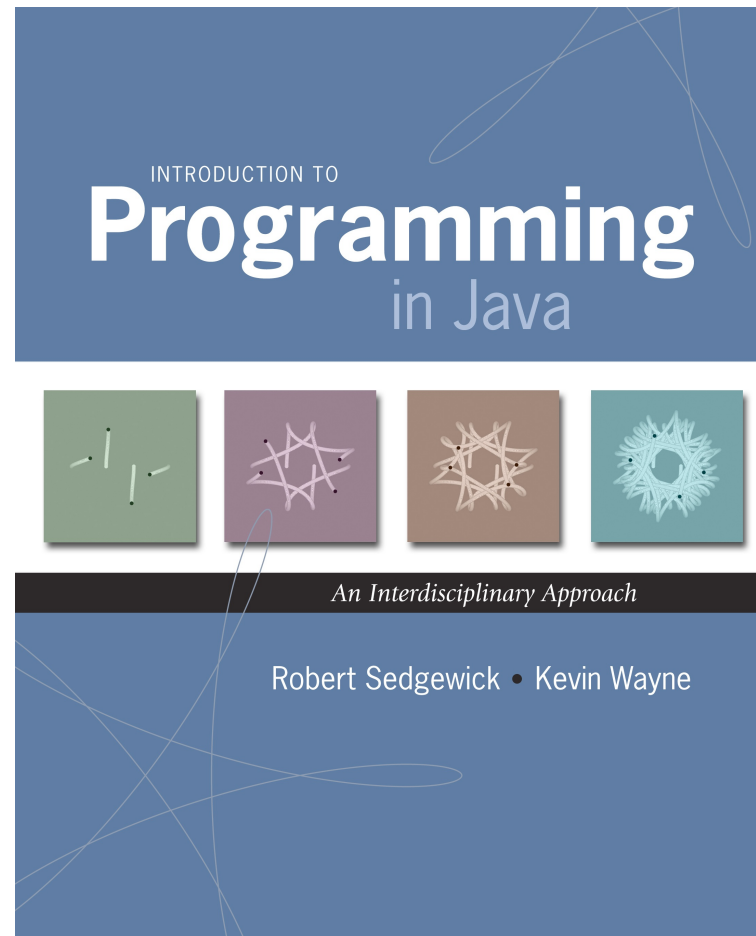


# 1.1 Your First Program

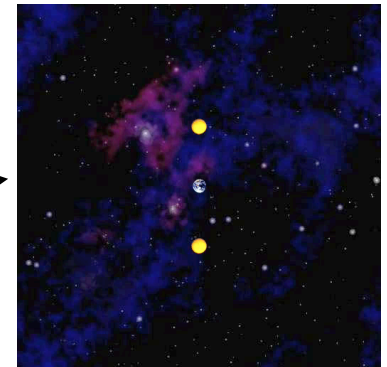
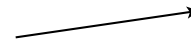


# Why Programming?

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

Naive ideal. Natural language instructions.

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



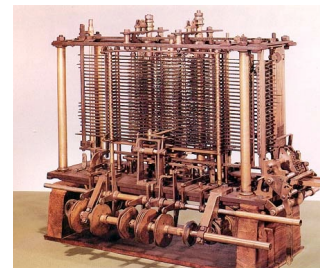
Prepackaged solutions (apps)? Great, when what they do is what you want.



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 compromise.

*“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,  
10 million developers  
billions of devices

