public class Expressions
{
    public static void main(String[] args)
    {
        System.out.println(1+2*3/2+7);
    }
}
Topic 08: Expressions

• It's all backwards
• Types
• Expressions
• Operator Precedence 2.2.1
• Operator Associativity Appendix B
• Operators in Detail Appendix B
• Mixed Types
• Examples
It's All Backwards

• You (probably) think of programs from the outside in:
  
  class
    contains a method
      contains a statement
        contains an expression
          contains a smaller expression
It's All Backwards

- Java thinks of programs from the inside out:
  
  smaller expression
  inside an expression
  inside a statement
  inside a method
  inside a class
Expressions, Inside Out

\[ a = (b \times c) + (d \times e) + (f \times g); \]
Expressions, Inside Out

\[ a = (b \times c) + (d \times e) + (f \times g); \]

**Java thinks:**

What does \( b \times c \) produce?

(Everything else is ignored!)
Expressions, Inside Out

\[ a = (b \times c) + (d \times e) + (f \times g); \]

**Java thinks:**

What does \( d \times e \) produce?

(Everything else is ignored!)
Expressions, Inside Out

\[ a = (b \times c) + (d \times e) + (f \times g) ; \]

Java thinks:

What does \(( \ldots ) + ( \ldots )\) produce?
Expressions, Inside Out

\[ a = (b \times c) + (d \times e) + (f \times g); \]

**Order of Operations**

In what order are the various operators evaluated?
Topic 08: Expressions

- It's all backwards
- **Types**
- Expressions
  - 2.2.1
- Operator Precedence
  - Appendix B
- Operator Associativity
  - Appendix B
- Operators in Detail
- Mixed Types
- Examples
Types

• Compiler doesn't know values
  – Those will be calculated at runtime

• Compiler knows **types**

• Every expression has a type
  – Same as variables (mostly)

• Compiler asks: can I do this op with these types?
Types

double a;
int b = ...;
int c = ...;
int d = ...;
double e = ...;
byte f = ...;
char g = ...;

\[ a = (b \times c) + (d \times e) + (f \times g) \]
double a;
int b = ...;
int c = ...;
int d = ...;
double e = ...;
byte f = ...;
char g = ...;

a = (int*c) + (d*e) + (f*g);
Types

double a;
int b = ...;
int c = ...;
int d = ...;
double e = ...;
byte f = ...;
char g = ...;

\[
a = (\text{int} \times \text{int}) + (d \times e) + (f \times g);\]
Types

double a;
int b = ...;
int c = ...;
int d = ...;
double e = ...;
byte f = ...;
char g = ...;

\[ a = \text{int} + (d \times e) + (f \times g); \]
Types

double a;
int b = ...;
int c = ...;
int d = ...;
double e = ...;
byte f = ...;
char g = ...;

\[
a = \text{int } + (\text{int } * e) + (f * g);
\]
Types

double a;
int b = ...;
int c = ...;
int d = ...;
double e = ...;
byte f = ...;
char g = ...;

\[
a = \textbf{int} + (\textbf{int} \times \textbf{double}) + (f \times g);
\]
Types

double a;
int b = ...;
int c = ...;
int d = ...;
double e = ...;
byte f = ...;
char g = ...;

\[ a = \textbf{int} + \textbf{double} + (f \times g) \]
Types

double a;
int b = ...;
int c = ...;
int d = ...;
double e = ...;
byte f = ...;
char g = ...;

\[
a = \quad \text{int} \ + \ \text{double} \ + \ (\text{byte} \ * g) \ ;
\]
double a;
int b = ...;
int c = ...;
int d = ...;
double e = ...;
byte f = ...;
char g = ...;

\[
a = \text{int} + \text{double} + (\text{byte} \times \text{char})
\]
Types

double a;
int b = ...;
int c = ...;
int d = ...;
double e = ...;
byte f = ...;
char g = ...;

\[
a = \text{int} + \text{double} + \text{int};
\]
Types

double a;
int b = ...;
int c = ...;
int d = ...;
double e = ...;
byte f = ...;
char g = ...;

a = \textcolor{blue}{\textbf{double}} + \textcolor{blue}{\textbf{int}} ;
double a;
int b = ...;
int c = ...;
int d = ...;
double e = ...;
byte f = ...;
char g = ...;

a = double;
Types

double a;
int b = ...;
int c = ...;
int d = ...;
double e = ...;
byte f = ...;
char g = ...;

double
;
Order of Evaluation

\[ a = (b \times c) + (d \times e) + (f \times g); \]

