CMSC 150
INTRODUCTION TO COMPUTING

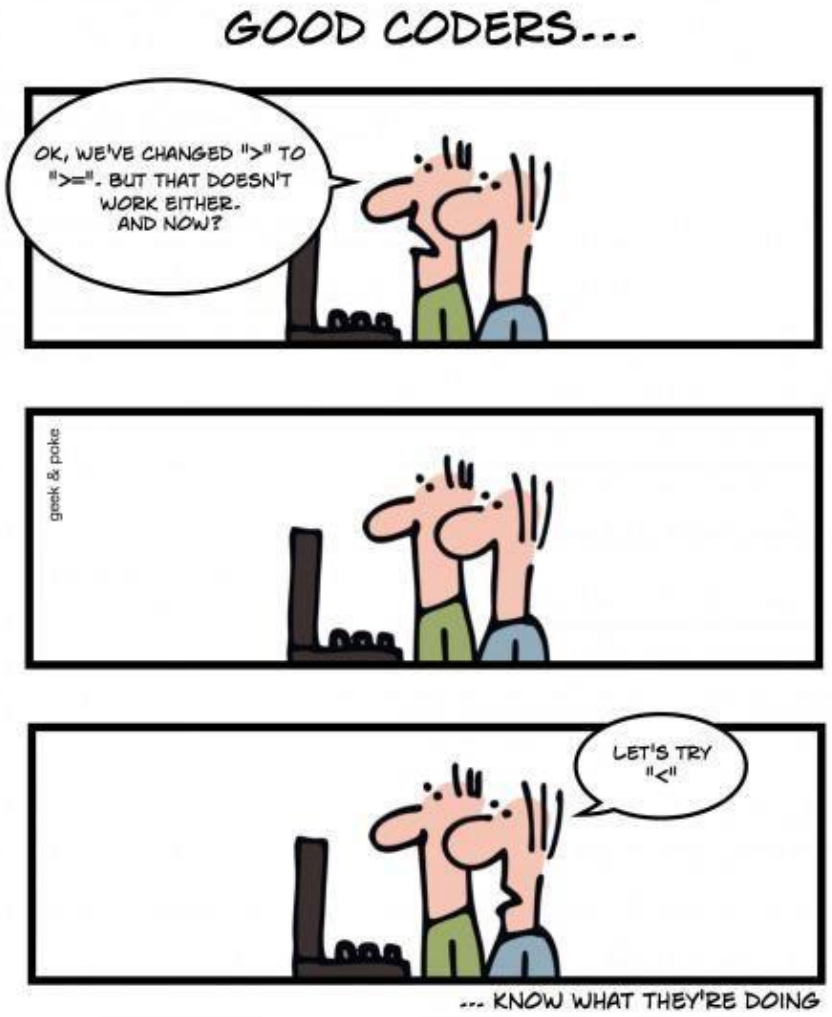
ACKNOWLEDGEMENT: THESE SLIDES ARE ADAPTED FROM SLIDES PROVIDED WITH INTRODUCTION TO PROGRAMMING IN JAVA: AN INTERDISCIPLINARY APPROACH, SEDGEWICK AND WAYNE (PEARSON ADDISON-WESLEY 2007)

LECTURE 2

- DATA!
- PRIMITIVE TYPES AND OPERATIONS ON THEM
- MATH, STRING

DATA

- At the lowest level – literally combination of 0s and 1s
- We need to think on a higher level to make programming easier. Java lets us do this.
  - We can think more ‘mathematically’ and more ‘logically’ compared with just 0s and 1s
  - Uses integral numbers, real numbers, true/false values, and characters



