1.2 Built-in Data Types

- strings
- integers
- floating-point numbers
- booleans
- type conversion
Questions during (or after) lecture

- raise your hand and ask
- ask on Ed
- attend office hours (or stay after lecture)
A data type (type) is a set of values and a set of operations on those values.

<table>
<thead>
<tr>
<th>type</th>
<th>set of values</th>
<th>example values</th>
<th>examples of operations</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>int</strong></td>
<td>integers</td>
<td>17, -12345</td>
<td>add, subtract, multiply, divide, compare, equality</td>
</tr>
<tr>
<td><strong>double</strong></td>
<td>floating-point numbers</td>
<td>2.5, -0.125</td>
<td>add, subtract, multiply, divide, compare, equality</td>
</tr>
<tr>
<td><strong>boolean</strong></td>
<td>truth values</td>
<td>true, false</td>
<td>and, or, not, equality</td>
</tr>
<tr>
<td><strong>String</strong></td>
<td>sequences of characters</td>
<td>&quot;Hello, World&quot;, &quot;COS 126 is fun&quot;</td>
<td>concatenate</td>
</tr>
</tbody>
</table>
Programming terminology

**Program.** Sequence of statements.  

**Declaration statement.** Associates a variable with a name and type.

**Variable.** A storage location for a data-type value.

**Assignment statement.** Stores a value in a variable.

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

**Expression.** A combination of variable names, literals, operators, etc. that evaluates to a value.
Assignment statements

Q. How does an assignment statement work?

A. Java evaluates the expression on the RHS and assigns that value to the variable on the LHS.

expression type must be compatible with variable type

assignment operator = means assignment (not math equality!)

variable name evaluates to value stored in variable

int a;
int b;
a = 1234;
b = 99;
int c = a + b;

variable of type int
expression of type int

expression type must be compatible with variable type

variables

a  b  c
1234 99 1333
Valid and invalid assignment statements

Q. Which of these independent code fragments are valid?

<table>
<thead>
<tr>
<th>statements</th>
<th>compiles?</th>
<th>remark</th>
</tr>
</thead>
<tbody>
<tr>
<td>int a = 1; 123 = a;</td>
<td>☹</td>
<td>LHS is not a variable (= does not mean math equality)</td>
</tr>
<tr>
<td>double a = 2.5; int b = a;</td>
<td>☹</td>
<td>RHS type is incompatible with LHS type</td>
</tr>
<tr>
<td>String s = 123;</td>
<td>☹</td>
<td>RHS type is incompatible with LHS type</td>
</tr>
<tr>
<td>int b = 2; int a = 3 * b;</td>
<td>😍</td>
<td>RHS can be an expression</td>
</tr>
<tr>
<td>int a = 3; a = 2 * a;</td>
<td>😍</td>
<td>a variable can be reassigned (that’s why it’s called a variable!)</td>
</tr>
<tr>
<td>int a = 2 * a;</td>
<td>😞</td>
<td>a variable must be assigned a value before it can be used in an expression</td>
</tr>
</tbody>
</table>
Tracing the execution of a program

Q. What does this code fragment do?
A. Let's trace the variables during execution of the code.

<table>
<thead>
<tr>
<th>start of code fragment</th>
<th>a</th>
<th>b</th>
<th>temp</th>
</tr>
</thead>
<tbody>
<tr>
<td>declared</td>
<td>1234</td>
<td>undeclared</td>
<td>undeclared</td>
</tr>
</tbody>
</table>

| int a = 1234;          |      |      |      |
| int b = 99;            |      |      |      |
| int temp = a;          |      |      |      |
| a = b;                 |      |      |      |
| b = temp;              |      |      |      |

*trace of variables (after each statement)*

*this idiom exchanges the values stored in the variables a and b*
What are the values stored in the variables \( a \) and \( b \) after the code fragment is executed?

