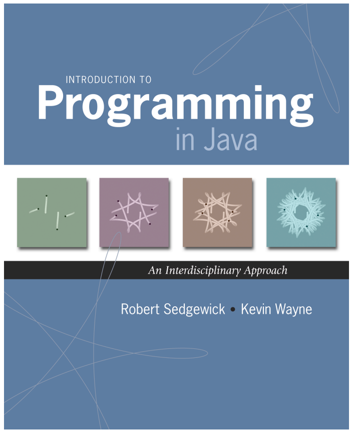
Program Development

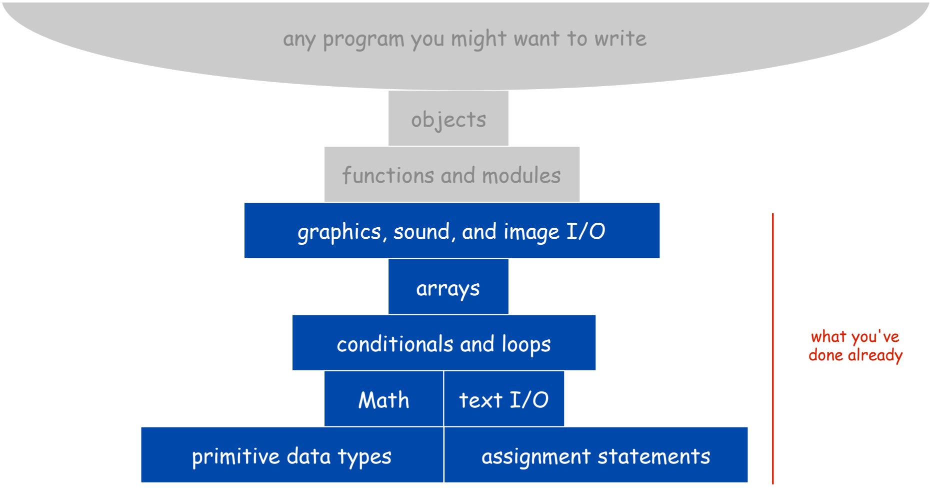


Introduction to Programming in Java: An Interdisciplinary Approach · Robert Sedgewick and Kevin Wayne · Copyright © 2008 · February 11, 2010 8:48 AM

Program Development

Ada Lovelace

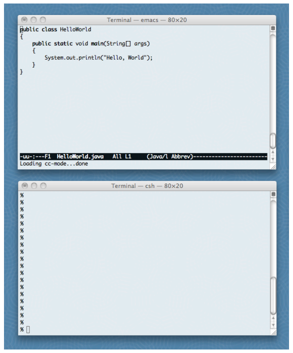
A Foundation for Programming



Program Development

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

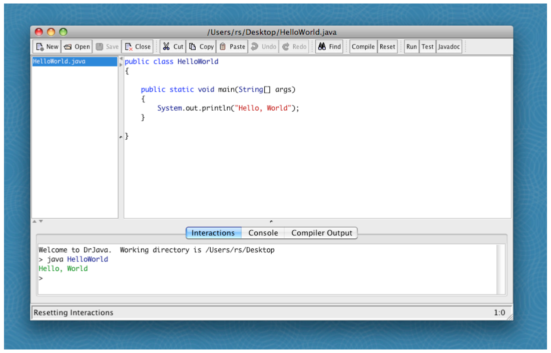
Program development environment. Software to support cycle of editing to fix mistakes, compiling programs, running programs, and examining output.



```
public class HelloWorld
{
    public static void main(String[] args)
    {
        System.out.println("Hello, World");
    }
}
-uu-:---F1  HelloWorld.java   All L1     (Java/l Abbrev)-----------------------
Loading cc-mode...done
```

```
%
%
%
%
%
%
%
%
%
%
%
%
%
%
%
%
%
%
%
%
```

command line



```
public class HelloWorld
{

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

}
```

```
Welcome to DrJava.  Working directory is /Users/rs/Desktop
> java HelloWorld
Hello, World
>
```

Dr. Java

Program Development

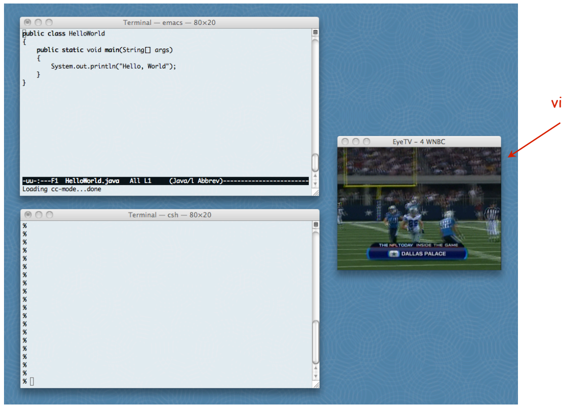
Program development in Java (bare-bones).