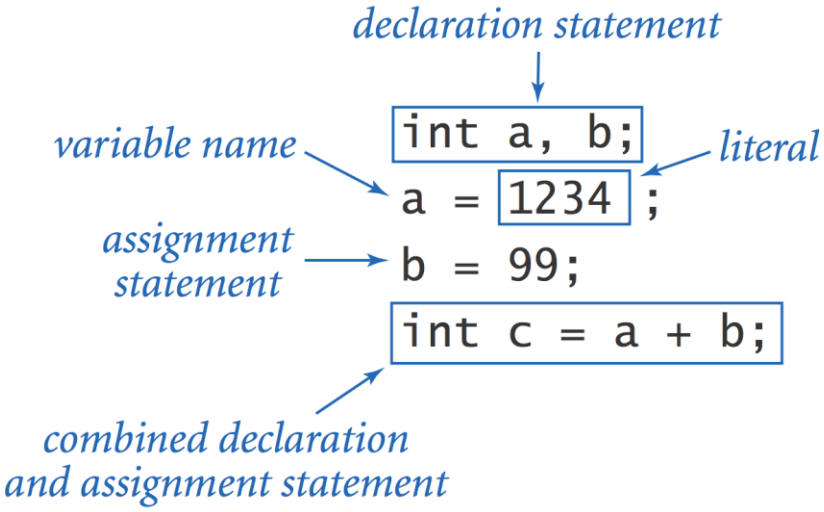
BUILT-IN DATA TYPES

- Data type. A set of values and operations defined on those values.
- Operations are completed using operators (e.g., '+')

Type	Set of values	Literal values	Example Operations
`char`	characters	`'A'` `'@'`	Compare (<, >, =)
`String`	sequences of characters	`"Hello World"` `"126 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

- Type. A set of values and allowable operations.
- Variable. A name that refers to a value of a type.
- Declaration. Notifying the compiler that a variable exists.
  - Must be done before a variable can be used!
- Literal. Programming language representation of a value.
- Assignment statement. Associates a value with a variable. Operator '='
- Statement. "A line of code". ends with ';'
- Expression. A combination of one or more variables, values, operators, and functions which produce another value. Example: a + b produces a new value.



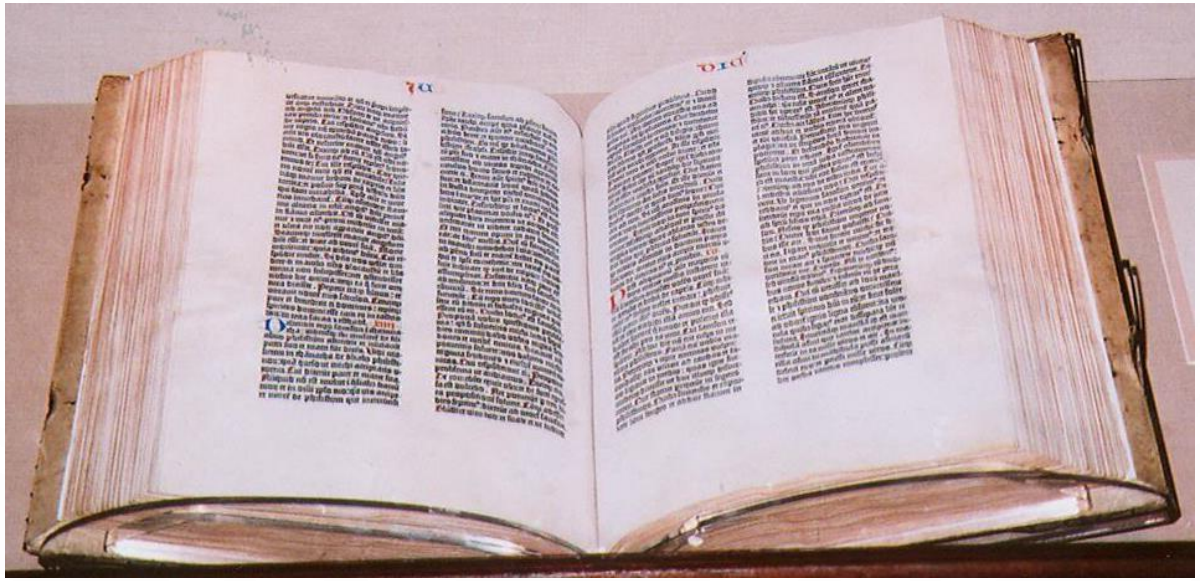
ACTIVITY - TRACE

- With a partner, create a table of variable values after each statement.
- This is called a Trace

```
int a, b;
a = 1234;
b = 99;
int t = a;
a = b;
b = t;
```