I'll use this `shorthand` sometimes in this slide deck to show what operations happen first.
Expressions Inside Statements

byte b = ... ;
System.out.println("s=" + (short)b);
Expressions Inside Statements

```java
byte b = ... ;
System.out.println("s=" + (short)byte);
```
Expressions Inside Statements

byte b = ... ;
System.out.println("s=" + short );
Expressions Inside Statements

byte b = ... ;
System.out.println("s=" + String + short );
Expressions Inside Statements

```java
code
byte b = ... ;
System.out.println(String);
```
Expressions Inside Statements

byte b = ...;
void ;
Expressions Inside Statements

byte b = ... ;
System.out.println(String);

How does Java evaluate this part of the statement?

Expressions!
Expressions Inside Statements

byte b = ... ;
???? out.println(String);
Expressions Inside Statements

byte b = ... ;
???? println(String);
Expressions Inside Statements

```java
byte b = ... ;
???? (String);
```
Expressions Inside Statements

```java
byte b = ... ;
void ;
```
Temporary Variables

**Original Code:**
byte b;
System.out.println("s=" + (short)b);

**Same Code, Using Temp Variables:**
byte b;
short temp1 = (short)b;
String temp2 = “s=” + temp1;
System.out.println(temp2);
Topic 08: Expressions

- It's all backwards
- Types
- **Expressions**
  - Operator Precedence
  - Operator Associativity
  - Operators in Detail
  - Mixed Types
  - Examples
  
  2.2.1
  
  Appendix B
  
  Appendix B
What is an Expression?

• An expression is:
  – One simple element
  ...or...
  – Several simple elements
  – Combined with one or more operators

• All expressions resolve to a value at runtime
• All expressions have a type, known at compile time
What is an Expression?

Assume that $a, b, c$ are all int's

Three simple expressions, all of type int
What is an Expression?

Assume that \(a, b, c\) are all \textit{int}'s

\[ \text{a} * \text{b} + \text{c} \]

Two operators.

How do we know which to execute first?
What Order?

In what order should each expression evaluated?

\[ a + b \times c \]
\[ a \times b \div c \]
\[ a - b + c \]
\[ (a-b) \times (c+d) \]
\[ a == b + c/d \]
\[ a = b + c/d \]
What Order?

In what order should each expression evaluated?

\[
\begin{align*}
& a + b \times c \\
& a \times b / c \\
& a - b + c \\
& (a-b) \times (c+d) \\
& a == b + c/d \\
& a = b + c/d \\
\end{align*}
\]

But how does Java know this?
Topic 08: Expressions

- It's all backwards
- Types
- Expressions
- **Operator Precedence**
- **Operator Associativity**
- Operators in Detail
- Mixed Types
- Examples

2.2.1 Appendix B

Appendix B
Operator Precedence

• Some operators are “higher precedence” than others
  – High precedence operators are resolved first
  – Use parentheses to force a different order

• Examples:
  – Multiplication higher than addition
  – Addition higher than comparison
Operator Precedence
(partial list)

* / %
+
== !=
&&
||
=

Higher Precedence

Lower Precedence

Full list at:
www.cs.arizona.edu/classes/cs127a/spring16/keywords.pdf
What Order?

In what order should each expression be evaluated?

\[
\begin{align*}
  a & \quad + \quad b \quad * \quad c \\
  a & \quad * \quad b \quad / \quad c \\
  a & \quad - \quad b \quad + \quad c \\
  (a-b) & \quad * \quad (c+d) \\
  a & \quad == \quad b \quad + \quad c/d \\
  a & \quad = \quad b \quad + \quad c/d
\end{align*}
\]

This expression is ordered by parentheses. Operator precedence is ignored when parentheses are present.
What Order?

