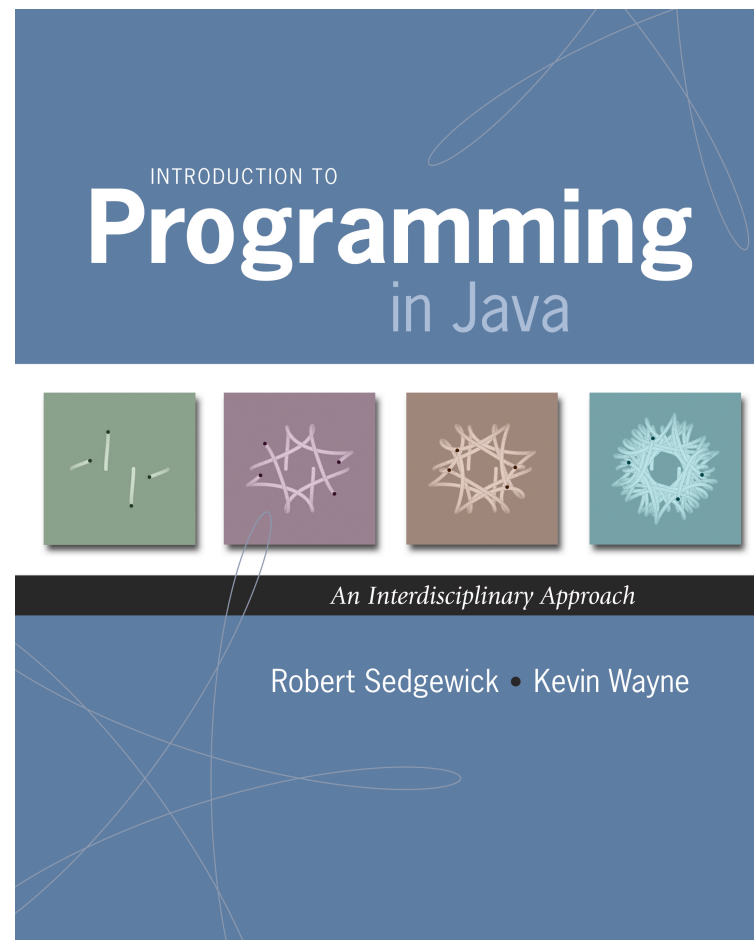


A human being should be able to  
change a diaper,  
plan an invasion,  
butcher a hog,  
conn a ship,  
design a building,  
write a sonnet,  
balance accounts,  
build a wall,  
set a bone,  
comfort the dying,  
take orders,  
give orders,  
cooperate,  
act alone,  
solve equations,  
analyze a new problem,  
pitch manure,  
program a computer,  
cook a tasty meal,  
fight efficiently, and  
die gallantly.

Specialization is for insects.

Robert A. Heinlein  
*Time Enough for Love* (1973)

# 1.1 Your First Program

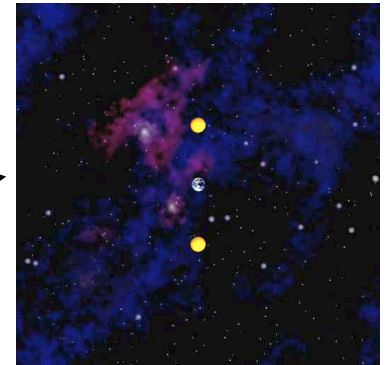


# Why Programming?

Why programming? Need to tell computer what you want it to do.

Naive ideal. Natural language instructions.

"Please simulate the motion of N heavenly bodies, subject to Newton's laws of motion and gravity."



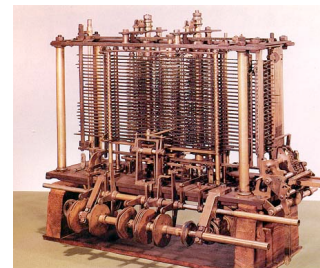
Prepackaged solutions (apps)? Great, when what they do is what you want.



Programming. Enables you to make a computer do **anything** you want.



Ada Lovelace



Analytic Engine

well, almost anything  
[stay tuned]



# Languages

**Machine languages.** Tedious and error-prone.

**Natural languages.** Ambiguous; can be difficult to parse.

**Kids Make Nutritious Snacks.**

**Red Tape Holds Up New Bridge.**

**Police Squad Helps Dog Bite Victim.**

**Local High School Dropouts Cut in Half.**

[ real newspaper headlines, compiled by Rich Pattis ]

**High-level programming languages.** Acceptable tradeoff.

*“Instead of imagining that our main task is to instruct a computer what to do, let us concentrate rather on explaining to human beings what we want a computer to do.” – Donald Knuth*



# Why Program?

## Why program?

- A natural, satisfying and creative experience.
- Enables accomplishments not otherwise possible.
- Opens new world of intellectual endeavor.

First challenge. Learn a programming language.

Next question. Which one?



Naive ideal. A single programming language.

# Our Choice: Java

## Java features.

- Widely used.
- Widely available.
- Embraces full set of modern abstractions.
- Variety of automatic checks for mistakes in programs.

## Java economy.

← \$100 billion,  
5 million developers

- Mars rover.
- Cell phones.
- Blu-ray Disc.
- Web servers.
- Medical devices.
- Supercomputing.
- ...



James Gosling

<http://java.net/jag>

# Why Java?

## Java features.

- Widely used.
- Widely available.
- Embraces full set of modern abstractions.
- Variety of automatic checks for mistakes in programs.

## Facts of life.

- No language is perfect.
- We need to choose **some** language.

## Our approach.

- Minimal subset of Java.
- Develop general programming skills that are applicable to many languages

It's not about the language!

*“There are only two kinds of programming languages: those people always [gripe] about and those nobody uses.”*

– Bjarne Stroustrup



# A Rich Subset of the Java Language

Built-In Types	
<code>int</code>	<code>double</code>
<code>long</code>	<code>String</code>
<code>char</code>	<code>boolean</code>

System
<code>System.out.println()</code>
<code>System.out.print()</code>
<code>System.out.printf()</code>

Math Library	
<code>Math.sin()</code>	<code>Math.cos()</code>
<code>Math.log()</code>	<code>Math.exp()</code>
<code>Math.sqrt()</code>	<code>Math.pow()</code>
<code>Math.min()</code>	<code>Math.max()</code>
<code>Math.abs()</code>	<code>Math.PI</code>

Flow Control	
<code>if</code>	<code>else</code>
<code>for</code>	<code>while</code>

Parsing
<code>Integer.parseInt()</code>
<code>Double.parseDouble()</code>

Primitive Numeric Types		
<code>+</code>	<code>-</code>	<code>*</code>
<code>/</code>	<code>%</code>	<code>++</code>
<code>--</code>	<code>&gt;</code>	<code>&lt;</code>
<code>&lt;=</code>	<code>&gt;=</code>	<code>==</code>
<code>!=</code>		

Boolean	
<code>true</code>	<code>false</code>
<code>  </code>	<code>&amp;&amp;</code>
<code>!</code>	

Punctuation	
<code>{</code>	<code>}</code>
<code>(</code>	<code>)</code>
<code>,</code>	<code>;</code>

