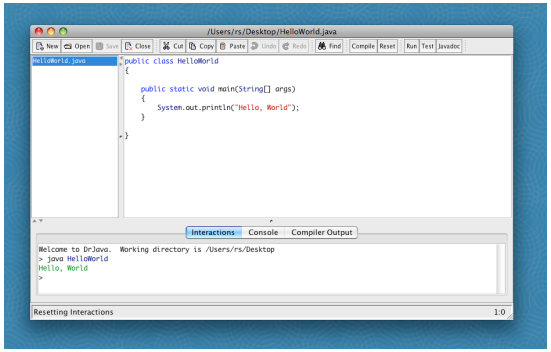
Program Development

Program development. Creating a program and putting it to good use.

Program development environment. Software to support cycle of editing, compiling, and executing programs.



command line



Dr. Java

Program Development in Java

0. Think about your problem.

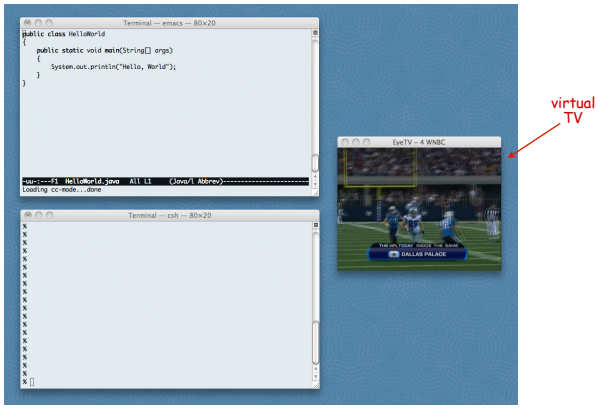
1. Edit your program.
   - Use a text editor.
   - Result: a text file such as `HelloWorld.java`.

2. Compile it to create an executable file.
   - Use the Java compiler
   - Result: a Java bytecode file file such as `HelloWorld.class`.
   - Mistake? Go back to 1 to fix and recompile.

3. Execute your program.
   - Use the Java runtime.
   - Result: your program's output.
   - Mistake? Go back to 1 to fix, recompile, and execute.

Program Development in Java (using command line)

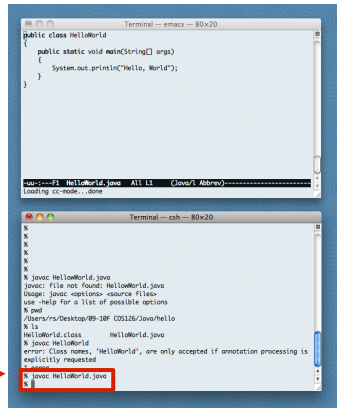
1. Edit your program using any text editor.
2. Compile it to create an executable file.
3. Execute your program.



Program Development in Java (using command line)

1. Edit your program.
2. Compile it by typing `javac HelloWorld.java` at the command line.
3. Execute your program.

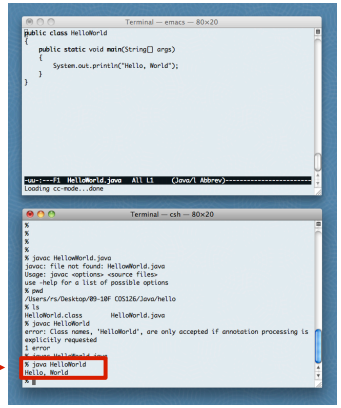
creates `HelloWorld.class`



Program Development in Java (using command line)

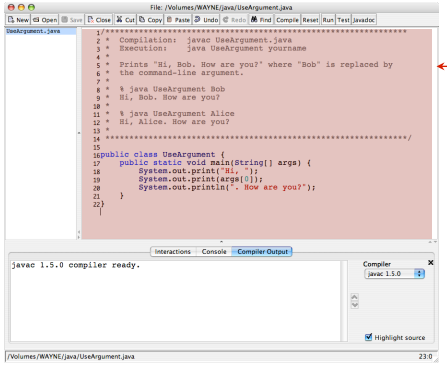
1. Edit your program.
2. Compile it to create an executable file.
3. Execute by typing `java HelloWorld` at the command line.