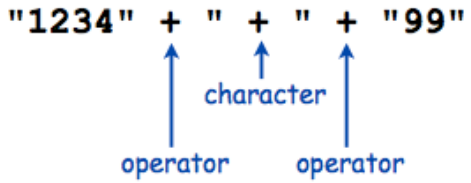
a	b	t
Answer to be revealed!		

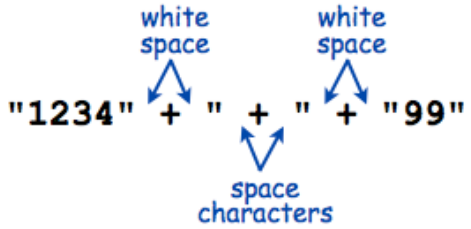
TEXT



TEXT

- String data type. A sequence of characters. Useful for program input and output.
- Caveat. Meaning of characters depends on context.





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

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

CONCATENATION EXAMPLE – SUBDIVISIONS OF A RULER


```
1. public class Ruler {
2.     public static void main(String[] args) {
3.         String ruler1 = "1";                    // "1"
4.         String ruler2 = ruler1 + " 2 " + ruler1;    // "1 2 1"
5.         String ruler3 = ruler2 + " 3 " + ruler2;    // "1 2 1 3 1 2 1"
6.         String ruler4 = ruler3 + " 4 " + ruler3;    // "1 2 1 3 1 2 1 4 1 2 1 3 1 2 1"
7.         System.out.println(ruler4);
8.     }
9. }
```

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

INTEGERS

…, -5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5, …

INTEGERS

- **int** data type. Integral values. Useful for expressing mathematics.

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

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
`3 - ( 5 - 2 )`	0	unambiguous

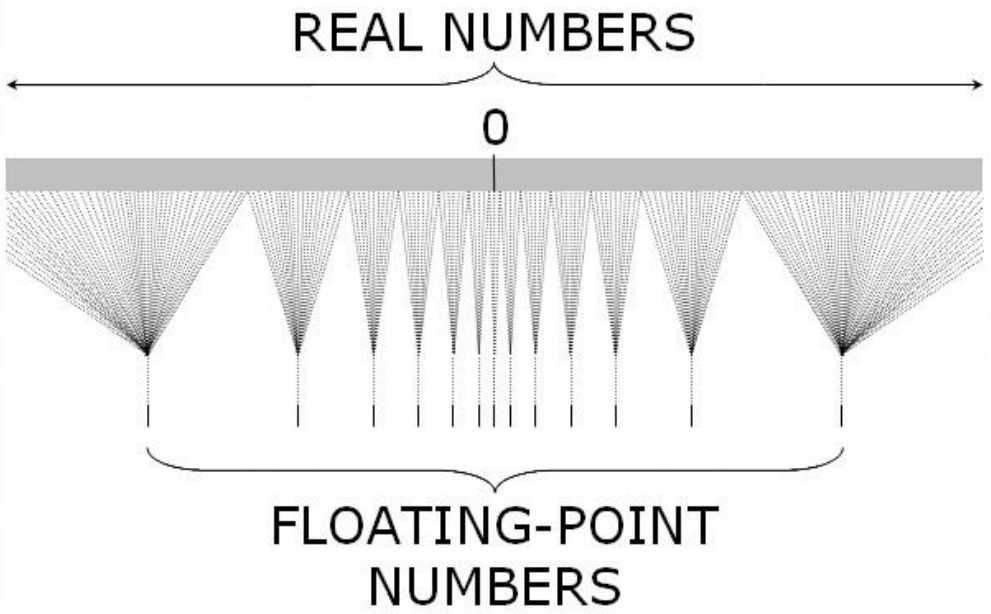
INTEGER OPERATIONS

```
1.  public class IntOps {
2.      public static void main(String[] args) {
3.          int a = Integer.parseInt(args[0]); //First command line argument
4.          int b = Integer.parseInt(args[1]); //Second command line argument
5.          int sum  = a + b;
6.          int prod = a * b;
7.          int quot = a / b;
8.          int rem  = a % b;
9.          //Java will automatically convert int variables to String
10.         System.out.println(a + " + " + b + " = " + sum);
11.         System.out.println(a + " * " + b + " = " + prod);
12.         System.out.println(a + " / " + b + " = " + quot);
13.         System.out.println(a + " % " + b + " = " + rem);
14.     }
15. }
```