Assignment
<code>=</code>

String	
<code>+</code>	<code>""</code>
<code>length()</code>	<code>compareTo()</code>
<code>charAt()</code>	<code>matches()</code>

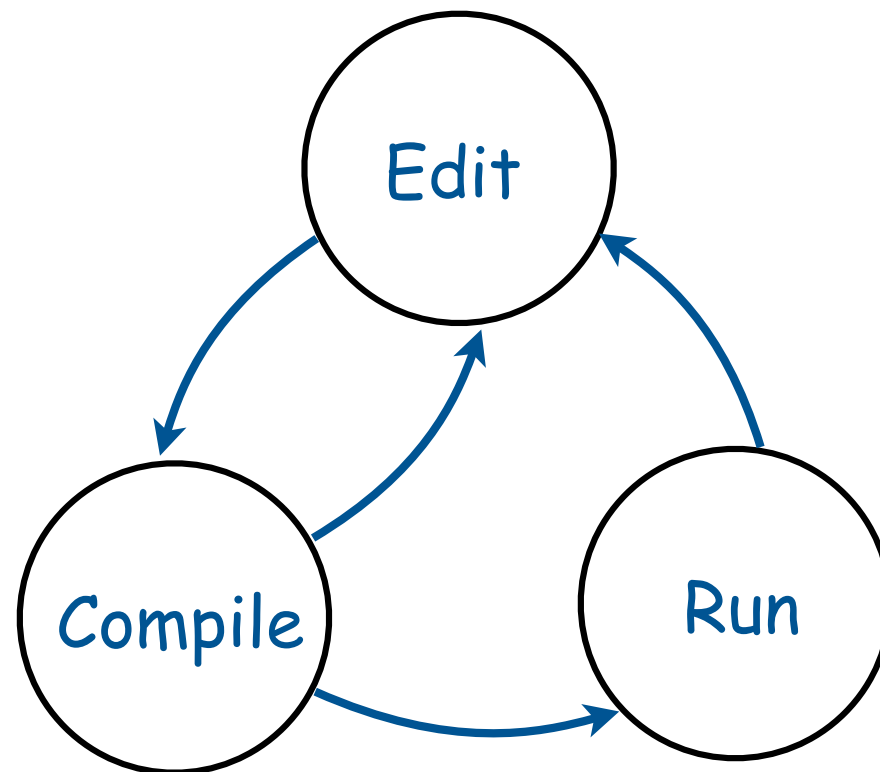
Arrays
<code>a[i]</code>
<code>new</code>
<code>a.length</code>

Objects	
<code>class</code>	<code>static</code>
<code>public</code>	<code>private</code>
<code>final</code>	<code>toString()</code>
<code>new</code>	<code>main()</code>



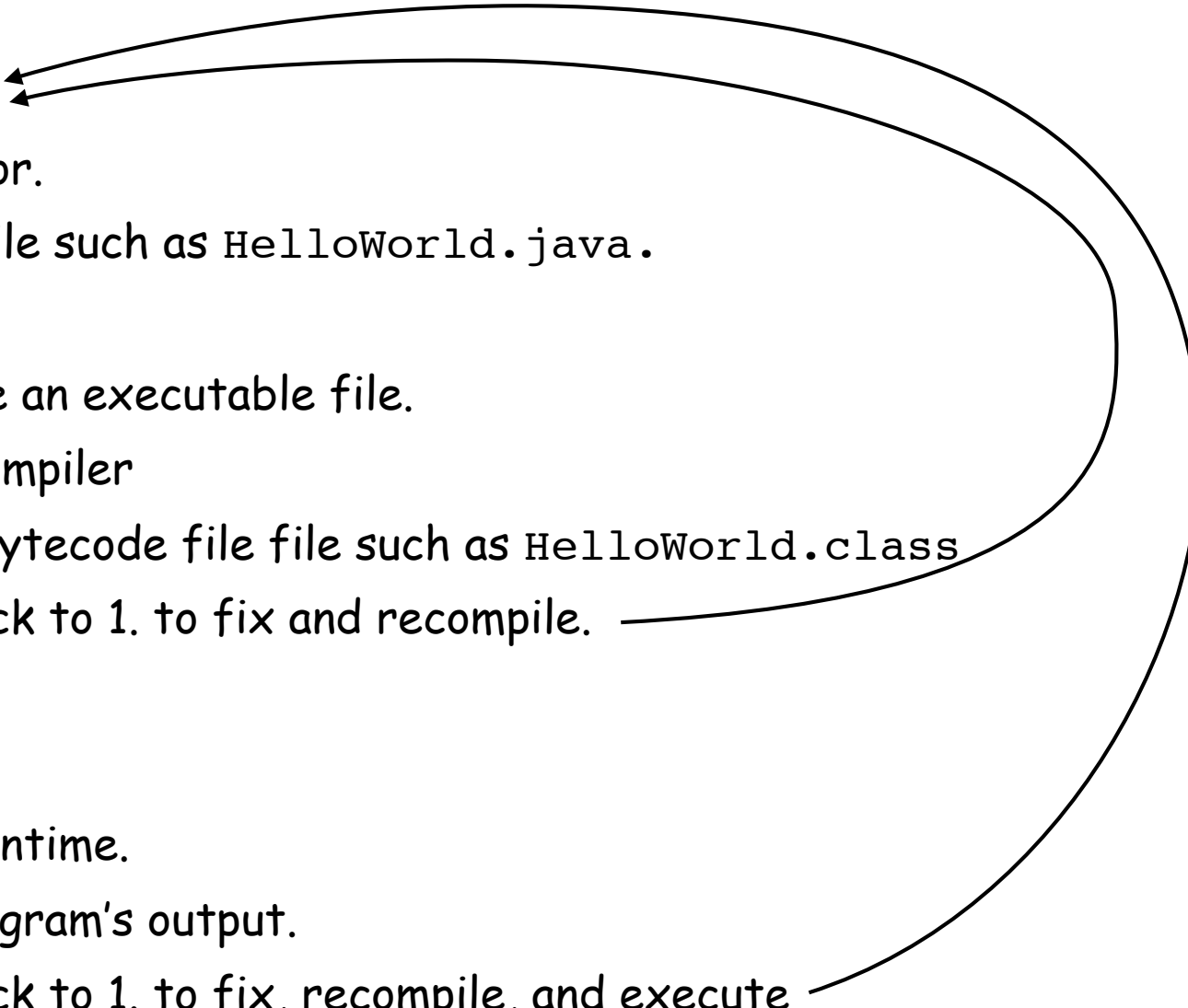
# Program Development

---



# 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
- 
- A diagram consisting of two curved arrows pointing from the right back to the left. The first arrow starts from the 'Mistake? Go back to 1. to fix and recompile.' bullet point in step 2 and points to the '1. Edit your program.' step. The second arrow starts from the 'Mistake? Go back to 1. to fix, recompile, and execute' bullet point in step 3 and also points to the '1. Edit your program.' step.