A. 1234 and 99.

int a = 1234;
int b = 99;
a = a + b;
b = a - b;
a = a - b;

B. 99 and 1234.

C. 1333 and 1135.

D. 1135 and 1135.

E. Compile-time error.

non-idiomatic code that exchanges the values stored in the variables \( a \) and \( b \)
**Built-In Data Types**

- strings
- integers
- floating-point numbers
- booleans
- type conversion
The *String* data type

**Typical usage.** Program input and output; text processing.

<table>
<thead>
<tr>
<th>values</th>
<th>sequences of characters</th>
</tr>
</thead>
<tbody>
<tr>
<td>typical literals</td>
<td>&quot;Hi&quot; &quot;1234&quot; &quot;Nǐ hāo&quot; &quot;💩💩💩&quot;</td>
</tr>
<tr>
<td>operations</td>
<td>concatenation</td>
</tr>
<tr>
<td>operator</td>
<td>+</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>expression</th>
<th>value</th>
<th>remark</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;My &quot; + &quot;Precious&quot;</td>
<td>&quot;My Precious&quot;</td>
<td>spaces within a string literal matter</td>
</tr>
<tr>
<td>&quot;1234&quot; + &quot;99&quot;</td>
<td>&quot;123499&quot;</td>
<td>strings are not integers</td>
</tr>
<tr>
<td>&quot;A&quot; + &quot;B&quot; + &quot;C&quot;</td>
<td>&quot;ABC&quot;</td>
<td>can concatenate several strings together, in one expression</td>
</tr>
<tr>
<td>&quot;አበም &quot; + &quot;አዐለ吠!&quot;</td>
<td>&quot;አበም አዐለ吠!&quot;</td>
<td>Unicode supported</td>
</tr>
</tbody>
</table>
Command-line arguments are strings

**Command-line arguments.** The variables `args[0]`, `args[1]`, `args[2]`, ... are of type `String`.

```java
public class CommandLineArguments {
    public static void main(String[] args) {
        String a = args[0];
        String b = args[1];
        String c = args[2];
        String result = a + "-" + b + "-" + c;
        System.out.println(result);
    }
}
```

~/cos26/datatypes> java CommandLineArguments A B C
A-B-C

~/cos26/datatypes> java CommandLineArguments do re mi
do-re-mi

~/cos26/datatypes> java CommandLineArguments
Exception in thread "main"
java.lang.ArrayIndexOutOfBoundsException:
Index 0 out of bounds for length 0 at
CommandLineArguments.main(CommandLineArguments.java:3)
## Ruler function

```java
public class Ruler {
    public static void main(String[] args) {
        String ruler0 = "0";
        String ruler1 = ruler0 + " 1 " + ruler0;
        String ruler2 = ruler1 + " 2 " + ruler1;
        String ruler3 = ruler2 + " 3 " + ruler2;
        System.out.println(ruler3);
    }
}
```

<table>
<thead>
<tr>
<th>ruler0</th>
<th>ruler1</th>
<th>ruler2</th>
<th>ruler3</th>
</tr>
</thead>
<tbody>
<tr>
<td>undeclared</td>
<td>undeclared</td>
<td>undeclared</td>
<td>undeclared</td>
</tr>
<tr>
<td>&quot;0&quot;</td>
<td>undeclared</td>
<td>undeclared</td>
<td>undeclared</td>
</tr>
<tr>
<td>&quot;0&quot;</td>
<td>&quot;0 1 0&quot;</td>
<td>undeclared</td>
<td>undeclared</td>
</tr>
<tr>
<td>&quot;0&quot;</td>
<td>&quot;0 1 0&quot;</td>
<td>&quot;0 1 0 2 0 1 0&quot;</td>
<td>undeclared</td>
</tr>
<tr>
<td>&quot;0&quot;</td>
<td>&quot;0 1 0&quot;</td>
<td>&quot;0 1 0 2 0 1 0&quot;</td>
<td>&quot;0 1 0 2 0 1 0 3 0 1 0 2 0 1 0&quot;</td>
</tr>
</tbody>
</table>

**trace of variables (after each statement)**

```
>`/cos126/datatypes> java Ruler
0 1 0 2 0 1 0 3 0 1 0 2 0 1 0
```
**Built-in Data Types**