- 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>

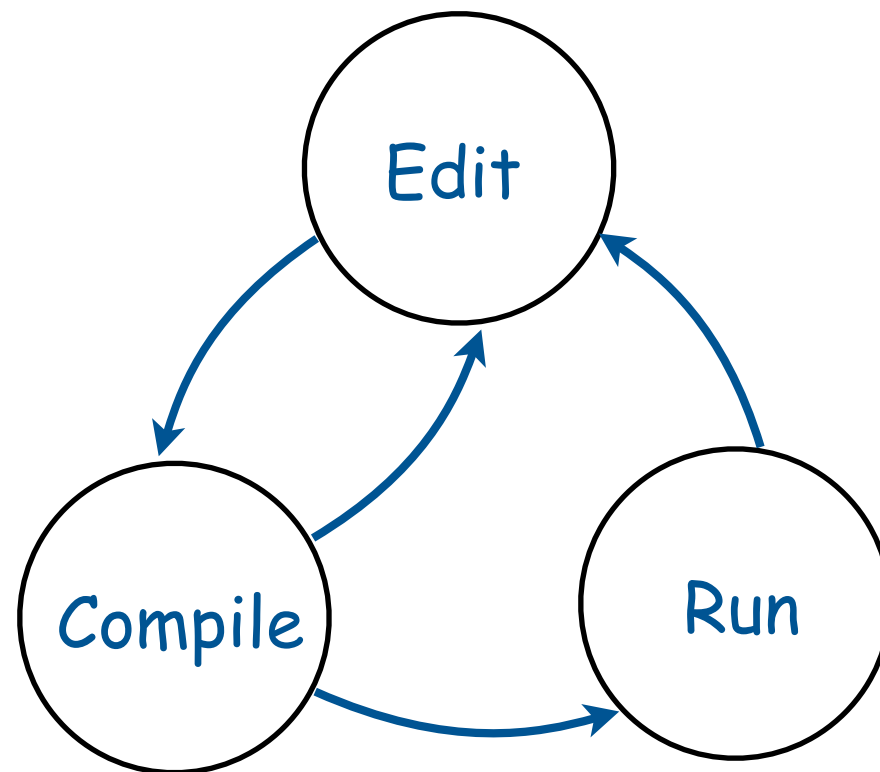
String	
<code>+</code>	<code>""</code>
<code>length()</code>	<code>compareTo()</code>
<code>charAt()</code>	<code>matches()</code>

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

---





# Programming in Java

## Programming in Java.

- **Create** the program by typing it into a text editor, and save it as HelloWorld.java.

```
/*  
 * Prints "Hello, World"  
 * Everyone's first Java program.  
 */  
  
public class HelloWorld {  
    public static void main(String[] args) {  
        System.out.println("Hello, World");  
    }  
}
```


HelloWorld.java

# Programming in Java

## Programming in Java.

- Create the program by typing it into a text editor, and save it as HelloWorld.java.
- **Compile** it by typing at the command-line:  
javac HelloWorld.java.

command-line



```
% javac HelloWorld.java
```

- This creates a Java bytecode file named: HelloWorld.class.

# Programming in Java

## Programming in Java.

- Create the program by typing it into a text editor, and save it as `HelloWorld.java`.
- Compile it by typing at the command-line:  
`javac HelloWorld.java`.
- **Execute** it by typing at the command-line:  
`java HelloWorld`.