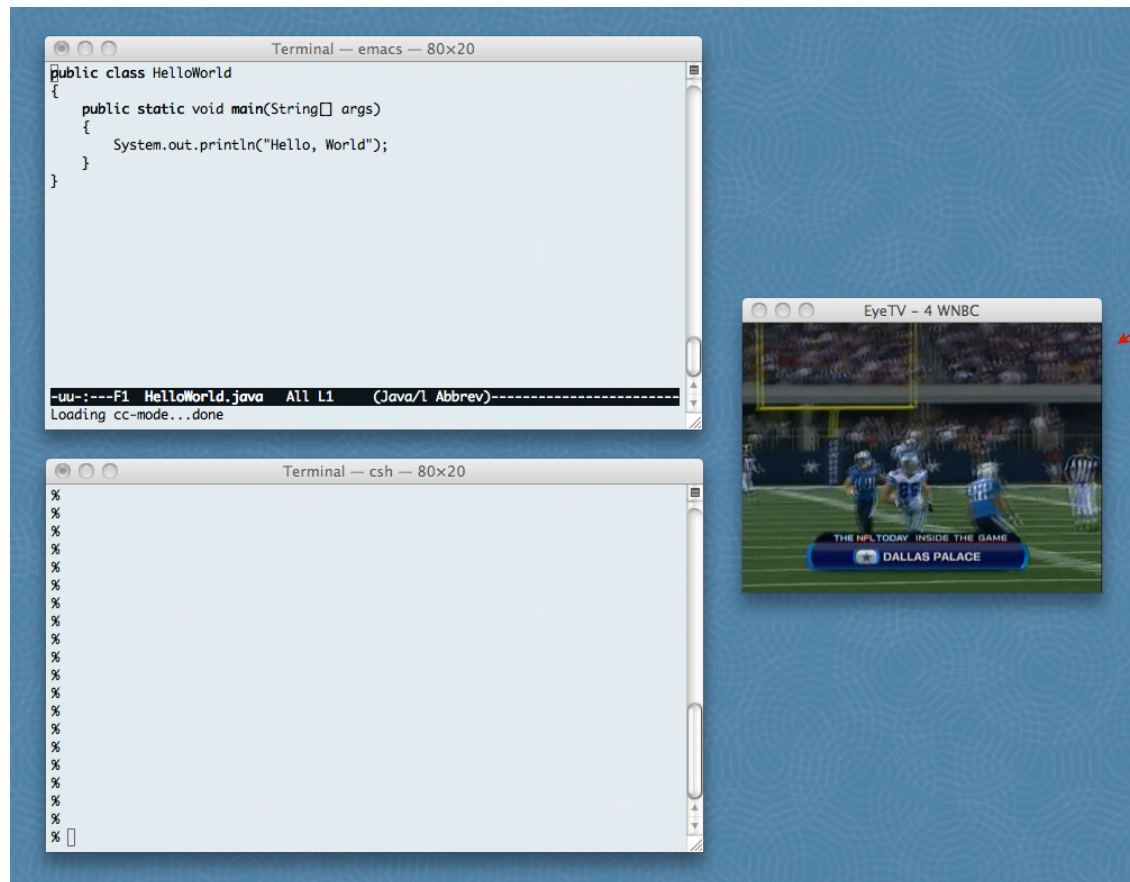
# Program Development (virtual terminals)

## Program development in Java (using virtual terminals).



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



virtual  
TV

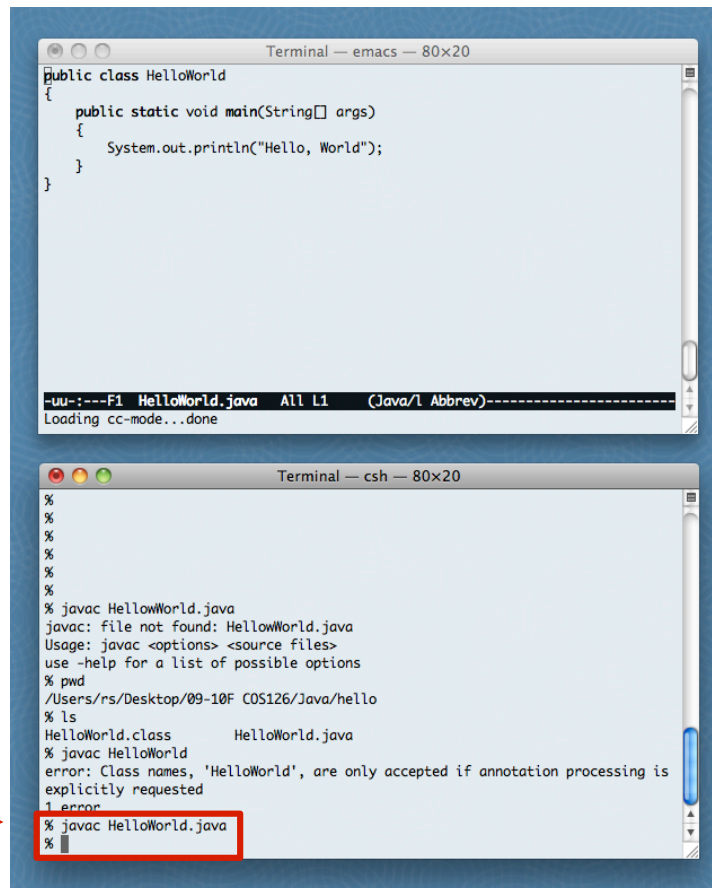
second terminal  
for commands

# Program Development (virtual terminals)

## Program development in Java (using virtual terminals).

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

creates  
HelloWorld.class



The image shows two terminal windows. The top window, titled 'Terminal — emacs — 80x20', displays the source code for a Java class named HelloWorld. The code is as follows:

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

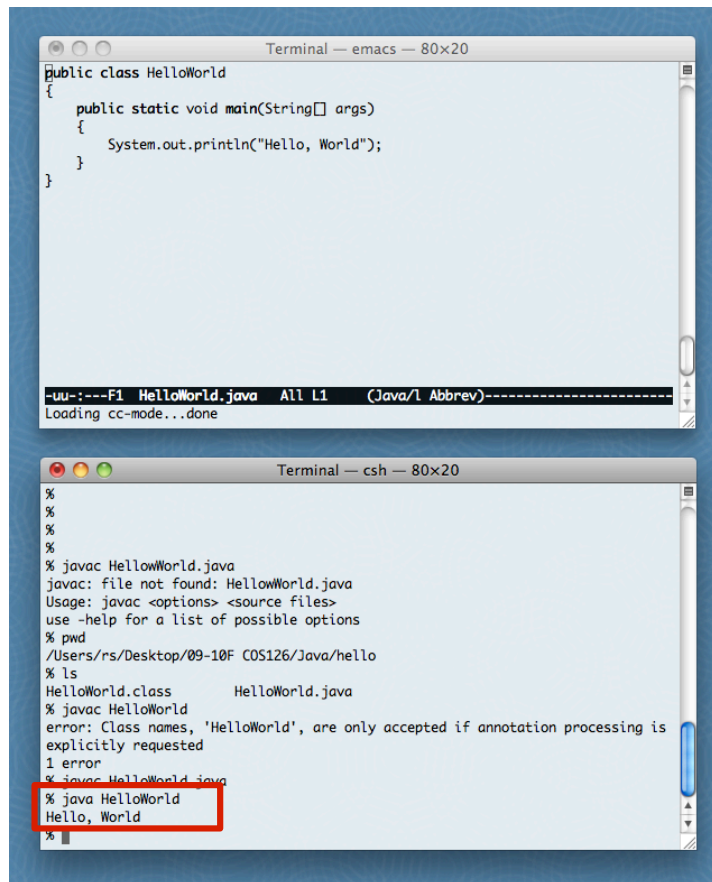
The bottom window, titled 'Terminal — csh — 80x20', shows the output of the compilation process. It starts with several blank lines, followed by the command `javac HelloWorld.java`. The output shows an error: `javac: file not found: HelloWorld.java`. The user then runs `pwd`, which shows the current directory is `/Users/rs/Desktop/09-10F COS126/Java/hello`. The user then runs `ls`, which shows the files `HelloWorld.class` and `HelloWorld.java` in the current directory. Finally, the user runs `javac HelloWorld`, which results in an error: `error: Class names, 'HelloWorld', are only accepted if annotation processing is explicitly requested`. The user then runs `javac HelloWorld.java`, which is highlighted with a red box.