In what order should each expression be evaluated?

\[ a + b \times c \]
\[ a \times b / c \]
\[ a - b + c \]
\[ (a-b) \times (c+d) \]
\[ a == b + c/d \]
\[ a = b + c/d \]

This expression is ordered by operator precedence. Multiplication happens before addition.
What Order?

In what order should each expression be evaluated?

\[
\begin{align*}
\text{a} & \text{ + } \text{b} \times \text{c} \\
\text{a} & \times \text{b} \div \text{c} \\
\text{a} & \text{ - } \text{b} \text{ + } \text{c} \\
(\text{a-b}) & \times (\text{c+d}) \\
\text{a} & \text{ == } \text{b} \text{ + } \text{c/d} \\
\text{a} & \text{ = } \text{b} \text{ + } \text{c/d}
\end{align*}
\]

These expressions are ordered by operator precedence. Multiplication happens before addition; addition happens before either comparison or assignment.
What Order?

In what order should each expression be evaluated?

\[ \text{a + b * c} \]

\[ \text{a * b / c} \]

\[ \text{a - b + c} \]

\[ \text{(a-b) * (c+d)} \]

\[ \text{a == b + c/d} \]

\[ \text{a = b + c/d} \]

But how to handle these?
Topic 08: Expressions

- It's all backwards
- Types
- Expressions
- Operator Precedence
- **Operator Associativity**
- Operators in Detail
- Mixed Types
- Examples

Appendix B
Operator Associativity

- Determines order when operators have the same precedence
- Right associative: do the right thing first
- Left associative: do the left thing first
## Operator Precedence

(partial list)

<table>
<thead>
<tr>
<th>Operator(s)</th>
<th>Associativity</th>
<th>Prec.</th>
</tr>
</thead>
<tbody>
<tr>
<td>* / %</td>
<td>Left Assoc.</td>
<td>Higher Prec.</td>
</tr>
<tr>
<td>+ -</td>
<td>Left Assoc.</td>
<td></td>
</tr>
<tr>
<td>== !=</td>
<td>Left Assoc.</td>
<td></td>
</tr>
<tr>
<td>&amp;&amp;</td>
<td>Left Assoc.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>=</td>
<td>Right Assoc.</td>
<td>Lower Prec.</td>
</tr>
</tbody>
</table>

Full list at:

[www.cs.arizona.edu/classes/cs127a/spring16/keywrdos.pdf](http://www.cs.arizona.edu/classes/cs127a/spring16/keywrdos.pdf)
What Order?

In what order should each expression be evaluated?

\[
\begin{align*}
a + b \times c \\
\hline
a \times b / c \\
\hline
a - b + c \\
\hline
(a-b) \times (c+d) \\
\hline
a == b + c/d \\
\hline
a = b + c/d
\end{align*}
\]

These expressions are ordered by operator associativity. Both multiplication/division and addition/subtraction are left-associative.

Note that mul/div/mod are grouped together, as are add/sub.
Why is = Right-Associative?

\[ x = y = 0; \]

Programmer thinks: “I want to set both \( x \) and \( y \) to zero.”
Why is $=$ Right-Associative?

$x = y = 0;$

$x = (y = 0);$
Why is \( = \) Right-Associative?

\[
x = y = 0;
\]

\[
x = (y = 0);
\]

\[
x = 0;
\]

Evaluating the
\[
y = 0
\]
expression
Why is $= =$ Right-Associative?

$$x = y = 0;$$
$$x = (y = 0);$$
$$x = 0;$$

$y = 0;$$
$$x = 0;$$

The same thing, as two statements.
Why is $\texttt{=}$ Right-Associative?

```c
int x = ... ;
int y = ... ;
int z = ... ;

x = y = z;
```
Why is $\mathbf{=}$ Right-Associative?

```plaintext
int x = ... ;
int y = ... ;
int z = ... ;

x = y = int ;
```
Why is \( = \) Right-Associative?