- strings
- integers
- floating-point numbers
- booleans
- type conversion

https://introcs.cs.princeton.edu
The `int` data type

**Typical usage:** math calculations involving integers; program control flow.

<table>
<thead>
<tr>
<th>values</th>
<th>$\text{integers between } -2^{31} \text{ and } 2^{31} - 1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>typical literals</td>
<td>1234 99 0 1000000 -3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>operations</th>
<th>add</th>
<th>subtract</th>
<th>multiply</th>
<th>divide</th>
<th>remainder</th>
</tr>
</thead>
<tbody>
<tr>
<td>operators</td>
<td>+</td>
<td>-</td>
<td>*</td>
<td>/</td>
<td>%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>expression</th>
<th>value</th>
<th>remark</th>
</tr>
</thead>
<tbody>
<tr>
<td>20 + 3</td>
<td>23</td>
<td></td>
</tr>
<tr>
<td>20 - 3</td>
<td>17</td>
<td></td>
</tr>
<tr>
<td>20 * 3</td>
<td>60</td>
<td></td>
</tr>
<tr>
<td>20 / 3</td>
<td>6</td>
<td>drop fractional part</td>
</tr>
<tr>
<td>20 % 3</td>
<td>2</td>
<td>remainder</td>
</tr>
<tr>
<td>20 / 0</td>
<td>-</td>
<td>division-by-zero error</td>
</tr>
<tr>
<td>$2^{31} - 1$ + 1</td>
<td>-2147483648</td>
<td>integer overflow</td>
</tr>
</tbody>
</table>

**only $2^{32}$ different int values**

(not quite the same as integers)

- **typical literals**
- **operations**
- **operators**

- **expression**
- **value**
- **remark**

- applying an int operator to two int operands always results in an int (or division-by-zero error)
- drop fractional part
- remainder
- division-by-zero error
- integer overflow

**don’t use int with very large integers**

- only $2^{32}$ different int values (not quite the same as integers)

- **values**
- **typical literals**
- **operations**
- **operators**

- **expression**
- **value**
- **remark**
Input and output

Java I/O model. [for now]

- Read strings from the command line.
- Print strings to standard output.

Q. How to read integers from the command line?
A. The system method `Integer.parseInt()` converts from a `String` to an `int`.

Q. How to print integers to standard output?
A. When a `String` is concatenated with an `int`, Java converts the `int` to a `String`.
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);
    }
}

converts from
int to String

converts from
String to int

~/cos126/datatypes> java IntOps 20 3
20 + 3 = 23
20 * 3 = 60
20 / 3 = 6
20 % 3 = 2

~/cos126/datatypes> java IntOps 1234 99
1234 + 99 = 1333
1234 * 99 = 122166
1234 / 99 = 12
1234 % 99 = 46

~/cos126/datatypes> java IntOps Hello 123
Exception in thread "main"
java.lang.NumberFormatException:
For input string: "Hello"
Order of operations

**PEMDAS.** Rules for evaluating an arithmetic expression.

**Operator precedence.** Priority for grouping operands with operators in an expression.

**Operator associativity.** Rule when two operators in an expression have same priority.

<table>
<thead>
<tr>
<th>expression</th>
<th>equivalent to</th>
<th>value</th>
<th>remark</th>
</tr>
</thead>
<tbody>
<tr>
<td>3 * 5 – 2</td>
<td>(3 * 5) - 2</td>
<td>13</td>
<td>* has higher precedence than –</td>
</tr>
<tr>
<td>3 + 5 / 2</td>
<td>3 + (5 / 2)</td>
<td>5</td>
<td>/ has higher precedence than +</td>
</tr>
<tr>
<td>3 – 5 – 2</td>
<td>(3 - 5) - 2</td>
<td>-4</td>
<td>left-to-right associative</td>
</tr>
<tr>
<td>(3 - 5) - 2</td>
<td>itself</td>
<td>-4</td>
<td>better style</td>
</tr>
<tr>
<td>8 / 2 * (2 + 2)</td>
<td>(8 / 2) * (2 + 2)</td>
<td>16</td>
<td>left-to-right associative</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(multiplication and division have same precedence)</td>
</tr>
</tbody>
</table>
What value does the following expression evaluate to?