invoke Java compiler  
at command line

# Program Development (virtual terminals)

## Program development in Java (using virtual terminals).

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



The image shows two terminal windows. The top window, titled 'Terminal — emacs — 80x20', displays the source code for a Java class named 'HelloWorld'. The code is as follows:

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

The bottom window, titled 'Terminal — csh — 80x20', shows the execution of the program. It starts with several blank lines, followed by the command `javac HelloWorld.java`. The output shows an error: `javac: file not found: HelloWorld.java`. The user then runs `pwd`, which returns `/Users/rs/Desktop/09-10F COS126/Java/hello`. Next, the user runs `ls`, which lists `HelloWorld.class` and `HelloWorld.java`. The user then runs `javac HelloWorld`, which results in an error: `error: Class names, 'HelloWorld', are only accepted if annotation processing is explicitly requested`. Finally, the user runs `java HelloWorld`, which outputs `Hello, World`. A red box highlights the `java HelloWorld` command and its output.

uses  
HelloWorld.class

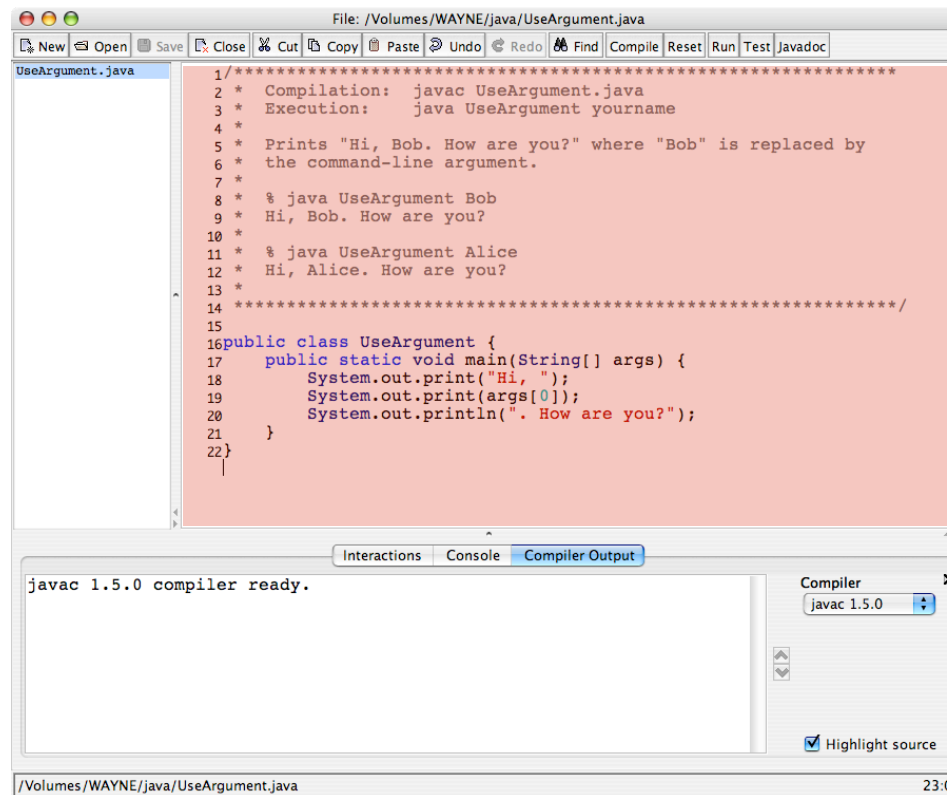
invoke Java runtime  
at command line

# Program Development (using DrJava)

## Program development in Java (using DrJava).



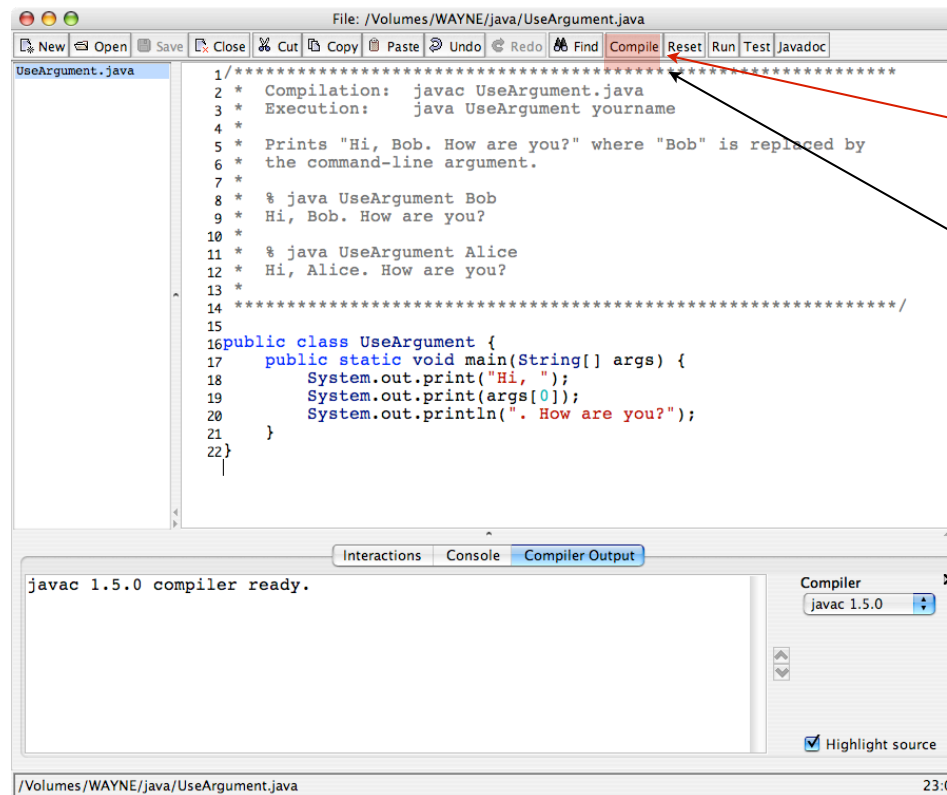
1. **Edit** your program using the built-in text editor.
2. Compile it to create an executable file.
3. Run your program.