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

Program Development (using command line)

Program development in Java (using command line).

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



editor running in virtual terminal

Terminal — emacs — 80×20

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

```
-uu-:---F1  HelloWorld.java   All L1     (Java/l Abbrev)-----------------------
Loading cc-mode...done
```

second terminal for commands

Terminal — csh — 80×20

```
%
%
%
%
%
%
%
%
%
%
%
%
%
%
%
%
%
%
%
% 
```

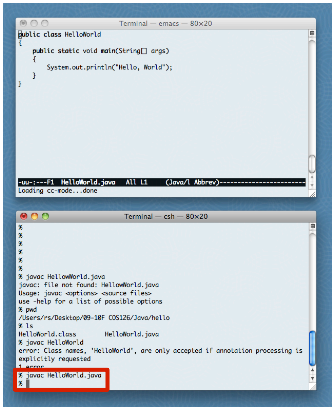
virtual TV

Program Development (using command line)

Program development in Java (using command line).

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

creates `HelloWorld.class`



Terminal — emacs — 80×20

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

```
-uu-:---F1  HelloWorld.java   All L1     (Java/l Abbrev)-----------------------
Loading cc-mode...done
```

Terminal — csh — 80×20

```
%
%
%
%
%
%
% javac HellowWorld.java
javac: file not found: HellowWorld.java
Usage: javac <options> <source files>
use -help for a list of possible options
% pwd
/Users/rs/Desktop/09-10F COS126/Java/hello
% ls
HelloWorld.class        HelloWorld.java
% javac HelloWorld
error: Class names, 'HelloWorld', are only accepted if annotation processing is
explicitly requested
1 error
% javac HelloWorld.java
% 
```

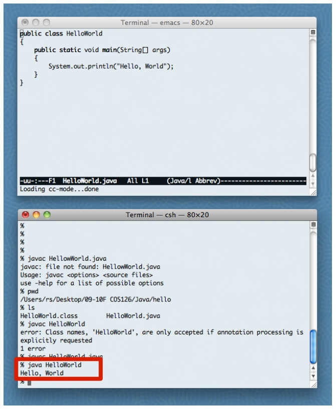
invoke Java compiler at command line

Program Development (using command line)

Program development in Java (using command line).

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

uses `HelloWorld.class`



Terminal — emacs — 80×20

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

```
-uu-:---F1  HelloWorld.java   All L1     (Java/l Abbrev)-----------------------
Loading cc-mode...done
```

Terminal — csh — 80×20

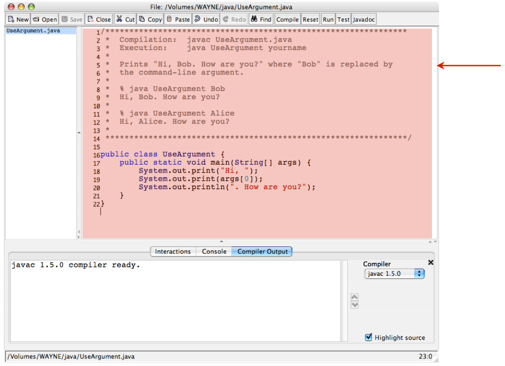
```
%
%
%
%
% javac HellowWorld.java
javac: file not found: HellowWorld.java
Usage: javac <options> <source files>
use -help for a list of possible options
% pwd
/Users/rs/Desktop/09-10F COS126/Java/hello
% ls
HelloWorld.class        HelloWorld.java
% javac HelloWorld
error: Class names, 'HelloWorld', are only accepted if annotation processing is
explicitly requested
1 error
% javac HelloWorld.java
% java HelloWorld
Hello, World
% 
```

invoke Java runtime at command line

Program Development (using Dr. Java)

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. Run your program.