\[1 + 2 + "ABC" + 3 + 4\]

A. "12ABC34"
B. "3ABC7"
C. "3ABC34"
D. "12ABC7"
E. Compile-time error.
BUILT-IN DATA TYPES

- strings
- integers
- floating-point numbers
- booleans
- type conversion
# The double data type

**Typical usage:** scientific calculations involving real numbers.

<table>
<thead>
<tr>
<th>Values</th>
<th>IEEE floating-point numbers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Typical literals</td>
<td>18.25  -2.0  1.4142135623730951  6.022E23</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Operations</th>
<th>add</th>
<th>subtract</th>
<th>multiply</th>
<th>divide</th>
<th>remainder</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operators</td>
<td>+</td>
<td>-</td>
<td>*</td>
<td>/</td>
<td>%</td>
</tr>
</tbody>
</table>

- **expression** | **value** | **remark** |
- 1.5 + 0.25 | 1.75 | |
- 1.5 - 0.25 | 1.25 | |
- 1.5 * 2.0 | 3.0 | |
- 5.0 / 3.0 | 1.6666666666666667 | not exactly \( \frac{5}{3} \) |
- -1.0 / 0.0 | -Infinity | not an error |
- 0.0 / 0.0 | NaN | “not a number” |

- only 2\(^{64}\) different double values (not quite the same as real numbers)
- 6.022 \times 10^{23} (scientific notation)

- applying a double operator to two double operands always results in a double (can’t result in an error)
- only binary fractional values can be represented exactly, such as \( \frac{1}{4} + \frac{1}{16} + \frac{1}{128} = 0.3203125 \) (but not \( \frac{5}{3} \), \( \frac{1}{10} \), or \( \pi \))
### Excerpts from Java’s Math library

<table>
<thead>
<tr>
<th>Math library function</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>static double abs(double a)</td>
<td>absolute value of a</td>
</tr>
<tr>
<td>static double max(double a, double b)</td>
<td>maximum of a and b</td>
</tr>
<tr>
<td>static double min(double a, double b)</td>
<td>minimum of a and b</td>
</tr>
<tr>
<td>static double sin(double theta)</td>
<td>sine (sin θ)</td>
</tr>
<tr>
<td>static double cos(double theta)</td>
<td>cosine (cos θ)</td>
</tr>
<tr>
<td>static double tan(double theta)</td>
<td>tangent (tan θ)</td>
</tr>
<tr>
<td>static double exp(double a)</td>
<td>exponential (e^a)</td>
</tr>
<tr>
<td>static double log(double a)</td>
<td>natural logarithm (log e a)</td>
</tr>
<tr>
<td>static double pow(double a, double b)</td>
<td>power (a^b)</td>
</tr>
<tr>
<td>static long round(double a)</td>
<td>round to the nearest integer</td>
</tr>
<tr>
<td>static double random()</td>
<td>pseudorandom number in [0, 1]</td>
</tr>
<tr>
<td>static double sqrt(double a)</td>
<td>positive square root (√a)</td>
</tr>
</tbody>
</table>

- Also defined for int
- Inverse functions also available: asin(), acos(), and atan()
- Degrees in radians; to convert, use Math.toDegrees() and Math.toRadians()

You can discard your calculator now (please).
Quadratic equation

Goal. Print the solutions to the equation $ax^2 + bx + c = 0$, assuming $a \neq 0$.

```java
public class Quadratic {
    public static void main(String[] args) {

        // Parse coefficients from command-line.
        double a = Double.parseDouble(args[0]);
        double b = Double.parseDouble(args[1]);
        double c = Double.parseDouble(args[2]);

        // Calculate roots of $ax^2 + bx + c$.
        double discriminant = b*b - 4.0*a*c;
        double d = Math.sqrt(discriminant);
        double root1 = (-b + d) / (2.0*a);
        double root2 = (-b - d) / (2.0*a);

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