\[
\text{int } x = \ldots ;
\]
\[
\text{int } y = \ldots ;
\]
\[
\text{int } z = \ldots ;
\]
\[
\text{int } x = \boxed{\text{int}} ;
\]
Why is = Right-Associative?

```java
int x = ... ;
int y = ... ;
int z = ... ;
```
Topic 08: Expressions

- It's all backwards
- Types
- Expressions
- Operator Precedence
- Operator Associativity
- Operators in Detail
- Mixed Types
- Examples

2.2.1 Appendix B

Appendix B
Math Operators

\[
\begin{align*}
\text{int} + \text{int} & \rightarrow \text{int} \\
\text{long} + \text{long} & \rightarrow \text{long} \\
\text{float} + \text{float} & \rightarrow \text{float} \\
\text{double} + \text{double} & \rightarrow \text{double}
\end{align*}
\]

Same Rules: \(- \quad * \quad / \quad \%\)
Math Operators

Same Rules:  -  *  /  %

This is why integer division must always round down!
How does Java know whether to perform concatenation or addition?

The types!
Casts

Not all casts are allowed – some are illegal.
Some methods return `void`. 
new

Arrays:
new `<type>` [ `int` ] → `<type>` [ ]

Objects:
new `<type>` ( `<args>` ) → `<type>`
Comparison

<table>
<thead>
<tr>
<th>Type</th>
<th>Comparison</th>
<th>Type</th>
<th>Comparison</th>
</tr>
</thead>
<tbody>
<tr>
<td>int</td>
<td>==</td>
<td>int</td>
<td>==</td>
</tr>
<tr>
<td>long</td>
<td>==</td>
<td>long</td>
<td>==</td>
</tr>
<tr>
<td>float</td>
<td>==</td>
<td>float</td>
<td>==</td>
</tr>
<tr>
<td>double</td>
<td>==</td>
<td>double</td>
<td>==</td>
</tr>
<tr>
<td>Object</td>
<td>==</td>
<td>Object</td>
<td>==</td>
</tr>
</tbody>
</table>

Same Rules:  

!= < <= > >=

(for `Object`, only `==` and `!=` are legal)
Comparison

- boolean & boolean → boolean
- boolean | boolean → boolean
- ! boolean → boolean
Assignment

\[ \text{<type>} \quad = \quad \text{<any>} \quad \rightarrow \quad \text{<type>} \]
Topic 08: Expressions

- It's all backwards
- Types
- Expressions
  - Operator precedence
  - Operator associativity
- The type of expressions
- Mixed types
- Examples
Mixed Types

- Many operators take two inputs
- What happens when they have different types?
- Smaller type is implicitly cast to the larger
Remember this?

• The following casts are “implicit” - meaning that they happen automatically, because they are safe

<table>
<thead>
<tr>
<th>Type 1</th>
<th>Type 2</th>
<th>Type 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>byte</td>
<td>short</td>
<td>int</td>
</tr>
<tr>
<td>char</td>
<td>int</td>
<td>long</td>
</tr>
<tr>
<td>float</td>
<td>double</td>
<td></td>
</tr>
</tbody>
</table>

[any integer] → [any floating point]
Type Promotion

\[
\text{int} + \text{long}
\]

- byte → short → int → long
- char → int → long
- float → double
- [any integer] → [any floating point]
Type Promotion

```
long + long
```

- `byte → short → int → long`
- `char → int → long`
- `float → double`
- `[any integer] → [any floating point]`
Type Promotion

\[
\begin{align*}
\text{double} + \text{<any>} & \rightarrow \text{double} \\
\text{float} + \text{<any>} & \rightarrow \text{float} \\
\text{long} + \text{<any>} & \rightarrow \text{long} \\
\text{int} + \text{<any>} & \rightarrow \text{int}
\end{align*}
\]

(and the same for swapped inputs)