# Program Development (using DrJava)

## Program development in Java (using DrJava).

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



# Program Development (using DrJava)

## Program development in Java (using DrJava).

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

The screenshot shows the DrJava IDE interface. The top window displays the source code for `UseArgument.java`. The code includes a main method that prints "Hi, " followed by the first command-line argument. The bottom window shows the command prompt with the following interactions:

```
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:  
command line  
(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");
    }
}
```



Fonts, color, comments,  
and extra space are not  
relevant to Java.

```
/*
 * 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"); } }
```

# Note: Program Style

Different styles are appropriate in different contexts.

- DrJava
- Booksite
- Book
- COS 126 assignment

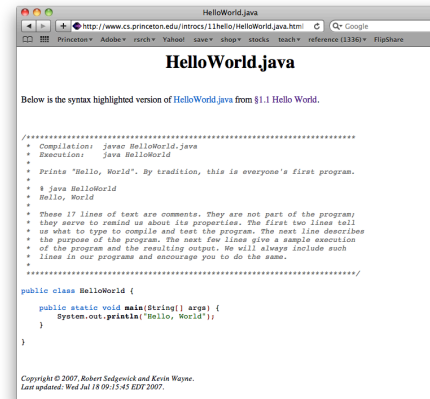
Enforcing consistent style can

- 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 DrJava style.



```

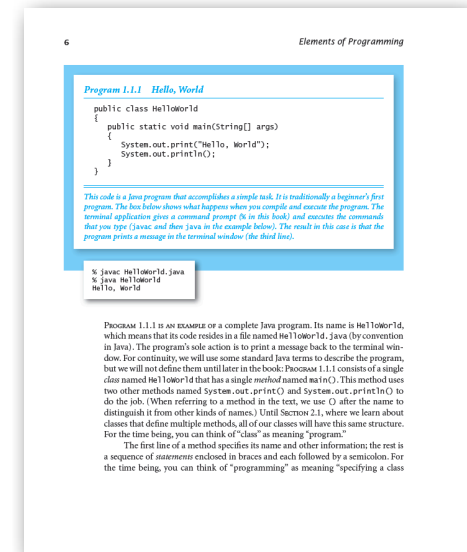
HelloWorld.java

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

/* *****
 * Compilation:  java HelloWorld.java
 * Execution:    java HelloWorld
 * Prints "Hello, World". By tradition, this is everyone's first program.
 *
 * <code> 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 Sedgwick and Kevin Wayne.
Last updated: Wed Jul 18 09:15:45 EDT 2007.
```



```

Program 1.1.1 Hello, World

public class HelloWorld
{
    public static void main(String[] args)
    {
        System.out.println("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 lines below show 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 (in Java 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).

% java 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 win-
dow. 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.println() 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
```

# 99% of program development

**Debugging.** Cyclic process of editing, compiling, and fixing mistakes (bugs).

**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*



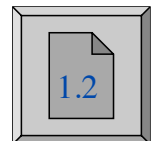
**Program Development Environment.** Software to support cycle of editing to fix mistakes, compiling programs, running programs, and examining output.

Examples: Terminal/editor, DrJava.

**Naive ideal.** "Please compile, execute, and debug my program".

**Bad news.** Even a computer can't find **all** the mistakes in your program.

profound idea  
[ stay tuned ]



# TEQ on Program Development

[easy if you did Exercise 1.1.2]

How do you cope with the following error messages?

A. `% javac HelloWorld.java`

`% java HelloWorld.java`

`Main method not public.`

B. `% javac HelloWorld.java`

`HelloWorld.java:3: invalid method declaration; return type required`

`public static main(String[] args)`

`^`

# Program Development Environments: A Short History

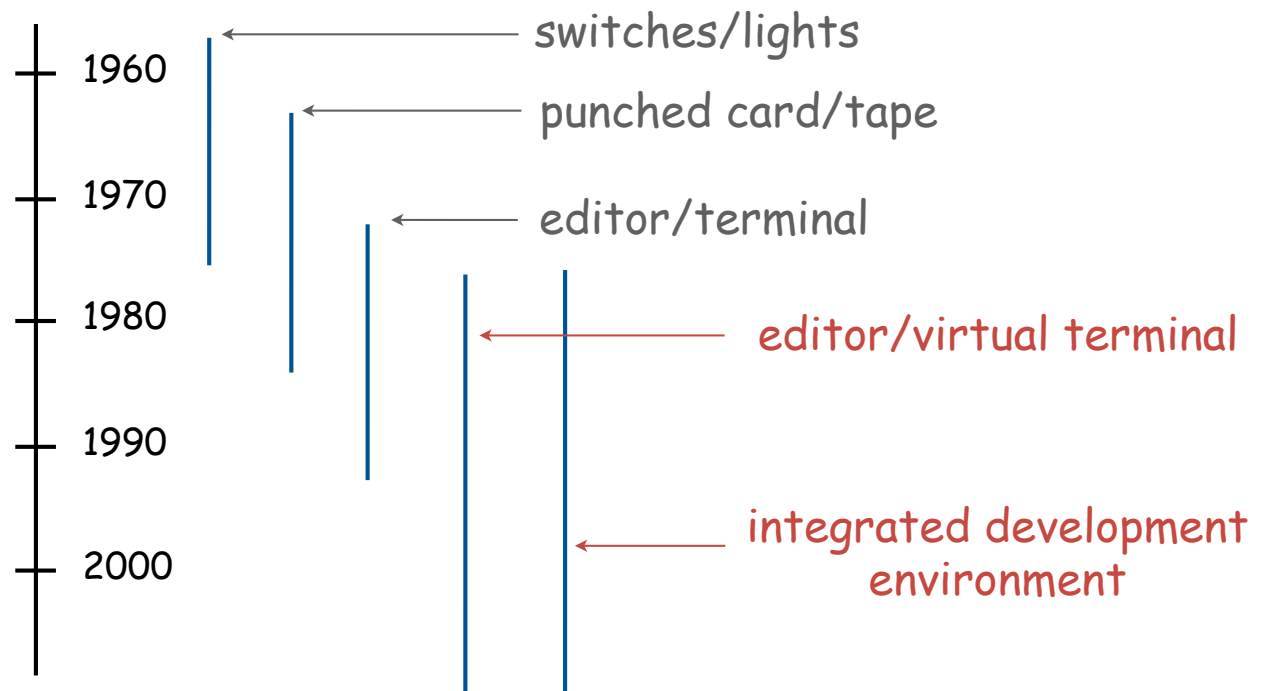
Historical context is important in computer science