uses `HelloWorld.class`



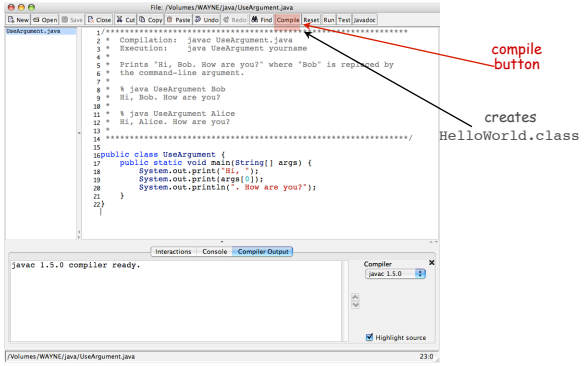
Program Development in Java (using Dr. Java)

1. Edit your program using the built-in text editor.
2. Compile it to create an executable file.
3. Execute your program.



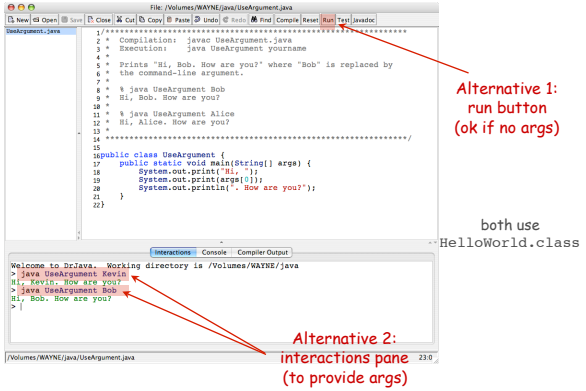
Program Development in Java (using Dr. Java)

1. Edit your program.
2. Compile it by clicking the "compile" button.
3. Execute your program.



Program Development in Java (using Dr. Java)

1. Edit your program.
2. Compile it to create an executable file.
3. Execute by clicking the "run" button or using Interactions pane.



A Short History

Program Development Environments: A Short History

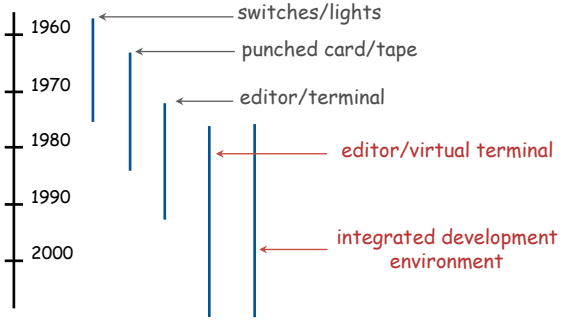
Historical context is important in computer science.

- We regularly use old software.
- We regularly emulate old hardware.
- We depend upon old concepts and designs.

First requirement in any computer system: program development.

Widely-used methods:

- Switches/lights.
- Punched cards.
- Terminal.
- Editor/virtual terminal.
- IDE.



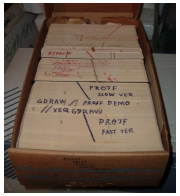
Switches and Lights

Use switches to enter binary program code, lights to read results.

PDP-8, circa 1970

Punched Cards / Line Printer

Use punched cards for program code, line printer for output.



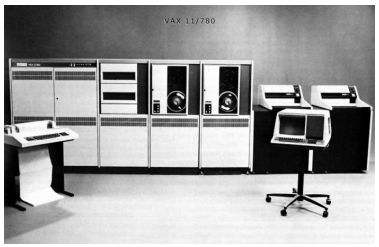
IBM System 360, circa 1975

Timesharing Terminal

Use terminal for editing program, reading output, and controlling computer.

VT-100 terminal

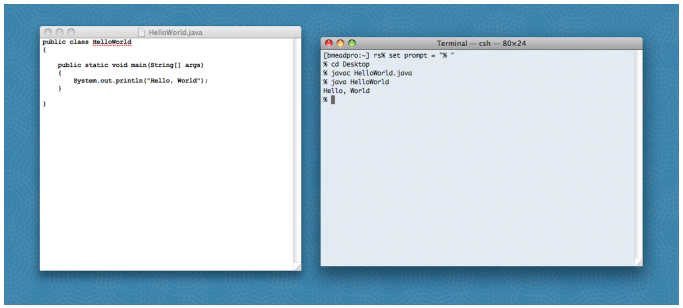
VAX 11/780 circa 1977



Timesharing: allowed many people to simultaneously use a single machine.

