All the Operators
Precedence

- An operator with higher precedence is done earlier (precedes) one with lower precedence
  - A higher precedence is indicated with a lower number; zero is the highest precedence
- Most of the time, operators with equal precedence are done left to right
  - Examples:
    - $3 + 4 \times 5$ gives 23
    - $10 - 5 - 2$ gives 3
- Exceptions: unary operators, casts, assignment operators, the ternary operator: all done right to left
Postfix operators have the highest precedence

(parameters) Parameter lists

[ ] Brackets indicate indexing into an array

- Accesses methods and variables

expr++, expr-- Postincrement, postdecrement
Unary prefix operators

Unary prefix operators have the next highest precedence:

++ expr  Preincrement

-- expr  Predecrement

+ -  Unary plus and unary minus

!  Logical negation (not)

~  Bitwise complement (invert every bit)
Object creation and casting

new Create a new instance of a class

(type) Cast (convert) to the given type

- Slides are in order of decreasing precedence
  - Higher precedence means “more tightly bound”
  - The lowest precedence operator is the “main” operator in an expression
  - Frequently the lowest precedence operator is assignment, for example x = y + z;
Multiplicative operators

*  Multiply

/  Divide

%  Modulus

- These all have the same precedence
Additive operators

+  Add

-  Subtract
Shift operators

<<  Left shift, end off
- For small integers, this is equivalent to multiplying by a power of two
- Example: 100 << 3 gives 800

>>  Right shift with sign extension
- For small integers, this is equivalent to an integer divide by a power of two
- Example: 100 >> 2 gives 25

>>>  Right shift with zero fill
- Does not make sense for numbers
Relational operators

<  Less than

<=  Less than or equal to

>  Greater than

>=  Greater than or equal to

instanceof  Determines whether its left operand is an object whose type (class or interface) is the right operand

Example: if (myPet instanceof Dog) {...}

- These all have the same precedence, and it is higher than equality/inequality tests
A beginner’s error

- if (0 <= i < a.length) { ... }
- Operations are done left to right
- 0 <= i will be either true or false
- Neither true < a.length nor false < a.length is legal
- The correct expression should be
- if (0 <= i && i < a.length) { ... }


Equality and inequality

==  Test if equal
  - For primitive types, tests if the values are equal
  - For objects, tests if both sides refer to the same object

!=  Test if not equal
  - For primitive types, tests if the values are unequal
  - For objects, tests if the sides refer to different objects

Reminder: these tests should not be used on floating-point numbers (float or double)
AND

& AND

- For integral types, ANDs each corresponding pair of bits
  - 0 & 0 == 0
  - 0 & 1 == 0
  - 1 & 0 == 0
  - 1 & 1 == 1

- For booleans, performs the logical AND operation

- Boolean & is like &&, but both operands are evaluated, even if it is possible to decide the result from the left operand alone
**Exclusive OR**

\[ ^\wedge \]

**XOR**

- For integral types, XORs each corresponding pair of bits
  - \(0 \wedge 0 = 0\)
  - \(0 \wedge 1 = 1\)
  - \(1 \wedge 0 = 1\)
  - \(1 \wedge 1 = 0\)

- For booleans, performs the logical XOR operation
  - \(a \wedge b\) is true if either \(a\) is true or \(b\) is true, but not both

- There is no \(^\wedge\wedge\) operation
OR

- For integral types, ORs each corresponding pair of bits
  - 0 | 0 == 0
  - 0 | 1 == 1
  - 1 | 0 == 1
  - 1 | 1 == 1

- For booleans, performs the logical OR operation

  Boolean | is like ||, but both operands are evaluated, even if it is possible to decide the result from the left operand alone
The ternary operator

- `boolean-expr ? expression-1 : expression-2`

- This is like `if-then-else` for values rather than for statements

- If the `boolean-expr` evaluates to `true`, the result is `expression-1`, else it is `expression-2`

- Example: `max = a > b ? a : b ;` sets the variable `max` to the larger of `a` and `b`

- `expression-1` and `expression-2` need not be the same type, but either result must be useable

- *The ternary operator is right associative!*
  - To avoid confusion, use parentheses if your expression has more than one ternary operator
The assignment operators I

- The assignment operators have the lowest precedence
  - Assignment is an operation
  - Assignment is right associative
    
    \[ a = b = c = 7.5 \times w; \]
    
    assigns \(7.5\times w\) to \(c\), then assigns \(c\) to \(b\), then assigns \(b\) to \(a\) – if all these assignments are legal

- Example:
  
  ```java
  if ((line = reader.newLine()) == null) { ... }
  ```
The assignment operators II

- There are a lot of assignment operations besides $=$:
- $\text{variable} \ += \ \text{expression}$ means the same as
  $\text{variable} = \text{variable} + \ \text{expression}$
- $\text{variable} \ -= \ \text{expression}$ means the same as
  $\text{variable} = \text{variable} - \ \text{expression}$
- $\text{variable} \ *= \ \text{expression}$ means the same as
  $\text{variable} = \text{variable} * \ \text{expression}$
- $\text{variable} \ /= \ \text{expression}$ means the same as
  $\text{variable} = \text{variable} / \ \text{expression}$
The assignment operators III

- `variable %= expression` means the same as `variable = variable % expression`
- `variable <<= expression` means the same as `variable = variable << expression`
- `variable >>= expression` means the same as `variable = variable >> expression`
- `variable >>= expression` means the same as `variable = variable >>> expression`
The assignment operators IV

- `variable &= expression` means the same as
  `variable = variable & expression`

- `variable |= expression` means the same as
  `variable = variable | expression`

- `variable ^= expression` means the same as
  `variable = variable ^ expression`
What you need to know

- You should understand what each operator does
- Parameter lists, array indexing, casting, postfix `++` and `--`, and the dot operator are done first
  - In particular, a cast refers to the *one* following entity, so to cast the result of an expression you need extra parentheses
  - Example 1: `variable = (type)(expression);`
  - Example 2: `variable = ((type)variable).method();`
- In arithmetic, the unary operators `+` and `-` are done first, then multiplication and division, then addition and subtraction
- All assignment operators are done last
- For anything else, it’s a good idea to use parentheses anyway (even if you remember the order of precedence, other people won’t)
The End