- 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

VAX 11/780 circa 1977



VT-100 terminal



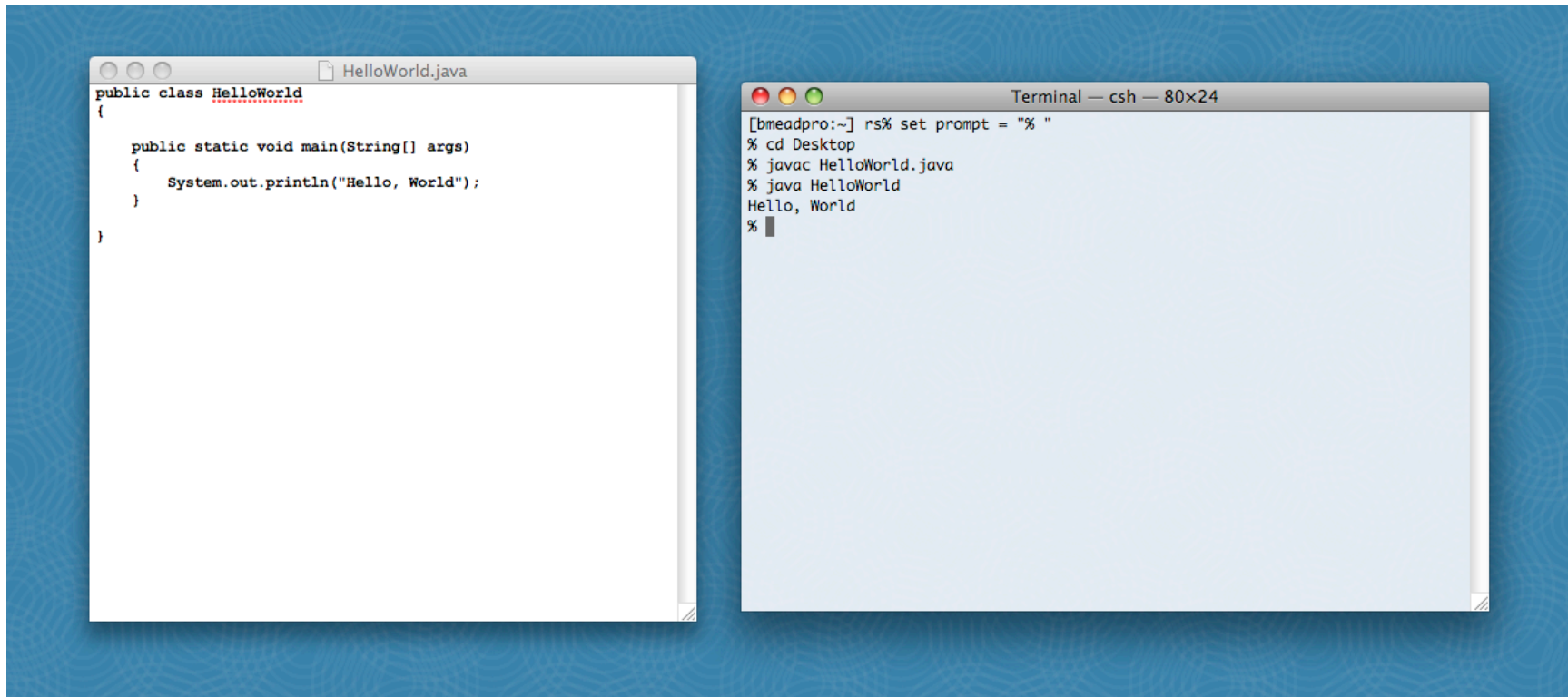
**Timesharing:** allowed many people to simultaneously use a single machine.



# 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 professionals.

## Cons:

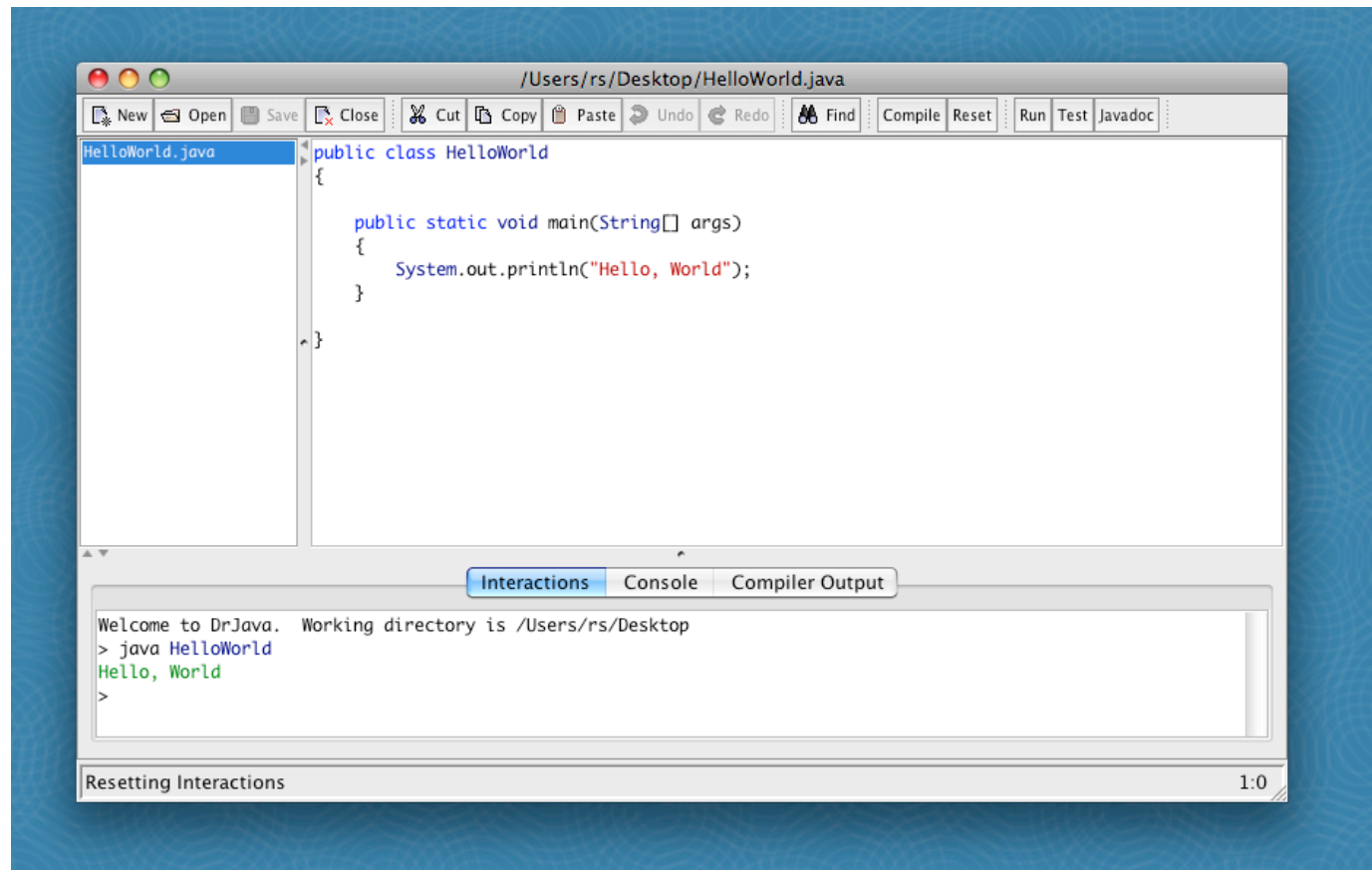
- Good enough for long programs?
- Dealing with two applications.

# Integrated Development Environment

Use a **customized application** for all program development tasks.