command-line

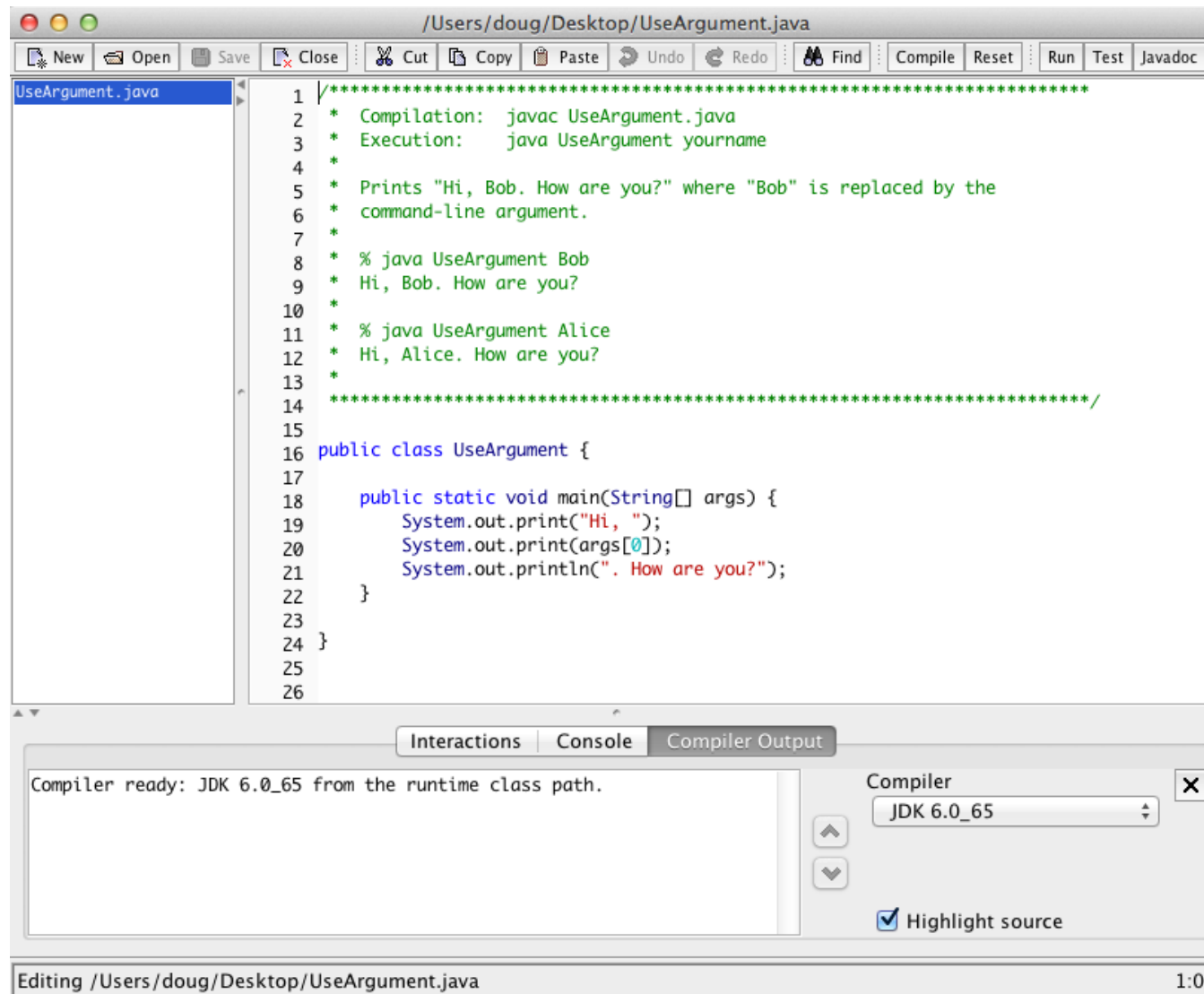
```
% javac HelloWorld.java  
  
% java HelloWorld  
Hello, World
```

# 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.



← text editor

# 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.

The screenshot shows the DrJava IDE interface. The main window displays the source code for `UseArgument.java`. The code includes a class definition and a `main` method that prints "Hi, Bob. How are you?" and "Hi, Alice. How are you?". The `Compile` button in the toolbar is highlighted in red. A red arrow points to this button with the label "compile button". Another black arrow points from the `Compile` button to the `UseArgument.class` file in the file explorer, with the label "creates UseArgument.class".

```
1 /*****
2  * Compilation: javac UseArgument.java
3  * Execution:   java UseArgument yourname
4  *
5  * Prints "Hi, Bob. How are you?" where "Bob" is replaced by the
6  * command-line argument.
7  *
8  * % java UseArgument Bob
9  * Hi, Bob. How are you?
10 *
11 * % java UseArgument Alice
12 * Hi, Alice. How are you?
13 *
14 *****/
15
16 public class UseArgument {
17
18     public static void main(String[] args) {
19         System.out.print("Hi, ");
20         System.out.print(args[0]);
21         System.out.println(". How are you?");
22     }
23
24 }
25
26
```

Compiler ready: JDK 6.0\_65 from the runtime class path.

Compiler: JDK 6.0\_65

Highlight source

Editing /Users/doug/Desktop/UseArgument.java 1:0

# 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 multi-line comment with instructions for compilation and execution, and a `main` method that prints "Hi, " followed by the first command-line argument and ". How are you?". The `Run` button in the toolbar is highlighted with a red arrow. Below the code editor, the `Console` tab is active, showing the output of two `java` commands: `java UseArgument Batman` and `java UseArgument Robin`, both resulting in "Hi, [name]. How are you?".

```
1 /*****
2  * Compilation: javac UseArgument.java
3  * Execution:   java UseArgument yourname
4  *
5  * Prints "Hi, Bob. How are you?" where "Bob" is replaced by the
6  * command-line argument.
7  *
8  * % java UseArgument Bob
9  * Hi, Bob. How are you?
10 *
11 * % java UseArgument Alice
12 * Hi, Alice. How are you?
13 *
14 *****/
15
16 public class UseArgument {
17
18     public static void main(String[] args) {
19         System.out.print("Hi, ");
20         System.out.print(args[0]);
21         System.out.println(". How are you?");
22     }
23
24 }
25
26
```

```
Welcome to DrJava. Working directory is /Users/doug/Desktop
> java UseArgument Batman
Hi, Batman. How are you?
> java UseArgument Robin
Hi, Robin. How are you?
> |
```