Same Rules: $- * / \%$
Topic 08: Expressions

- It's all backwards
- Types
- Expressions
- Operator precedence
- Operator associativity
- The type of expressions
- Mixed types
- Examples
Some Example Expressions

num \% factor1 == 0 && num \% factor2 == 0

a == 0 || b == 0 || c == 0 || d == 0

foo == bar || baz == 1 && foo == 0
Some Example Expressions

\[
\text{num} \mod \text{factor1} == 0 && \text{num} \mod \text{factor2} == 0
\]

\[
a == 0 || b == 0 || c == 0 || d == 0
\]

\[
\text{foo} == \text{bar} || \text{baz} == 1 && \text{foo} == 0
\]
Some Example Expressions

\[ \text{num \ mod \ factor1} == 0 \ \&\& \ \text{num \ mod \ factor2} == 0 \]

\[ \text{a == 0} \ \text{||} \ \text{b == 0} \ \text{||} \ \text{c == 0} \ \text{||} \ \text{d == 0} \]

\[ \text{foo == bar} \ \text{||} \ \text{baz == 1} \ \&\& \ \text{foo == 0} \]
Some Example Expressions

\[
\text{num} \mod \text{factor1} == 0 && \text{num} \mod \text{factor2} == 0
\]

\[
a == 0 || b == 0 || c == 0 || d == 0
\]

\[
\text{foo} == \text{bar} || \text{baz} == 1 && \text{foo} == 0
\]
Types of our Example Expressions

\[
\text{num} \mod \text{factor1} == 0 \land \text{num} \mod \text{factor2} == 0
\]

\[
a == 0 \lor b == 0 \lor c == 0 \lor d == 0
\]

\[
\text{foo} == \text{bar} \lor \text{baz} == 1 \land \text{foo} == 0
\]
Types of our Example Expressions

\begin{align*}
\text{int} & \quad == \quad 0 \quad \&\& \quad \text{int} \quad == \quad 0 \\

a & \quad == \quad 0 \quad || \quad b \quad == \quad 0 \quad || \quad c \quad == \quad 0 \quad || \quad d \quad == \quad 0 \\

\text{foo} & \quad == \quad \text{bar} \quad || \quad \text{baz} \quad == \quad 1 \quad \&\& \quad \text{foo} \quad == \quad 0
\end{align*}

Assume all variables are int

www.cs.arizona.edu/classes/cs127a/spring16/keywords.pdf
Types of our Example Expressions

\[ \text{boolean} \quad \&\& \quad \text{boolean} \]

\[ a == 0 || b == 0 || c == 0 || d == 0 \]

\[ \text{foo} == \text{bar} || \text{baz} == 1 \&\& \text{foo} == 0 \]

Assume all variables are \text{int}

www.cs.arizona.edu/classes/cs127a/spring16/keywords.pdf
Types of our Example Expressions

- \( \text{num} \% \text{factor1} == 0 \) && \( \text{num} \% \text{factor2} == 0 \)
- \( a == 0 \) || \( b == 0 \) || \( c == 0 \) || \( d == 0 \)
- \( \text{foo} == \text{bar} \) || \( \text{baz} == 1 \) && \( \text{foo} == 0 \)

Assume all variables are \( \text{int} \)

www.cs.arizona.edu/classes/cs127a/spring16/keywords.pdf
Types of our Example Expressions

num % factor1 == 0 && num % factor2 == 0

foo == bar || baz == 1 && foo == 0

Assume all variables are int
Types of our Example Expressions

num % factor1 == 0 && num % factor2 == 0

boolean || boolean || boolean

foo == bar || baz == 1 && foo == 0

Assume all variables are int

www.cs.arizona.edu/classes/cs127a/spring16/keywords.pdf
Types of our Example Expressions

num % factor1 == 0 && num % factor2 == 0

foo == bar || baz == 1 && foo == 0

Assume all variables are int

www.cs.arizona.edu/classes/cs127a/spring16/keywords.pdf
Types of our Example Expressions

\( \text{num} \% \text{factor1} == 0 \) \&\& \( \text{num} \% \text{factor2} == 0 \)

\( \text{foo} == \text{bar} \) \text{||} \( \text{baz} == 1 \) \&\& \( \text{foo} == 0 \)