File: /Volumes/WAYNE/java/UseArgument.java
New Open Save Close Cut Copy Paste Undo Redo Find Compile Reset Run Test Javadoc
UseArgument.java
Interactions Console Compiler Output
Compiler
javac 1.5.0
☑ Highlight source
/Volumes/WAYNE/java/UseArgument.java
23:0

```
/*************************************************************************
 *  Compilation:  javac UseArgument.java
 *  Execution:    java UseArgument yourname
 *
 *  Prints "Hi, Bob. How are you?" where "Bob" is replaced by
 *  the command-line argument.
 *
 *  % java UseArgument Bob
 *  Hi, Bob. How are you?
 *
 *  % java UseArgument Alice
 *  Hi, Alice. How are you?
 *
 *************************************************************************/

public class UseArgument {
    public static void main(String[] args) {
        System.out.print("Hi, ");
        System.out.print(args[0]);
        System.out.println(". How are you?");
    }
}
```

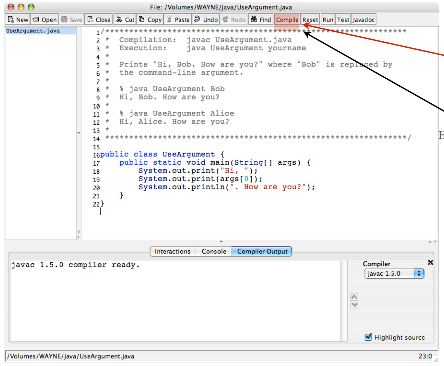
```
javac 1.5.0 compiler ready.
```

text editor

Program Development (using Dr. Java)

Program development in Java (using Dr. Java).

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



File: /Volumes/WAYNE/java/UseArgument.java
New Open Save Close Cut Copy Paste Undo Redo Find Compile Reset Run Test Javadoc
UseArgument.java
Interactions Console Compiler Output
Compiler
javac 1.5.0
☑ Highlight source
/Volumes/WAYNE/java/UseArgument.java
23:0

```
/*************************************************************************
 *  Compilation:  javac UseArgument.java
 *  Execution:    java UseArgument yourname
 *
 *  Prints "Hi, Bob. How are you?" where "Bob" is replaced by
 *  the command-line argument.
 *
 *  % java UseArgument Bob
 *  Hi, Bob. How are you?
 *
 *  % java UseArgument Alice
 *  Hi, Alice. How are you?
 *
 *************************************************************************/

public class UseArgument {
    public static void main(String[] args) {
        System.out.print("Hi, ");
        System.out.print(args[0]);
        System.out.println(". How are you?");
    }
}
```

```
javac 1.5.0 compiler ready.
```

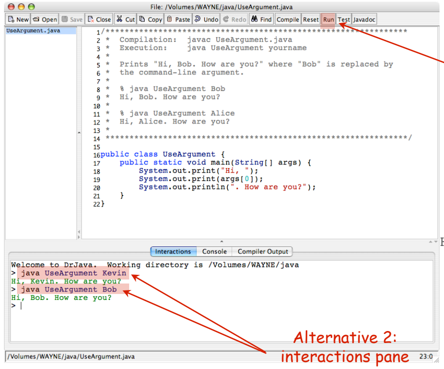
compile button

creates HelloWorld.class

Program Development (using Dr. Java)

Program development in Java (using Dr. Java).

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



File: /Volumes/WAYNE/java/UseArgument.java
New Open Save Close Cut Copy Paste Undo Redo Find Compile Reset Run Test Javadoc
UseArgument.java
Interactions Console Compiler Output
/Volumes/WAYNE/java/UseArgument.java
23:0

```
/*************************************************************************
 *  Compilation:  javac UseArgument.java
 *  Execution:    java UseArgument yourname
 *
 *  Prints "Hi, Bob. How are you?" where "Bob" is replaced by
 *  the command-line argument.
 *
 *  % java UseArgument Bob
 *  Hi, Bob. How are you?
 *
 *  % java UseArgument Alice
 *  Hi, Alice. How are you?
 *
 *************************************************************************/

public class UseArgument {
    public static void main(String[] args) {
        System.out.print("Hi, ");
        System.out.print(args[0]);
        System.out.println(". How are you?");
    }
}
```

```
Welcome to DrJava.  Working directory is /Volumes/WAYNE/java
> java UseArgument Kevin
Hi, Kevin. How are you?
> java UseArgument Bob
Hi, Bob. How are you?
> |
```

Alternative 1:
run button
(ok if no args)

both use
HelloWorld.class

Alternative 2:
interactions pane
(to provide args)

Note: Program Style

Three versions of the same program.

```
// java HelloWorld
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");
    }
}
```

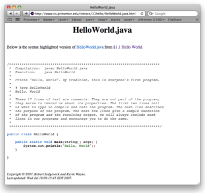
fonts, color, comments,
and extra space are for
human readability
(not machine readability)

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

Programming Style

Different styles are appropriate in different contexts.