Alternative 1:  
run button  
(OK if no args)

both use  
`UseArgument.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

### 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:

- Let the Doctor indent for you.
- Correct any style problems automatically discovered when you submit.
- Follow your preceptor/grader's advice on style.



## 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

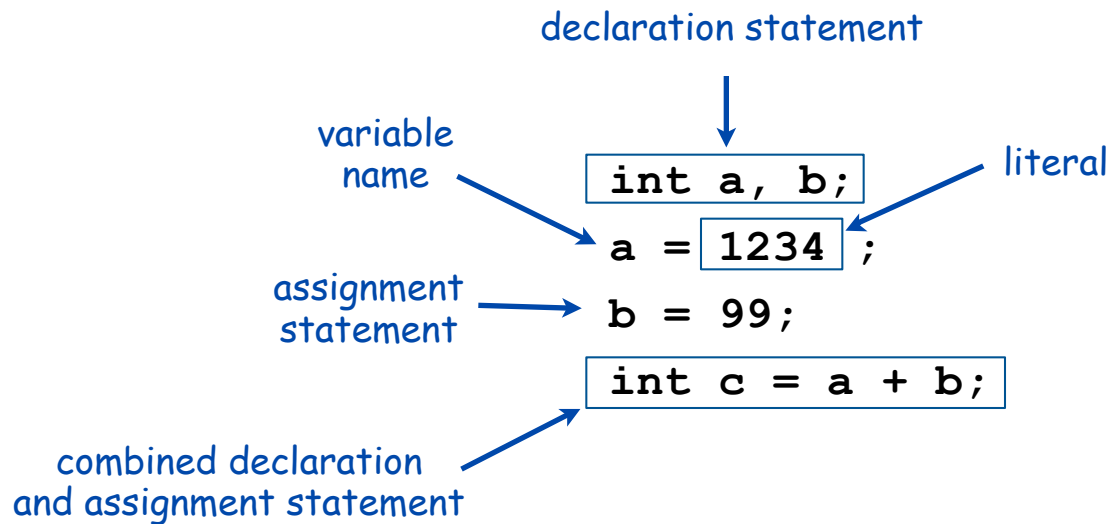
# Basic Definitions

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

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

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

**Program.** Sequence of statements.



# 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

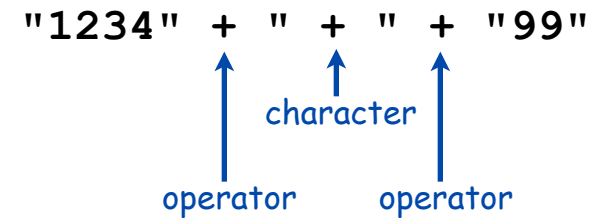
# Text

String data type. Useful for program input and output.

## String data type

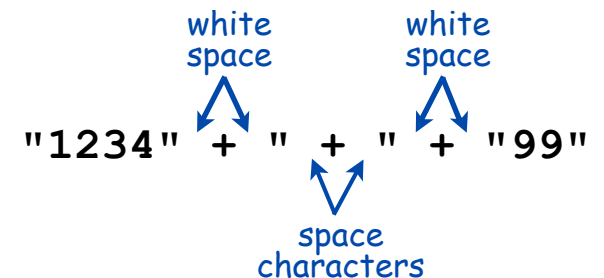
values	sequences of characters
typical literals	"Hello, " "1 " " * "
operation	concatenate
operator	+

Important note: meaning of characters depends on context!