Ex.  drjava

<http://drjava.org>



## 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

Macbook Air 2008



Xerox Alto 1978



## 1.2 Built-in Types of Data

---



# Built-in Data Types

**Data type.** A set of values and operations defined on those values.

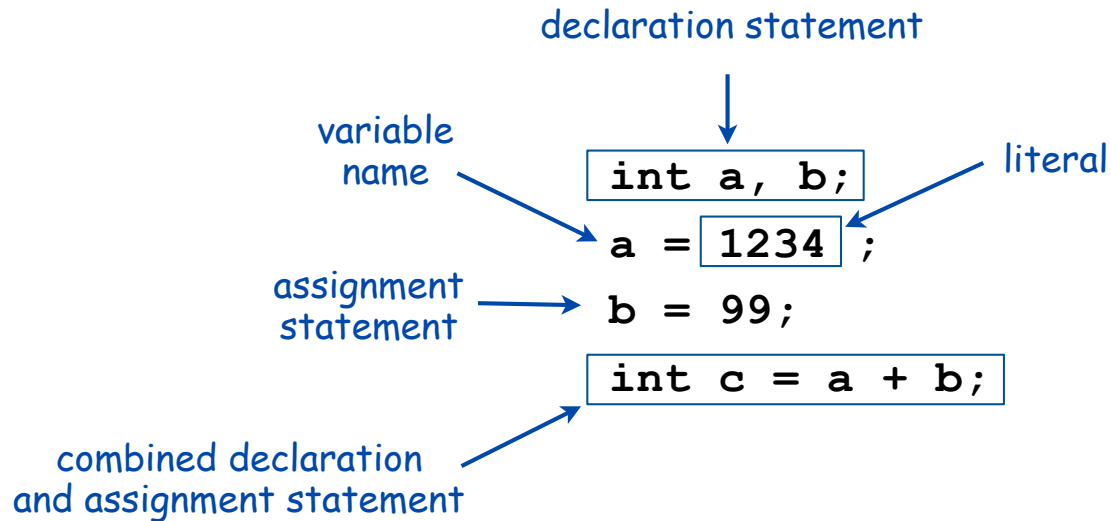
type	set of values	literal values	operations
char	characters	'A' '@'	compare
String	sequences of characters	"Hello World" "CS is fun"	concatenate
int	integers	17 12345	add, subtract, multiply, divide
double	floating-point numbers	3.1415 6.022e23	add, subtract, multiply, divide
boolean	truth values	true false	and, or, not

# Basic Definitions

**Variable.** A name that refers to a value.

**Literal.** Programming-language representation of a value.

**Assignment statement.** Associates a value with a variable.



# Trace

Trace. Table of variable values after each statement.

	<b>a</b>	<b>b</b>	<b>t</b>
<code>int a, b;</code>	undefined	undefined	undefined
<code>a = 1234;</code>	<b>1234</b>	undefined	undefined
<code>b = 99;</code>	1234	<b>99</b>	undefined
<code>int t = a;</code>	1234	99	<b>1234</b>
<code>a = b;</code>	<b>99</b>	99	1234
<code>b = t;</code>	99	<b>1234</b>	1234

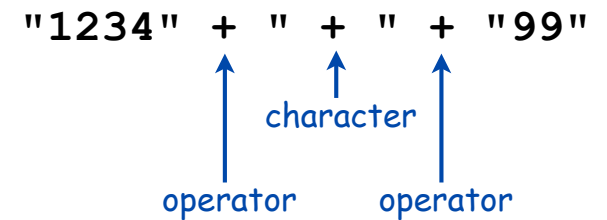
# Text

**String data type.** Useful for program input and output.

values	sequences of characters
typical literals	"Hello, " "1 " " * "
operation	concatenate
operator	+

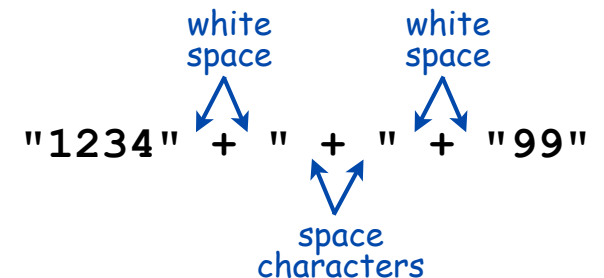
String data type

**Important note:** meaning of characters depends on context!



expression	value
"Hi, " + "Bob"	"Hi, Bob"
"1" + " 2 " + "1"	"1 2 1"
"1234" + " " + " " + "99"	"1234 + 99"
"1234" + "99"	"123499"

String concatenation examples





# Example: Subdivisions of a Ruler

```
public class Ruler
{
    public static void main(String[] args)
    {
        String ruler1 = "1";
        String ruler2 = ruler1 + " 2 " + ruler1;
        String ruler3 = ruler2 + " 3 " + ruler2;
        String ruler4 = ruler3 + " 4 " + ruler3;
        System.out.println(ruler4);
    }
}
```

"1"  
"1 2 1"  
"1 2 1 3 1 2 1"

string concatenation

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

1 2 1 3 1 2 1 4 1 2 1 3 1 2 1

# Integers

`int` data type. Useful for calculations, expressing algorithms.

values	integers between $-2^{31}$ and $+2^{31} - 1$				
typical literals	1234	99	-99	0	1000000
operations	add	subtract	multiply	divide	remainder
operators	+	-	*	/	%

`int` data type

expression	value	comment
5 + 3	8	
5 - 3	2	
5 * 3	15	
5 / 3	1	no fractional part
5 % 3	2	remainder
1 / 0		run-time error
3 * 5 - 2	13	* has precedence
3 + 5 / 2	5	/ has precedence
3 - 5 - 2	-4	left associative
( 3 - 5 ) - 2	-4	better style

examples of `int` operations

# Integer Operations

```
public class IntOps
{
    public static void main(String[] args)
    {
        int a = Integer.parseInt(args[0]);
        int b = Integer.parseInt(args[1]);
        int sum = a + b;
        int prod = a * b;
        int quot = a / b;
        int rem = a % b;
        System.out.println(a + " + " + b + " = " + sum);
        System.out.println(a + " * " + b + " = " + prod);
        System.out.println(a + " / " + b + " = " + quot);
        System.out.println(a + " % " + b + " = " + rem);
    }
}
```

command-line arguments

```
% javac IntOps.java
% java IntOps 1234 99
1234 + 99 = 1333
1234 * 99 = 122166
1234 / 99 = 12
1234 % 99 = 46
```

$$1234 = 12 * 99 + 46$$

Java automatically converts  
a, b, and rem to type String

# Floating-Point Numbers

`double` data type. Useful in scientific applications.

values	approximations to real numbers				
typical literals	3.14159	6.022e23	-3.0	2.0	1.4142135623730951
operations	add	subtract	multiply	divide	remainder
operators	+	-	*	/	%