Editor and Virtual Terminal on a Personal Computer

Use an editor to create and make changes to the program text.
Use a virtual terminal to invoke the compiler and run the executable code.



Pros. Works with any language, useful for other tasks, used by pros.
Cons. Good enough for large projects?

Integrated Development Environment

Use a customized application for all program development tasks.

Ex 1. DrJava.

- Ideal for novices.
- Easy-to-use language-specific tools.

Ex 2. Eclipse.

- Widely used by professionals.
- Powerful debugging and style-checking tools.
- Steep learning curve.
- Overkill for short programs.

Lessons from Short History

First requirement in any computer system: program development.

Program development environment must support cycle of editing, compiling, and executing programs.

Two approaches that have served for decades:

- Editor and virtual terminal.
- Integrated development environment.

Macbook Air 2008

Xerox Alto 1978

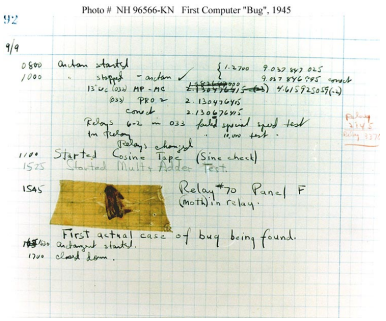
Debugging

Admiral Grace Murray Hopper

95% of Program Development

Def. A bug is a mistake in a computer program.

Programming is primarily a process of finding and fixing bugs.



Good news. Can use computer to test program.

Bad news. Cannot use computer to automatically find all bugs.

profound idea [stay tuned]

95% of Program Development

Debugging. 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)
            StdOut.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)
            StdOut.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 (int i = 0; i < n; i++) {
         while (n % i == 0)
            StdOut.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)
            StdOut.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)
            StdOut.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.ArithmeticExeption: / 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)
            StdOut.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 3 3 3 ...
```

infinite loop!

Debugging: The Beat Goes On

Success. Program factors 98 = 2 × 7².

- 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) {
            StdOut.print(i + " ");
            n = n / i;
         }
      }
   }
}
```

```
% java Factors 98
2 7 7 %
```

need newline

```
% java Factors 5
```

??? no output

```
% java Factors 6
2 %
```

??? missing the 3

Debugging: The Beat Goes On

Success. Program factors 98 = 2 × 7².

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

```
% 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

```
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) {
            StdOut.println(i + " ");
            n = n / i;
         }
         StdOut.println("TRACE: " + i + " " + 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) {
            StdOut.print(i + " ");
            n = n / i;
         }
      }
      StdOut.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) {
            StdOut.print(i + " ");
            n = n / i;
         }
      }
      StdOut.println();
   }
}
```

```
% java Factors 11111111
11 73 101 137

% java Factors 11111111111
21649 51329

% java Factors 11111111111111
11 239 4649 909091

% java Factors 1111111111111111
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) {
            StdOut.print(i + " ");
            n = n / i;
         }
      }
      StdOut.println();
   }
}
```

```
% java Factors 98
2 7 7

% java Factors 11111111
11 73 101

% java Factors 11111111111111
11 239 4649

% java Factors 1111111111111111
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) {
            StdOut.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 1111111111111111
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!

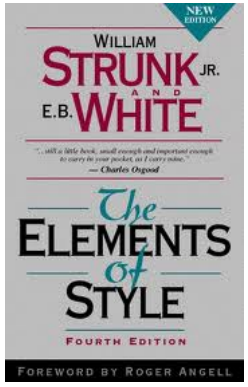
5. Too slow? Back to step 1 to try a different algorithm.