## String concatenation examples

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



# 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
```



# Integers

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

there is a largest `int`  
and a smallest `int`

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

## examples of `int` operations

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

# 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.

`double` data type

there is a largest double  
and a smallest double

values  
typical literals  
operations  
operators

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

examples of `double` operations

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>1.666666666666667</code>	
<code>10.0 % 3.141</code>	<code>0.577</code>	
<code>1.0 / 0.0</code>	<code>Infinity</code>	← special value
<code>Math.sqrt(2.0)</code>	<code>1.4142135623731</code>	
<code>Math.sqrt(-1.0)</code>	<code>NaN</code>	← special value "not a number"

# Excerpts from Java's Math Library

```
public class Math
```

---

```
double abs(double a)
```

absolute value of a

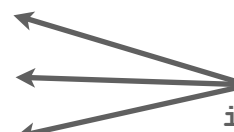
```
double max(double a, double b)
```

maximum of a and b

```
double min(double a, double b)
```

minimum of a and b

also defined for  
int, long, and float



```
double sin(double theta)
```

sine function

```
double cos(double theta)
```

cosine function

```
double tan(double theta)
```

tangent function

inverse functions  
asin(), acos(), and atan()  
also available

In radians. Use toDegrees() and toRadians() to convert.



```
double exp(double a)
```

exponential (e

```
double log(double a)
```

natural log (log

```
double pow(double a, double b)
```

raise a to the bth power (a

```
long round(double a)
```

found to the nearest integer

```
double random()
```

random number in [0. 1)

```
double sqrt(double a)
```

square root of a

```
double E
```

value of e (constant)

```
double PI
```

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.

`boolean` data type

values	true or false		
literals	true	false	
operations	and	or	not
operators	&&		!

Truth-table definitions of `boolean` operations

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

# Comparison Operators

## Comparison operators.

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

### comparison operators

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 &gt; 13</code>
<code>&gt;=</code>	greater than or equal	<code>3 &gt;= 2</code>	<code>2 &gt;= 3</code>

### comparison examples

non-negative discriminant? `( b * b - 4.0 * a * c ) >= 0.0`

beginning of a century? `( year % 100 ) == 0`

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

# 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 2016
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, explicit
(int) Math.round(3.14159)	int	3	cast, explicit
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



# 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, explicit
(int) Math.round(3.14159)	int	3	cast, explicit
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
(int) 0.3 * 11	?	?	?

# 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, explicit
(int) Math.round(3.14159)	int	3	cast, explicit
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
(int) 0.3 * 11	int	0	cast



Pay attention to the type of your data.

← type conversion can give counterintuitive results  
but gets easier to understand with practice

# 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.

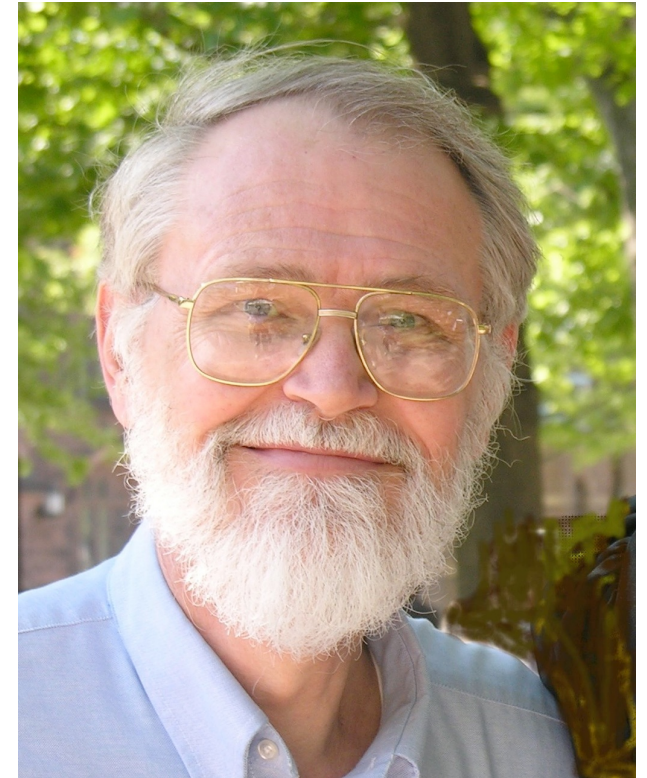


Example of bad type conversion



# The Very First HelloWorld

```
main( ) {  
    printf("hello, world");  
}
```



Brian Kernighan (today)