- Dr. Java.
- Booksite.
- Textbook.
- COS 126 assignment.



HelloWorld.java
http://www.cs.princeton.edu/introcs/11hello/HelloWorld.java.html
Google
Princeton ▾ Adobe ▾ rsrch ▾ Yahoo! save ▾ shop ▾ stocks teach ▾ reference (1336) ▾ FlipShare

HelloWorld.java

Below is the syntax highlighted version of HelloWorld.java from §1.1 Hello World.

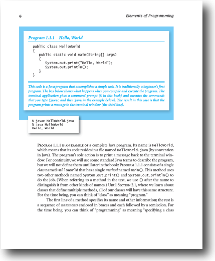
```
/*************************************************************************
 *  Compilation:  javac HelloWorld.java
 *  Execution:    java HelloWorld
 *
 *  Prints "Hello, World". By tradition, this is everyone's first program.
 *
 *  % java HelloWorld
 *  Hello, World
 *
 *  These 17 lines of text are comments. They are not part of the program;
 *  they serve to remind us about its properties. The first two lines tell
 *  us what to type to compile and test the program. The next line describes
 *  the purpose of the program. The next few lines give a sample execution
 *  of the program and the resulting output. We will always include such
 *  lines in our programs and encourage you to do the same.
 *
 *************************************************************************/

public class HelloWorld {

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

}
```

Copyright © 2007, Robert Sedgewick and Kevin Wayne.
Last updated: Wed Jul 18 09:15:45 EDT 2007.



6 *Elements of Programming*

Program 1.1.1 Hello, World

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

This code is a Java program that accomplishes a simple task. It is traditionally a beginner's first program. The box below shows what happens when you compile and execute the program. The terminal application gives a command prompt (% in this book) and executes the commands that you type (javac and then java in the example below). The result in this case is that the program prints a message in the terminal window (the third line).

```
% javac HelloWorld.java
% java HelloWorld
Hello, World
```

Program 1.1.1 is an example of a complete Java program. Its name is HelloWorld, which means that its code resides in a file named HelloWorld.java (by convention in Java). The program's sole action is to print a message back to the terminal window. For continuity, we will use some standard Java terms to describe the program, but we will not define them until later in the book: Program 1.1.1 consists of a single *class* named HelloWorld that has a single *method* named main(). This method uses two other methods named System.out.print() and System.out.println() to do the job. (When referring to a method in the text, we use () after the name to distinguish it from other kinds of names.) Until Section 2.1, where we learn about classes that define multiple methods, all of our classes will have this same structure. For the time being, you can think of "class" as meaning "program."

The first line of a method specifies its name and other information; the rest is a sequence of *statements* enclosed in braces and each followed by a semicolon. For the time being, you can think of "programming" as meaning "specifying a class

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 development environment to provide useful visual cues.

Bottom line for COS 126. Life is easiest if you use Dr. Java style.

A Short History

Program Development Environments: A Short History

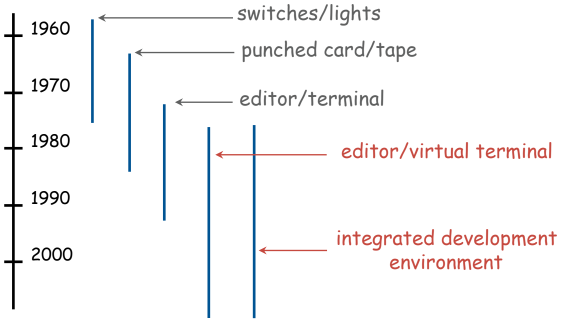
Historical context is important in computer science.

- 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.



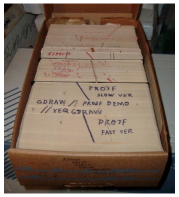
Switches and Lights

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.