```
x^2 - 3x + 2
x^2 - x - 1
x^2 + x + 1
```
Floating-point catastrophe

Patriot missile.

- In February 1991, a Patriot missile failed to track and intercept an incoming Scud missile.
- Scud missile hit a U.S. Army barracks, killing 28 and wounding 260.
- Time measured in tenths of a second, but stored using binary floating-point.
- After 100 hours of continuous use, system’s internal clock had drifted by 1/3 second.
Built-In Data Types

- strings
- integers
- floating-point numbers
- booleans
- type conversion
The **boolean** data type

**Typical usage:** decision making in a program.

---

<table>
<thead>
<tr>
<th>values</th>
<th>true and false</th>
</tr>
</thead>
<tbody>
<tr>
<td>literals</td>
<td>true  false</td>
</tr>
<tr>
<td>operations</td>
<td>not  and  or</td>
</tr>
<tr>
<td>operators</td>
<td>!    &amp;&amp;</td>
</tr>
</tbody>
</table>

**Logical operators**

<table>
<thead>
<tr>
<th>expression</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>!false</td>
<td>true</td>
</tr>
<tr>
<td>!true</td>
<td>false</td>
</tr>
<tr>
<td>true and false</td>
<td></td>
</tr>
</tbody>
</table>

**Truth table for NOT**

<table>
<thead>
<tr>
<th>expression</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>false</td>
<td>true</td>
</tr>
<tr>
<td>false &amp;&amp; true</td>
<td>false</td>
</tr>
<tr>
<td>true &amp;&amp; false</td>
<td>false</td>
</tr>
<tr>
<td>true &amp;&amp; true</td>
<td>true</td>
</tr>
</tbody>
</table>

**Truth table for AND**

<table>
<thead>
<tr>
<th>expression</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>false</td>
<td></td>
</tr>
<tr>
<td>false</td>
<td></td>
</tr>
<tr>
<td>true</td>
<td></td>
</tr>
<tr>
<td>true</td>
<td></td>
</tr>
</tbody>
</table>

**Truth table for OR**

---

stay tuned for conditionals and loops
Boolean meme

!FALSE

IT'S FUNNY BECAUSE IT'S TRUE
Equality and comparison operators

Equality and comparison operators. To compare numeric values.

- Operands: two numeric expressions.
- Evaluates to: a value of type `boolean`.

```text
<table>
<thead>
<tr>
<th>operator</th>
<th>meaning</th>
<th>true</th>
<th>false</th>
</tr>
</thead>
<tbody>
<tr>
<td>==</td>
<td>equal</td>
<td>2 == 2</td>
<td>2 == 3</td>
</tr>
<tr>
<td>!=</td>
<td>not equal</td>
<td>3 != 2</td>
<td>2 != 2</td>
</tr>
<tr>
<td>&lt;</td>
<td>less than</td>
<td>2 &lt; 13</td>
<td>13 &lt; 2</td>
</tr>
<tr>
<td>&lt;=</td>
<td>less than or equal</td>
<td>2 &lt;= 2</td>
<td>3 &lt;= 2</td>
</tr>
<tr>
<td>&gt;</td>
<td>greater than</td>
<td>13 &gt; 2</td>
<td>2 &gt; 13</td>
</tr>
<tr>
<td>&gt;=</td>
<td>greater than or equal</td>
<td>2 &gt;= 2</td>
<td>2 &gt;= 3</td>
</tr>
</tbody>
</table>
```

equality and comparison operators in Java
### Equality and comparison operators: examples

<table>
<thead>
<tr>
<th>Condition</th>
<th>Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zero denominator?</td>
<td>denominator == 0</td>
</tr>
<tr>
<td>Non-negative discriminant?</td>
<td>( (b^2 - 4.0a^3c) \geq 0.0 )</td>
</tr>
<tr>
<td>Divisible by 60?</td>
<td>( (\text{minutes} \mod 60) == 0 )</td>
</tr>
<tr>
<td>RGB color is not black?</td>
<td>( (\text{red} &gt; 0)</td>
</tr>
<tr>
<td>Valid month?</td>
<td>( (\text{month} \geq 1) &amp;&amp; (\text{month} \leq 12) )</td>
</tr>
<tr>
<td>Invalid month?</td>
<td>(!((\text{month} \geq 1) &amp;&amp; (\text{month} \leq 12)) )</td>
</tr>
<tr>
<td>Floating-point roundoff error</td>
<td>( 0.1 \times 3.0 == 0.3 )</td>
</tr>
<tr>
<td>String equality</td>
<td>( \text{args}[0] == &quot;Hello&quot; )</td>
</tr>
</tbody>
</table>

- Parentheses for clarity: Arithmetic operators have higher precedence than equality/comparison operators.
- Compound boolean expressions.
- Don't do this! (Evaluates to false).
- Or this! (Always evaluates to false).
Example of computing with booleans: leap year test

Q. Is a given year a leap year?  
A. Yes if either (1) divisible by 400 or (2) divisible by 4 but not 100.

