

CMSC 131: Chapter 4: Supplement

More on Variables, Operators, and Types

Primitive Data Types

Java's basic data types:

Integer Types:

byte	1 byte	Range: -128 to +127
short	2 bytes	Range: roughly -32 thousand to +32 thousand
int	4 bytes	Range: roughly -2 billion to +2 billion
long	8 bytes	Range: Huge!

Floating-Point Types:

float	4 bytes	Roughly 7 digits of precision
double	8 bytes	Roughly 15 digits of precision

Other types:

boolean	1 byte	{true, false}
char	2 bytes	A single (Unicode) character

Constants (Literals)

Specifying constants: (also called literals)

Integer Types:

byte	
short	optional sign and digits (0-9)
int	
long	Same as above, but followed by 'L' or 'l'

Floating-Point Types:

double	Two allowable forms: Decimal notation: Scientific notation: (use E or e for base 10 exponent)
---------------	---

float	Same as double, but followed by 'f' or 'F'
--------------	--

Character and String Constants

char constants: Single character enclosed in single quotes ('...') including:

- letters and digits:
- punctuation symbols:
- escape sequences:

String constants: Zero or more characters enclosed in double quotes ("...")

Escape sequences:

\"	double quote	\n	new-line character (start a new line)
\'	single quote	\t	tab character
\\	backslash		

Variable Names

Valid Variable Names:

- **Starts with:** a letter (a-z or A-Z), dollar sign (\$), or underscore (_).
- **Followed by:** zero or more letters, dollar signs, underscores, or digits (0-9).
- Uppercase and lowercase are different.
- Cannot be any of the **reserved names**. Examples:

class, float, int, if, then, else, do, public, private, void, ...

Variable Names

Examples of valid and invalid identifier names:

Valid:

\$_
R2D2
INT
_dogma_95_
riteOnThru
SchultzieVonWienerschnitzelIII

Invalid:

30DayAbs
2
pork&beans
private
C-3PO

Good Variable Names

Choosing Good Names:

- Do not use ` \$ '.
- Avoid names that are identical other than differences in case.
- Use meaningful names, but avoid excessive length.

Variable Name Conventions

Naming Conventions:

Variables and methods: Start with lowercase, and use uppercase for each new word:

Class names: Start with uppercase and uppercase for each new word:

Named constants (variables whose value never changes): All uppercase with underscores between words:

More About Operators

We will discuss the following additional elements:

- Short-circuiting with logical operators
- Increment and decrement operators
- Assignment operators
- Operator precedence

Short-Circuiting

Short-circuiting in Logical Operators: The logical operator `&&` does not evaluate the right operand if the left operand is **false**.

```
// ... suppose that: int x = 15
if ( ( x < 10 ) && ( z > 5*y ) ) ...      // the test z > 5*y is not made
```

Why is this useful? The left half of the condition is used as a **shield** against executing the right half of the condition:

```
if ( ( x != 0 ) && ( z/x > 20 ) ) ...
```

Also works with `||`: If the left operand evaluates to **true**, the entire condition is true, and so we do not need to evaluate the right operand.

```
// ... suppose that: char c = 'q'
if ( ( c == 'q' ) || ( c == 'Q' ) ) ...      // the test c == 'Q' is not made
```

Increment/Decrement Operators

Increment/Decrement Operators:

Increment: `n = n + 1;` Java shorthand: `n++;`

Decrement: `n = n - 1;` Java shorthand: `n--;`

Example: Print "x bottles of beer..." for x from 10 down to 0.

```
int x = 10;
while ( x >= 0 ) {
    System.out.println( x + " bottles of beer on the wall" );
    x--;           // decrement x
}
```

Pre/Post Increment

Even shorter shorthand:

```
int x = 10;
while ( x >= 0 )
    System.out.println( (x--) + " bottles of beer on the wall" );
```

Do you want the value before or after incrementing/decrementing?

Pre-increment: `++x`

Post-increment: `x++`

Pre-decrement: `--x`

Post-decrement: `x--`

Example: `int x = 5; int y = 8;`

```
int z = 2 * (++x);    // x = 6, and so z = 12
int w = 5 + (y--);    // w = 5+8 = 13, and now y = 7
```

Assignment Operators

Many assignment statements update the value of a single variable:

Java provides convenient shorthand for these operations:

`<variable> <op>= <expression>`

is equivalent to:

`<variable> = <variable> <op> <expression>`

The above assignments can be written more succinctly as:

```
x = x * 2;      →    x *= 2;
y = y + 10;     →    y += 10;
z = z / 4;      →    z /= 4;
```

Operator Precedence

Operator Precedence:

Unary ops: `++x`, `-x`, `++x`, `--x`, `x++`, `x--`, `!x`

Multiplicative ops: `*`, `/`, `%`

Addition/Subtraction: `+`, `-`

Comparisons: `<`, `<=`, `>`, `>=`

Equality: `==`, `!=`

Logical ops: `&&`, `||` (`&&` is higher than `||`)

Assignments: `=`, `+=`, `-=`, `*=`, `/=`, etc.

Example:

```
if ( 2 * x++ < 5 * z + 3 && - w != x / 2 * y ) ...
```

Equivalent:

```
if ( ( ( 2*(x++)) < ((5*z) + 3) ) && ((-w) != ((x/2)*y)) ) ...
```

More on Operator Precedence

Style Suggestions:

- Add **spaces** and **parentheses** so the order of evaluation is clear:

Poor:

```
if ( 2 * x ++ < 5 * z + 3 && - w != x / 2 ) ...
```

Better:

```
if ( ( 2*(x++) < (5*z + 3) ) && (-w != x/2) ) ...
```

- Replace complex expressions with temporary variables:

Poor:

```
if ( (temp >= 97 && temp <= 99) || (systolic <= 120 && diastolic <= 80) ) ...
```

Better:

```
boolean temperatureIsOkay = (temp >= 97) && (temp <= 99);  
boolean bloodPressureIsOkay = (systolic <= 120) && (diastolic <= 80);  
if ( temperatureIsOkay || bloodPressureIsOkay ) ...
```

Type Casting

Casting: Assigning a variable/expression of one type to a variable of a different type is called **type casting**.)

Automatic (Implicit) Casting:

It is always safe to make an assignment to a variable of larger range.

`double ← float ← long ← int ← short ← byte`

Type Casting

Automatic Casting Examples:

```
int intVar = 12;           // no problem
double doubleVar = 5;      // okay
long longVar = intVar;     // okay
int intVar2 = longVar;     // illegal!
float floatVar1 = 2.3;     // illegal!
float floatVar2 = 2.3f;    // no problem
intVar = 2.0 * intVar;     // illegal!
```

Explicit Casting

Sometimes you need to cast one numeric type to another:

Explicit cast: Converts one numeric type explicitly into another.

(<desired type>) <expression>

Example 1:

```
int x = 23; int y = 4;
double d = x / y;           // d = 5 (integer division)
double e = (double) x / (double) y; // e = 5.75 (double division)
```

Example 2:

```
int degreesCelsius = ... ;
int degreesFahrenheit = (int) ( ( 9.0 / 5.0 ) * degreesCelsius ) + 32.0 );
```