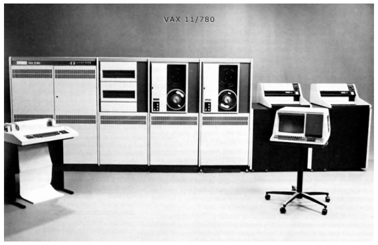
IBM System 360, circa 1975

Timesharing Terminal

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

VT-100 terminal

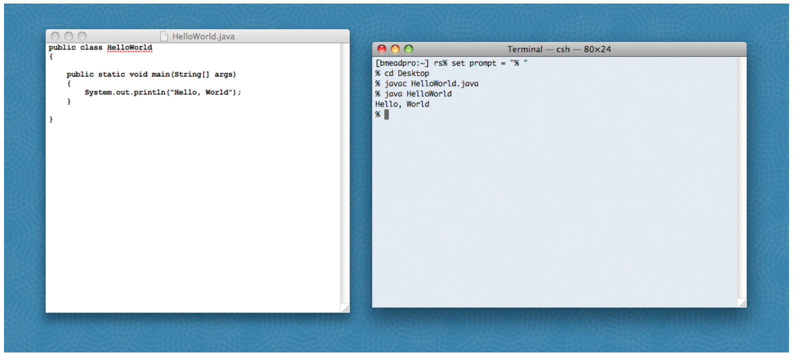
VAX 11/780 circa 1977



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.



```
public class HelloWorld
{

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

}
```

```
[bmeadpro:~] rs% set prompt = "% "
% cd Desktop
% javac HelloWorld.java
% java HelloWorld
Hello, World
%
```

Pros:

- Works with any language.
- Useful for other tasks.
- Used by professionals.

Cons:

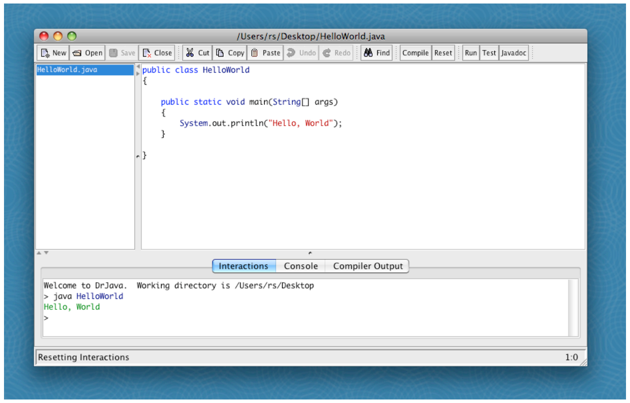
- Good enough for long programs?
- Dealing with two applications.

Integrated Development Environment

Use a customized application for all program development tasks.

Ex. drjava

http://drjava.org



```
public class HelloWorld
{

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

}
```

```
Welcome to DrJava.  Working directory is /Users/rs/Desktop
> java HelloWorld
Hello, World
>
```

Pros:

- Easy-to-use language-specific tools.
- System-independent (in principle).
- Used by professionals.

Cons:

- Overkill for short programs?
- Large application to learn and maintain.
- Skills may not transfer to other languages.

Lessons from Short History

First requirement in any computer system: program development.

Programming is primarily a process of finding and fixing mistakes.

Program development environment must support cycle of editing to fix errors, compiling program, running program, and examining output.

Two approaches that have served for decades:

- Editor and virtual terminal.
- Integrated development environment.

Xerox Alto 1978

Macbook Air 2008

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.

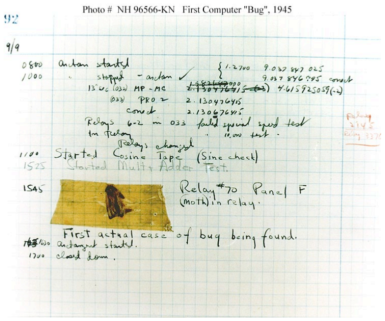


Photo # NH 96566-KN First Computer "Bug", 1945

92

9/9

0800 Antan started {1.2700 9.037 847 025
1000 stopped - antan ✓ 9.037 846 995 correct
13" UC (032) MP-MC ~~1.982147000~~ ~~2.130476415~~ ~~(-3)~~ 4.615925059(-2)
(033) PRO 2 2.130476415
correct 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.

Relay 2145
Relay 3370

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

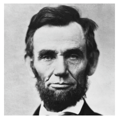
Debugging. Cyclic process of editing, compiling, and fixing errors.

- 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.

check if i is a factor

```
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

      }
   }
}
```

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.