```java
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);
    }
}
```

if argument to System.out.println() is of type boolean, it prints either true or false

~/cos126/datatypes> java LeapYear 2024 true
~/cos126/datatypes> java LeapYear 2023 false
~/cos126/datatypes> java LeapYear 1900 false
~/cos126/datatypes> java LeapYear 2000 true
What does the following expression evaluate to?

A. Works: equivalent to \((\text{month} \geq 1) \&\& (\text{month} \leq 12)\).

B. Compile–time error: equivalent to \((1 \leq \text{month}) \leq 12\).
Data types: quiz 4

What does the following expression evaluate to?

\[
\text{month} \geq 1 \land \text{month} \leq 12
\]

A. Works: equivalent to \((\text{month} \geq 1) \land (\text{month} \leq 12)\)

B. Compile-time error: equivalent to \((\text{month} \geq (1 \land \text{month})) \leq 12\)

C. Compile-time error: equivalent to \(((\text{month} \geq 1) \land \text{month}) \leq 12\)
BUILT-IN DATA TYPES

- strings
- integers
- floating-point numbers
- booleans
- type conversion
Data types

Types limit the allowable operations on values and determine the meaning of those operations.

```java
public class StringMultiply {
    public static void main(String[] args) {
        String s = "123" * "456";
    }
}
```

Java compiler. The compiler checks for type mismatch errors in your code.
Data types

Types limit the allowable operations on values and determine the meaning of those operations.

<table>
<thead>
<tr>
<th>operator</th>
<th>int</th>
<th>double</th>
<th>boolean</th>
<th>String</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>addition</td>
<td>addition</td>
<td>no</td>
<td>concatenation</td>
</tr>
<tr>
<td>-</td>
<td>subtraction</td>
<td>subtraction</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>*</td>
<td>multiplication</td>
<td>multiplication</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>/</td>
<td>integer division</td>
<td>division</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>&amp; &amp;</td>
<td>no</td>
<td>no</td>
<td>logical AND</td>
<td>no</td>
</tr>
<tr>
<td></td>
<td></td>
<td>no</td>
<td>logical OR</td>
<td>no</td>
</tr>
<tr>
<td>!</td>
<td>no</td>
<td>no</td>
<td>logical NOT</td>
<td>no</td>
</tr>
<tr>
<td>&lt;</td>
<td>less than</td>
<td>less than</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>:</td>
<td>:</td>
<td>:</td>
<td>:</td>
<td>:</td>
</tr>
</tbody>
</table>

can’t subtract, multiply, or divide two String or boolean values  
(compile-time errors)

Static typing. Every Java variable and expression has a type that is known at compile time.

- Benefit: compiler catches entire class of programming errors automatically.
- Drawback: extra boilerplate code.
Ariane 5 rocket.

• European Space Agency spent a decade and $7 billion in research and development.
• Rocket self-destructed 39 seconds after first launch.
• Source of bug: unsafe type conversion of 64-bit floating-point number to 16-bit integer.

https://www.youtube.com/watch?v=PK_yguLapgA
Type conversions with built-in types

Type conversion is an essential aspect of programming.