\textbf{boolean}

\( \text{foo} == \text{bar} \) \text{||} \( \text{baz} == 1 \) \&\& \( \text{foo} == 0 \)

\textbf{Assume all \text{variables} \text{are int}}

\textbf{www.cs.arizona.edu/classes/cs127a/spring16/keywords.pdf}
Types of our Example Expressions

num % factor1 == 0 && num % factor2 == 0

a == 0 || b == 0 || c == 0 || d == 0

Assume all variables are int

www.cs.arizona.edu/classes/cs127a/spring16/keywords.pdf
Types of our Example Expressions

num % factor1 == 0 && num % factor2 == 0

a == 0 || b == 0 || c == 0 || d == 0

Assume all variables are int

www.cs.arizona.edu/classes/cs127a/spring16/keywords.pdf
Types of our Example Expressions

- num % factor1 == 0 && num % factor2 == 0
- a == 0 || b == 0 || c == 0 || d == 0

Assume all variables are int
In Practice

- You do **not** have to memorize the entire precedence table!
- Mostly, it just works the way you would expect
  - mul/div/mod higher than add/sub
  - comparisons lower than math
  - Assignment lower than all
- Use parens when you're unsure
- Use parens if your *reader* will be unsure
Crazy results

• Occasionally, the results are really surprising!
# The + Operator

<table>
<thead>
<tr>
<th></th>
<th>+</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>String</td>
<td>String</td>
<td>→</td>
</tr>
<tr>
<td>String</td>
<td>int</td>
<td>→</td>
</tr>
<tr>
<td>String</td>
<td>char</td>
<td>→</td>
</tr>
<tr>
<td>int</td>
<td>int</td>
<td>→</td>
</tr>
<tr>
<td>int</td>
<td>char</td>
<td>→</td>
</tr>
<tr>
<td>char</td>
<td>char</td>
<td>→</td>
</tr>
</tbody>
</table>
The + Operator

- `String + String` → `String`
- `String + int` → `String`
- `String + char` → `String`
- `int + int` → `int`
- `int + char` → `String`
- `char + char` → `char`
The + Operator

- `String + String → String`
- `String + int → String`
- `String + char → String`
- `int + int → int`
- `int + char → char`
- `char + char → char`
The + Operator

\[
\begin{array}{ccc}
\text{String} & + & \text{String} & \rightarrow & \text{String} \\
\text{String} & + & \text{int} & \rightarrow & \text{String} \\
\text{String} & + & \text{char} & \rightarrow & \text{String} \\
\text{int} & + & \text{int} & \rightarrow & \text{int} \\
\text{int} & + & \text{char} & \rightarrow & \text{int} \\
\text{char} & + & \text{char} & \rightarrow & \text{int} \\
\end{array}
\]
char c1 = 'H';
char c2 = 'e';
char c3 = 'l';
char c4 = 'l';
char c5 = 'o';

System.out.print (c1);
System.out.print (c2);
System.out.print (c3);
System.out.print (c4);
System.out.println(c5);

System.out.println(""+c1+c2+c3+c4+c5);
System.out.println(c1+c2+c3+c4+c5);
char c1 = 'H';
char c2 = 'e';
char c3 = 'l';
char c4 = 'l';
char c5 = 'o';

System.out.print (c1);
System.out.print (c2);
System.out.print (c3);
System.out.print (c4);
System.out.println(c5);

System.out.println(""+c1+c2+c3+c4+c5);
System.out.println(72+101+108+108+111);
Characters

char c1 = 'H';
char c2 = 'e';
char c3 = 'l';
char c4 = 'l';
char c5 = 'o';

System.out.print  (c1);
System.out.print  (c2);
System.out.print  (c3);
System.out.print  (c4);
System.out.println(c5);

System.out.println(""+c1+c2+c3+c4+c5);
System.out.println(500);

Output:
Hello
Hello
500
Topic 08: Expressions

- It's all backwards
- Types
- Expressions
- Operator precedence
- Operator associativity
- The type of expressions
- Mixed types
- Examples

Summary