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

1234 = 12*99 + 46

FLOATING-POINT NUMBERS



FLOATING-POINT NUMBERS

- **double** data type. Real numbers. Useful in scientific applications.

values	real numbers (specified by IEEE 754 standard)			
typical literals	`3.14159`	`6.022e23`	`-3.0`	`2.0` `1.414213562373095`
operations	add	subtract	multiply	divide
operators	+	-	*	/

expression	*value*
`3.141 + .03`	`3.171`
`3.141 - .03`	`3.111`
`6.02e23 / 2.0`	`3.01e23`
`5.0 / 3.0`	`1.6666666666666667`
`10.0 % 3.141`	`0.577`
`1.0 / 0.0`	`Infinity`
`Math.sqrt(2.0)`	`1.4142135623730951`
`Math.sqrt(-1.0)`	`NaN`

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*

Note 1: `abs()`, `max()`, *and* `min()` *are defined also for* `int`, `long`, *and* `float`.

`double sin(double theta)`	*sine function*
`double cos(double theta)`	*cosine function*
`double tan(double theta)`	*tangent function*

Note 2: Angles are expressed in radians. Use `toDegrees()` *and* `toRadians()` *to convert.*
Note 3: Use `asin()`, `acos()`, *and* `atan()` *for inverse functions.*

`double exp(double a)`	*exponential* (e^a)
`double log(double a)`	*natural log* ($\log_e a$, *or* $\ln a$)
`double pow(double a, double b)`	*raise a to the bth power* (a^b)
`long round(double a)`	*round 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 π (constant)*

QUADRATIC EQUATION

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

```
1.  public class Quadratic {
2.     public static void main(String[] args) {

3.        // parse coefficients from command-line
4.        double b = Double.parseDouble(args[0]);
5.        double c = Double.parseDouble(args[1]);

6.        // calculate roots
7.        double discriminant = b*b - 4.0*c;
8.        double d = Math.sqrt(discriminant);
9.        double root1 = (-b + d) / 2.0;
10.       double root2 = (-b - d) / 2.0;

11.       // print them out
12.       System.out.println(root1);
13.       System.out.println(root2);
14.    }
15. }
```

% **java Quadratic –3.0 2.0** **Here – command line inputs**
2.0
1.0

% **java Quadratic –1.0 –1.0**
1.618033988749895
-0.6180339887498949

% **java Quadratic 1.0 1.0**
NaN NaN – Not a Number
NaN

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

% **java Quadratic 1.0**
java.lang.ArrayIndexOutOfBoundsException

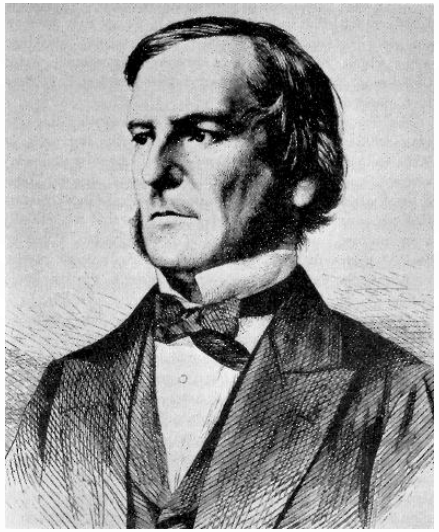
OPERATOR PRECEDENCE (ORDERING)