Automatic type conversions.

- String conversion: from any type to String (via string concatenation).
- Numeric promotion: from int to double (when a double is expected).

System methods.

- `Integer.parseInt()` from String to int.
- `Double.parseDouble()` from String to double.

Explicit casts from one type to another.

- Cast from double to int. 
- Cast from int to double.
Q. What is type and value of each expression on the left?

<table>
<thead>
<tr>
<th>expression</th>
<th>type</th>
<th>value</th>
<th>remark</th>
</tr>
</thead>
<tbody>
<tr>
<td>((7 / 2) \ast 2.0)</td>
<td>double</td>
<td>6.0</td>
<td>integer division; then promotion to double</td>
</tr>
<tr>
<td>((7 / 2.0) \ast 2)</td>
<td>double</td>
<td>7.0</td>
<td>promotion to double; then floating-point division</td>
</tr>
<tr>
<td>&quot;12&quot; + 6</td>
<td>String</td>
<td>&quot;126&quot;</td>
<td>conversion to String</td>
</tr>
<tr>
<td>0 == false</td>
<td>compile-time error</td>
<td></td>
<td>can't compare int to boolean</td>
</tr>
</tbody>
</table>
Simulate the rolling of a fair die

Goal. Given an integer $n > 0$, generate a uniformly random integer between 1 and $n$. Each possible integer is equally likely.
Generate pseudo-random integers

**Problem.** Given an integer \( n > 0 \), generate a uniformly random integer between 0 and \( n - 1 \).

**Useful system method.** \( \text{Math.random()} \) returns a pseudorandom \textit{double} value in \([0, 1)\).

**Idea.** Scale to desired range, round down to nearest integer.

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

---

String to int (system method)

double to int (cast)

int to double (automatic)
Which expression generates a pseudorandom even integer between 0 and 2n−1?

A. 2 * (int) n * Math.random()
B. 2 * (int) (n * Math.random())
C. 2 * n * (int) Math.random()
D. (int) (2 * n * Math.random())
Overview

This lecture. Write programs with declaration, assignment, and print statements.
Next week. Write programs with conditionals and loops.
<table>
<thead>
<tr>
<th>media</th>
<th>source</th>
<th>license</th>
</tr>
</thead>
<tbody>
<tr>
<td>PEMDAS</td>
<td>Mometrix</td>
<td></td>
</tr>
<tr>
<td>PEMDAS meme</td>
<td>New York Times</td>
<td></td>
</tr>
<tr>
<td>Scientific Calculator</td>
<td>Wikimedia</td>
<td>CC BY-SA 3.0</td>
</tr>
<tr>
<td>Solving Quadratic Equations</td>
<td>Adobe Stock</td>
<td>education license</td>
</tr>
<tr>
<td>Patriot Missile Launcher</td>
<td>Raytheon</td>
<td></td>
</tr>
<tr>
<td>Incorrectly Calculated Range Gate</td>
<td>GAO</td>
<td>public domain</td>
</tr>
<tr>
<td>Scud Missile Hits a U.S. Barracks</td>
<td>New York Times</td>
<td></td>
</tr>
<tr>
<td>!FALSE</td>
<td>redbubble.com</td>
<td></td>
</tr>
<tr>
<td>Ariane 5 Rocket Launch</td>
<td>European Space Agency</td>
<td></td>
</tr>
<tr>
<td>Two Red Dice</td>
<td>Wikimedia</td>
<td>public domain</td>
</tr>
<tr>
<td>Ten-Sided Die</td>
<td>Wikimedia</td>
<td>CC BY-SA 3.0</td>
</tr>
<tr>
<td>Hundred-Sided Die</td>
<td>gameoutonline.com</td>
<td></td>
</tr>
</tbody>
</table>