

EX

double z = 6.5

int x ;

x = z ; // This narrowing conversion
 // is NOT allowed and will
 // cause a syntax error.

Narrowing conversions are likely
 to entail significant loss of data.

NOTE IN PAPER CASES, WIDENING
 CONVERSIONS CAN ALSO LOSE DATA
 (SER. P. 32).

TO CAUSE AN EXPLICIT CONVERSION
 BETWEEN TYPES, USE A CAST
 OPERATION:

(type).

EX.

double z = 6.5;

int x ;

x = (int) z

System.out.println(x); // prints 6

The precise rules for conversions between types in Java are complicated. SEE

[http://java.sun.com/docs/books/jls/](http://java.sun.com/docs/books/jls/second-edition/html/conversions.doc.html)

[second-edition/html/conversions.doc.html](http://java.sun.com/docs/books/jls/second-edition/html/conversions.doc.html)

One simple rule to remember is that conversions from floating point types to integer types truncate.

If you want to round instead of truncate use `Math.round()`

EX.

```

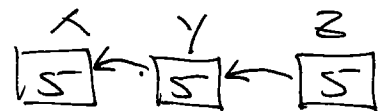
double z = 6.6;
int x, y;
x = (int) z;
y = (int) Math.round(z);
System.out.println(x); // prints 6
System.out.println(y); // prints 7

```

NOTE `round()` returns a long, so a narrowing conversion to `int` is required.

NOTE THE ASSIGNMENT STATEMENT $x = y$ IS ITSELF AN EXPRESSION WHOSE VALUE IS THE VALUE ASSIGNED

EX $\text{int } x, y, z = 5;$



$$x = (y = z);$$

= ASSOCIATED FROM RIGHT TO LEFT SO ONE CAN ALSO DO

$$x = y = z;$$

COMPOUND ASSIGNMENT

$a += b$	same as	$a = a + b$
$a -= b$	" "	$a = a - b$
$a *= b$	" "	$a = a * b$
$a /= b$	" "	$a = a / b$
$a \% = b$	" "	$a = a \% b$

NOTE WITH INTEGER OPERANDS, / AND % ARE INTEGER DIVISION, ? REMAINDER.

EX $7/5$ EVALUATES TO 1
 $7\%5$ EVALUATES TO 2

SINCE $7 = 1 \cdot 5 + 2$

Ex.

```
double seconds = 3725.7;
int h, m, s;
```

```
s = (int) Math.round(seconds); // s = 3726
```

```
m = s / 60; // m = 62
```

```
s = s % 60; // s = 6. EQUIV: s % 60;
```

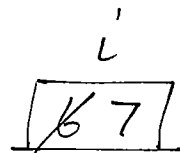
```
h = m / 60; // h = 1
```

```
m = m % 60; // m = 2 EQUIV: m % 60;
```

```
System.out.println(h + " ... m ... s");
```

OFTEN ONE WISHES TO JUST ADD 1 TO A VARIABLE, THIS IS CALLED AN INCREMENT OPERATION.

```
i = i + 1;
```



WE CAN DO THIS WITH COMPOUND ASSIGNMENT: $i += 1$; OR WITH THE AUTO-INCREMENT OPERATION

```
i++;
```

LIKewise FOR DECREMENT:

```
i = i - 1; OR i -= 1; OR i--
```

THE AUTO-INCREMENT AND AUTO-DECREMENT OPERATIONS HAVE TWO FORMS.

POSTFIX: $i++$, $i--$

PREFIX: $++i$, $--i$

THE DIFFERENCE IS IN THE VALUE OF THE EXPRESSIONS $i++$ VS. $++i$ ITSELF!

- $i++$ FIRST EVALUATES TO THE OLD VALUE OF i , THEN INCREMENTS i .
- $++i$ FIRST INCREMENTS i THEN EVALUATES TO THE NEW VALUE OF i .