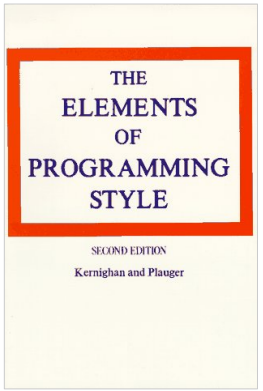
Debugging is Hard

" Debugging is twice as hard as writing the code in the first place. Therefore, if you write the code as cleverly as possible, you are, by definition, not smart enough to debug it. " — Brian Kernighan

" There are two ways of constructing a software design. One way is to make it so simple that there are obviously no deficiencies. And the other way is to make it so complicated that there are no obvious deficiencies. " — C. A. R. Hoare

Programming Style





Three Versions of the Same Program

```
public class HelloWorld
{
   public static void main(String[] args)
   {
      System.out.println("Hello, World");
   }
}
```

```
/*************************************************************
 *  Compilation:  javac HelloWorld.java
 *  Execution:    java HelloWorld
 *
 *  Prints "Hello, World".
 *  By tradition, this is everyone's first program.
 *
 *  % java HelloWorld
 *  Hello, World
 *
 *************************************************************/

public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello, World");
    }
}
```

```
public class HelloWorld { public static void main(String[] args)
{ System.out.println("Hello, World"); } }
```

Java

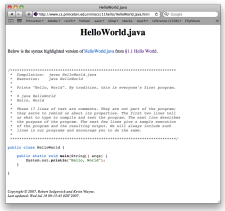
Programming Style

Different styles are appropriate in different contexts.

- Booksite.
- Textbook.
- COS 126 assignment.
- Java system libraries.

Enforcing consistent style can:

- Stifle creativity.
- Confuse style rules with language rules.



Emphasizing consistent style can:

- Make it easier to spot errors.
- Make it easier for others to read and use code.
- Enable IDE to provide useful visual cues.

Naming Conventions

Best practices.

- Be consistent.
- Choose descriptive variables names.
- Obey Java conventions on upper/lowercase.

purpose	good	bad	worse
factoring program	`Factors.java`	`factors.java`	`f.java`
is it a leap year?	`isLeapYear`	`leapyear`	`_$11110001`
loop-index variable	`i`	`ithTimeThroughLoop`	`fred`
read an int from standard input	`readInt()`	`int()`	`i()`
days per week	`DAYS_PER_WEEK`	`DPW`	`SEVEN`

Whitespace

Add whitespace to make your program more readable.

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

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

Best practices.

- Be consistent.
- One statement per line.
- Space between binary operators.

Indenting

Indent and add blank lines to reveal structure and nesting.

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

```
public class Factors {

   public static void main(String[] args) {
      long n = Long.parseLong(args[0]);

      for (long i = 2; i <= n; i++) {
         while (n % i == 0) {
            StdOut.print(i + " ");
            n = n / i;
         }
      }

   }
}
```

Best practices.

- Be consistent.
- 4 spaces per level of indentation.
- Blank lines between logical blocks of code.

Comments

Annotate what or why you are doing something, rather than how.

```
//  an end-of-line comment

/************************************************
 *  A block comment draws attention
 *  to itself.
 ************************************************/
```

Best practices.

- Comment logical blocks of code.
- Ensure comments agree with code.
- Comment every important variable.
- Comment any confusing code (or rewrite so that it's clear).
- Include header that describe purpose of program, how to compile, how to execute, any dependencies, and a sample execution.

COS 126 students:
also name, precept, and login

Comments

```
/*************************************************************
 *  Compilation:  javac Factors.java
 *  Execution:    java Factors n
 *  Dependencies: StdOut.java
 *
 *  Computes the prime factorization of n using brute force.
 *
 *   % java Factors 4444444444
 *   2 2 11 41 271 9091
 *
 *************************************************************/

public class Factors {

    public static void main(String[] args) {

        // integer to be factored
        long n = Long.parseLong(args[0]);

        // for each potential factor i of n
        for (long i = 2; i <= n; i++) {

            // if i is a factor of n, repeatedly divide it out
            while (n % i == 0) {
                StdOut.print(i + " ");
                n = n / i;
            }
        }
    }
}
```


Coding Standards

De facto Java coding standard.

http://www.oracle.com/technetwork/java/codeconvtoc-136057.html

Less pedantic version of Sun standard.

http://introcs.cs.princeton.edu/11style

COS 126 students:
follow these guidelines

Automated tool to enforce coding standard.

http://checkstyle.sourceforge.net

used when you click
"Check all Submitted Files"

Google Microsoft YAHOO! Adobe facebook

U.S.S. Grace Murray Hopper