- Operator ordering has defined precedence
- Can impose your own with () in an expression
  - $a * b + c$ vs $a * (b + c)$

Operators	Associativity
[] . () (method call)	Left to right
! ~ ++ -- + (unary) - (unary) () (cast) new	Right to left
* / %	Left to right
+ -	Left to right
<< >> >>>	Left to right
< <= > >= instanceof	Left to right
== !=	Left to right
&	Left to right
^	Left to right
|	Left to right
&&	Left to right
||	Left to right
?:	Right to left
= += -= *= /= %= &= |= ^= <<= >>= >>>=	Right to left

BOOLEANS

TO BE OR NOT TO BE?



BOOLEANS

- Boolean data type. Useful to control logic and flow of a program.

  Reminiscent of '0's and '1's…

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

a	!a
true	false
false	true

a	b	a && b	a || b
false	false	false	false
false	true	false	true
true	false	false	true
true	true	true	true

ACTIVITY – BOOLEANS

- Make the truth table for the following. () help define order of operations.
  ((A && !B) || (B && !A))
- What combinations of A, B, C make the following expression true?
  ((A || C) && (!A && B))

COMPARISON OPERATORS

- Comparisons. Take two operands of one type (e.g., int) and produce a result of type boolean.

op	*meaning*	`true`	`false`
`==`	*equal*	`2 == 2`	`2 == 3`
`!=`	*not equal*	`3 != 2`	`2 != 2`
`<`	*less than*	`2 < 13`	`2 < 2`
`<=`	*less than or equal*	`2 <= 2`	`3 <= 2`
`>`	*greater than*	`13 > 2`	`2 > 13`
`>=`	*greater than or equal*	`3 >= 2`	`2 >= 3`

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

```
1.  public class LeapYear {
2.      public static void main(String[] args) {
3.          int year = Integer.parseInt(args[0]);
4.          boolean isLeapYear;
5.          // divisible by 4 but not 100
6.          isLeapYear = (year % 4 == 0) && (year % 100 != 0);
7.
8.          // or divisible by 400
9.          isLeapYear = isLeapYear || (year % 400 == 0);
10.         System.out.println(isLeapYear);
11.     }
12.}
```

```
% java LeapYear 2004
true
% java LeapYear 1900
false
% java LeapYear 2000
true
```

TYPE CONVERSION

TYPE CONVERSION

- Type conversion. Convert value from one data type to another.
  - Automatic: no loss of precision; or with strings.
  - Explicit: cast; or method.

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

RANDOM INTEGER BETWEEN 0 AND $n - 1$

```
1. public class RandomInt {
2.     public static void main(String[] args) {
3.         int N = Integer.parseInt(args[0]); // args[0] converted from String to int
4.         double r = Math.random(); // Get double between 0.0 and 1.0
5.         int n = (int) (r * N); // First, N is converted to double implicitely
6.                                // Second, the result is converted explicitely to int
7.         System.out.println("random integer is " + n); // n converted from int to
   String
8.     }
9. }
```

```
% 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.
  - **double, int** mathematical calculation.
  - **boolean** decision making.
- In Java, you must:
  - Declare type of values.
  - Convert between types when necessary. However, can cause
    disastrous errors, e.g. Ariane 5 rocket
- Why do we need types?
  - Represent things! Perform computations on things!

- example of bad type conversion