EX.

int $i=6, a, b, c, d;$

$a = i++;$

$b = ++i;$

$c = i--;$

$d = --i;$

TRACE!

i	a	b	c	d
6	-	-	-	-
7	6	-	-	-
8	6	8	-	-
7	6	8	8	-
6	6	8	8	6

OPERATOR PRECEDENCE GIVE THE HIERARCHY THAT DETERMINES THE ORDER IN WHICH OPERATIONS ARE PERFORMED.

$$a \text{ op}_1 b \text{ op}_2 c$$

IF op_1 HAS HIGHER PRECEDENCE, THIS IS EQUIVALENT TO

$$(a \text{ op}_1 b) \text{ op}_2 c$$

WHILE IF op_2 HAS HIGHER PRECEDENCE IT IS EQUIVALENT TO

$$a \text{ op}_1 (b \text{ op}_2 c)$$

IF op_1 AND op_2 HAVE EQUAL PRECEDENCE THEY ARE EVALUATED FROM LEFT TO RIGHT
i.e.

$$(a \text{ op}_1 b) \text{ op}_2 c$$

EX $7 + 5 * 3$ + LOWER THAN *

VALUE: 22

EX $100 / 5 * 2$ / EQUAL TO *

VALUE: 40

SEE JAVA DOCUMENTATION OR APP. B
OF TEXT FOR DETAILS ON OPERATOR
PRECEDENCE. IF THERE IS EVER
ANY DOUBT INSIST PARENTHESES.

READ & DO

EXERCISES P. 40

STATEMENTS & CONTROL FLOW

All EXECUTABLE STATEMENTS in JAVA (i.e. those other than DECLARATIVE STATEMENTS.) ARE OF THREE KINDS

- (1) SEQUENTIAL
- (2) CONDITIONAL (ALSO BRANCHING)
- (3) ITERATIVE (ALSO LOOPING)

THESE TYPES PERTAIN TO THE FLOW OF CONTROL IN A PROGRAM, i.e. AFTER SOME STATEMENT IS EXECUTED, WHICH STATEMENT WILL BE EXECUTED NEXT.

SEQUENTIAL EXECUTION IS THE DEFAULT CONTROL FLOW, i.e. AFTER A STATEMENT IS EXECUTED, CONTROL GOES TO THE FOLLOWING STATEMENT IN THE PROGRAM.

All our programs up to this point HAVE USED ONLY SEQUENTIAL STATEMENTS. SUCH PROGRAMS ARE SOMETIMES CALLED STRAIGHT LINE PROGRAMS

CONDITIONAL STATEMENTS FIRST TEST THE VALUE OF SOME EXPRESSION (USUALLY A BOOLEAN EXPRESSION), AND THE NEXT INSTRUCTION DEPENDS ON THE VALUE.

ITERATIVE STATEMENTS CAUSE A BLOCK OF STATEMENTS TO BE REPEATED UNTIL SOME LOGICAL CONDITION BECOMES TRUE OR FALSE.

BOTH CONDITIONAL AND ITERATIVE OPERATIONS RELY ON THE EVALUATION OF LOGICAL EXPRESSIONS, i.e. EXPRESSIONS WHOSE VALUE IS A BOOLEAN TRUE OR FALSE.

TO UNDERSTAND THESE EXPRESSIONS WE MUST REVIEW RELATIONAL AND LOGICAL OPERATORS.

<u>RELATIONAL OP.</u>	<u>NAME</u>	<u>EX</u>	<u>VAL</u>
<	LESS THAN	$5 < 10$	true
>	GREATER THAN	$5 > 10$	false
==	EQUAL	$5 == 10$	false
<=	LESS THAN OR EQ.	$5 <= 10$	true
>=	GREATER THAN OR EQ.	$5 >= 10$	false
!=	NOT EQ.	$5 != 10$	true

<u>LOGICAL OP</u>	<u>Name</u>	<u>EXAMPLE</u>	<u>VALUE</u>
&&	and	(1<2) && (3==4)	FALSE
	or	(1<2) (3==4)	true
!	not	!(1<2)	false

LET a AND b BE BOOLEAN VARIABLES. THE FOLLOWING TRUTH TABLES DEFINE THE ABOVE OPERATIONS.

a	b	a && b	a b
F	F	F	F
F	T	F	T
T	F	F	T
T	T	T	T

a	!a
F	T
T	F

NOTE || DENOTES THE INCLUSIVE OR i.e. a || b SAYS "a or b, or possibly BOTH", AS OPPOSED TO THE EXCLUSIVE OR WHICH SAYS "a or b, BUT NOT BOTH". SOME OTHER COMMON NOTATIONS FOR THESE OPERATORS ARE!

<u>English</u>	<u>Java</u>	<u>math</u>	<u>CIRCUIT THEORY</u>
a and b	a && b	$a \wedge b$	$a \cdot b$
a or b	a b	$a \vee b$	$a + b$
not a	!a	$\neg a$	\overline{a}