`double` data type

expression	value
<code>3.141 + .03</code>	<code>3.171</code>
<code>3.141 - .03</code>	<code>3.111</code>
<code>6.02e23/2</code>	<code>3.01E+23</code>
<code>5.0 / 3.0</code>	<code>1.66666666666666700</code>
<code>10.0 % 3.141</code>	<code>0.577</code>
<code>1.0 / 0.0</code>	<code>Infinity</code>
<code>Math.sqrt(2.0)</code>	<code>1.4142135623731000</code>
<code>Math.sqrt(-1.0)</code>	<code>NaN</code>

special value

special value  
"not a number"

examples of `double` operations

# Excerpts from Java's Math Library

```
public class Math
```

---

<code>double abs(double a)</code>	absolute value of a	← also defined for int, long, and float
<code>double max(double a, double b)</code>	maximum of a and b	
<code>double min(double a, double b)</code>	minimum of a and b	
<code>double sin(double theta)</code>	sine function	← inverse functions asin(), acos(), and atan() also available
<code>double cos(double theta)</code>	cosine function	
<code>double tan(double theta)</code>	tangent function	
	← In radians. Use toDegrees() and toRadians() to convert.	
<code>double exp(double a)</code>	exponential ( $e^a$ )	
<code>double log(double a)</code>	natural log ( $\log_e a$ , or $\ln a$ )	
<code>double pow(double a, double b)</code>	raise a to the bth power ( $a^b$ )	
<code>long round(double a)</code>	found to the nearest integer	
<code>double random()</code>	random number in [0. 1)	
<code>double sqrt(double a)</code>	square root of a	
<code>double E</code>	value of e (constant)	
<code>double PI</code>	value of p (constant)	

# Quadratic Equation

Ex. Solve quadratic equation  $x^2 + bx + c = 0$ .

$$\text{roots} = \frac{-b \pm \sqrt{b^2 - 4c}}{2}$$

```
public class Quadratic
{
    public static void main(String[] args)
    {
        // Parse coefficients from command-line.
        double b = Double.parseDouble(args[0]);
        double c = Double.parseDouble(args[1]);

        // Calculate roots.
        double discriminant = b*b - 4.0*c;
        double d = Math.sqrt(discriminant);
        double root1 = (-b + d) / 2.0;
        double root2 = (-b - d) / 2.0;

        // Print them out.
        System.out.println(root1);
        System.out.println(root2);
    }
}
```

# Testing

Testing. Some valid and invalid inputs.

```
% java Quadratic -3.0 2.0
2.0
1.0
```

← command-line arguments

$$x^2 - 3x + 2$$

```
% java Quadratic -1.0 -1.0
1.618033988749895
-0.6180339887498949
```

← golden ratio

$$x^2 - x - 1$$

```
% java Quadratic 1.0 1.0
NaN
NaN
```

← "not a number"

$$x^2 + x + 1$$

```
% java Quadratic 1.0 hello
java.lang.NumberFormatException: hello
```

```
% java Quadratic 1.0
java.lang.ArrayIndexOutOfBoundsException
```

# Booleans

`boolean` data type. Useful to control logic and flow of a program.

values	true or false		
literals	true	false	
operations	and	or	not
operators	<code>&amp;&amp;</code>	<code>  </code>	<code>!</code>

`boolean` data type

<code>a</code>	<code>!a</code>	<code>a</code>	<code>b</code>	<code>a &amp;&amp; b</code>	<code>a    b</code>
<code>true</code>	<code>false</code>	<code>false</code>	<code>false</code>	<code>false</code>	<code>false</code>
<code>false</code>	<code>true</code>	<code>false</code>	<code>true</code>	<code>false</code>	<code>true</code>
		<code>true</code>	<code>false</code>	<code>false</code>	<code>true</code>
		<code>true</code>	<code>true</code>	<code>true</code>	<code>true</code>

Truth-table definitions of `boolean` operations



# Comparison Operators

## Comparison operators.

- Two operands of the same type.
- Result: a value of type `boolean`.

op	meaning	true	false
<code>==</code>	equal	<code>2 == 2</code>	<code>2 == 3</code>
<code>!=</code>	not equal	<code>3 != 2</code>	<code>2 != 2</code>
<code>&lt;</code>	less than	<code>2 &lt; 13</code>	<code>2 &lt; 2</code>
<code>&lt;=</code>	less than or equal	<code>2 &lt;= 2</code>	<code>3 &lt;= 2</code>
<code>&gt;</code>	greater than	<code>13 &gt; 2</code>	<code>2 &lt; 13</code>
<code>&gt;=</code>	greater than or equal	<code>3 &gt;= 2</code>	<code>2 &gt;= 3</code>

comparison operators

non-negative discriminant?

```
( b*b - 4.0*a*c ) >= 0.0
```

beginning of a century?

```
( year % 100 ) == 0
```

legal month?

```
( month >= 1 ) && ( month <= 12 )
```

comparison examples

# Leap Year

Q. Is a given year a leap year?

A. Yes if either (i) divisible by 400 or (ii) divisible by 4 but not 100.

```
public class LeapYear
{
    public static void main(String[] args)
    {
        int year = Integer.parseInt(args[0]);
        boolean isLeapYear;

        // divisible by 4 but not 100
        isLeapYear = (year % 4 == 0) && (year % 100 != 0);

        // or divisible by 400
        isLeapYear = isLeapYear || (year % 400 == 0);

        System.out.println(isLeapYear);
    }
}
```

```
% java LeapYear 2004
true
```

```
% java LeapYear 1900
false
```

```
% java LeapYear 2000
true
```

# Type Conversion

**Type conversion.** Convert from one type of data to another.

- Automatic (done by Java when no loss of precision; or with strings).
- Explicitly defined by function call.
- Cast (write desired type within parens).

expression	type	value	
"1234" + 99	String	"123499"	automatic
Integer.parseInt("123")	int	123	explicit
(int) 2.71828	int	2	cast
Math.round(2.71828)	long	3	explicit
(int) Math.round(2.71828)	int	3	cast
(int) Math.round(3.14159)	int	3	cast
11 * 0.3	double	3.3	automatic
(int) 11 * 0.3	double	3.3	cast, automatic
11 * (int) 0.3	int	0	cast
(int) (11 * 0.3)	int	3	cast, automatic

# TEQ on Type Conversion

[not difficult if you read Exercise 1.2.6]

What is the type and value of each of the following expression?

A.  $(7 / 2) * 2.0$

B.  $(7 / 2.0) * 2$

# Type Conversion Example: Random Integer

Ex. Generate a pseudo-random number between 0 and  $N-1$ .

```
public class RandomInt
{
    public static void main(String[] args)
    {
        int N = Integer.parseInt(args[0]);
        double r = Math.random();
        int n = (int) (r * N);
        System.out.println("random integer is " + n);
    }
}
```

String to int (method)

double between 0.0 and 1.0

double to int (cast)

int to double (automatic)

int to String (automatic)

```
% java RandomInt 6
random integer is 3

% java RandomInt 6
random integer is 0

% java RandomInt 10000
random integer is 3184
```

# Summary

A **data type** is a set of values and operations on those values.

- **String** text processing, input and output.
- **double, int** mathematical calculation.
- **boolean** decision making.

Be aware. In Java you must:

- Declare type of values.
- Convert between types when necessary.

Why do we need types?

- Type conversion must be done at some level.
- Compiler can help do it correctly.
- Example: In 1996, Ariane 5 rocket exploded after takeoff because of bad type conversion.



Example of bad type conversion

